

Listas Dinâmicas em C

INF0286 | INF0447 – Algoritmos e Estruturas de Dados I
(versão preliminar)

Prof. Me. Raphael Guedes

raphaelguedes@ufg.br

2024

INF

INSTITUTO DE
INFORMÁTICA



Listas Encadeadas

- Usam a criação dos nós por encadeamento.
- Podem ser:
 - simplesmente encadeadas;
 - simplesmente encadeadas com nó descritor;
 - circulares.
 - duplamente encadeadas;
- É possível realizar combinações entre os variados tipos de listas.



Listas Simplemente Encadeadas

Lista Simp. Encadeada: estrutura

Arquivo ListaDinEncad.h

```
01 struct aluno{  
02     int matricula;  
03     char nome[30];  
04     float n1,n2,n3;  
05 };  
06 typedef struct elemento* Lista;  
07  
08 Lista* cria_lista();  
09 void libera_lista(Lista* li);  
10 int insere_lista_final(Lista* li, struct aluno al);  
11 int insere_lista_inicio(Lista* li, struct aluno al);  
12 int insere_lista_ordenada(Lista* li, struct aluno al);  
13 int remove_lista(Lista* li, int mat);  
14 int remove_lista_inicio(Lista* li);  
15 int remove_lista_final(Lista* li);  
16 int tamanho_lista(Lista* li);  
17 int lista_vazia(Lista* li);  
18 int lista_cheia(Lista* li);  
19 int busca_lista_mat(Lista* li, int mat, struct aluno *al);  
20 int busca_lista_pos(Lista* li, int pos, struct aluno *al);
```

Lista Simp. Encadeada: estrutura

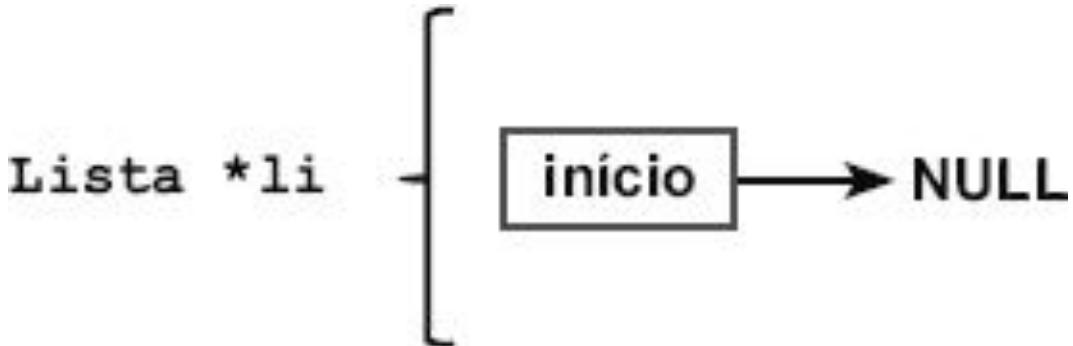
Arquivo ListaDinEncad.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include "ListaDinEncad.h" //inclui os protótipos
04 //Definição do tipo lista
05 struct elemento{
06     struct aluno dados;
07     struct elemento *prox;
08 };
09 typedef struct elemento Elem;
```

Lista Simp. Encadeada: criação

Criando uma lista

```
01  Lista* cria_lista(){
02      Lista* li = (Lista*) malloc(sizeof(Lista));
03      if(li != NULL)
04          *li = NULL;
05      return li;
06  }
```



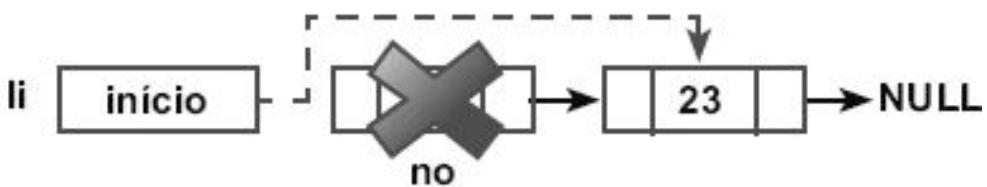
Lista Simp. Encadeada: destruição

Lista inicial



Passo 1:

```
no = *li;  
*li = (*li)->prox;  
free(no);
```



Passo 2:

```
no = *li;  
*li = (*li)->prox;  
free(no);
```



Fim:

```
no == NULL
```



Lista Simp. Encadeada: destruição

Destruindo uma lista

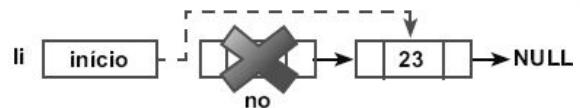
```
01 void libera_lista(Lista* li) {  
02     if(li != NULL){  
03         ELEM* no;  
04         while((*li) != NULL){  
05             no = *li;  
06             *li = (*li)->prox;  
07             free(no);  
08         }  
09         free(li);  
10     }  
11 }
```

Lista inicial



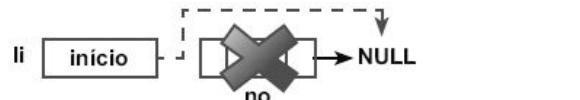
Passo 1:

```
no = *li;  
*li = (*li)->prox;  
free(no);
```



Passo 2:

```
no = *li;  
*li = (*li)->prox;  
free(no);
```



Fim:

```
no == NULL
```



Lista Simp. Encadeada: tamanho

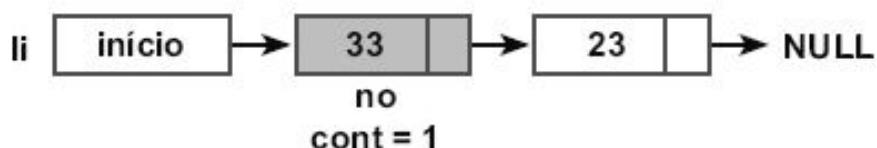
Lista inicial:

```
cont = 0;  
no = *li;
```



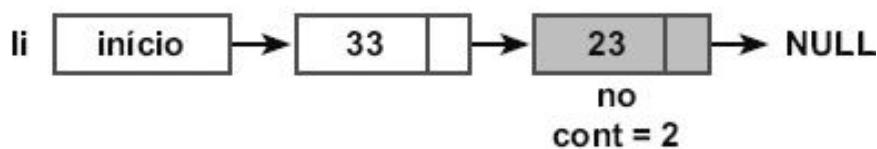
Passo 1:

```
cont++;  
no = no->prox;
```



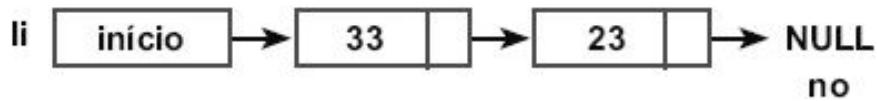
Passo 2:

```
cont++;  
no = no->prox;
```



Fim:

```
no == NULL
```



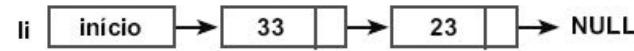
Lista Simp. Encadeada: tamanho

Tamanho da lista

```
01 int tamanho_lista(Lista* li){  
02     if(li == NULL)  
03         return 0;  
04     int cont = 0;  
05     ELEM* no = *li;  
06     while(no != NULL) {  
07         cont++;  
08         no = no->prox;  
09     }  
10     return cont;  
11 }
```

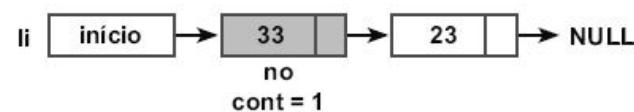
Lista inicial:

```
cont = 0;  
no = *li;
```



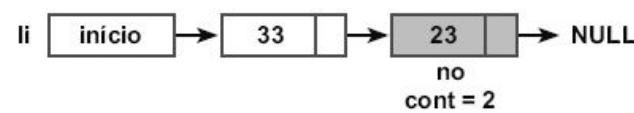
Passo 1:

```
cont++;  
no = no->prox;
```



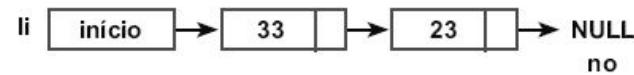
Passo 2:

```
cont++;  
no = no->prox;
```



Fim:

```
no == NULL
```



Lista Simp. Encadeada: lista cheia

Não teríamos uma abordagem melhor?

E se tentarmos alocar um nó, resolve?

Retornando se a lista está cheia

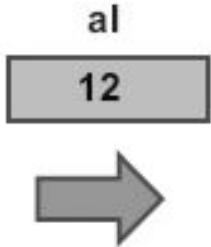
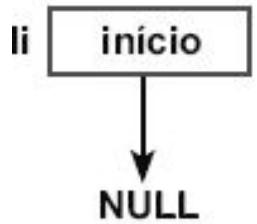
```
01  int lista_cheia(Lista* li) {  
02      return 0;  
03  }
```

Lista Simp. Encadeada: lista vazia

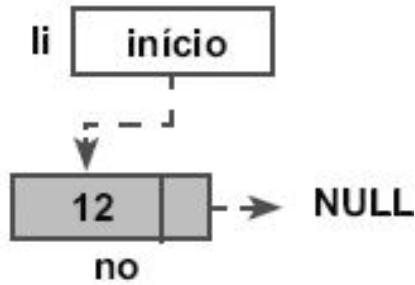
Retornando se a lista está vazia

```
01 int lista_vazia(Lista* li) {  
02     if(li == NULL)  
03         return 1;  
04     if(*li == NULL)  
05         return 1;  
06     return 0;  
07 }
```

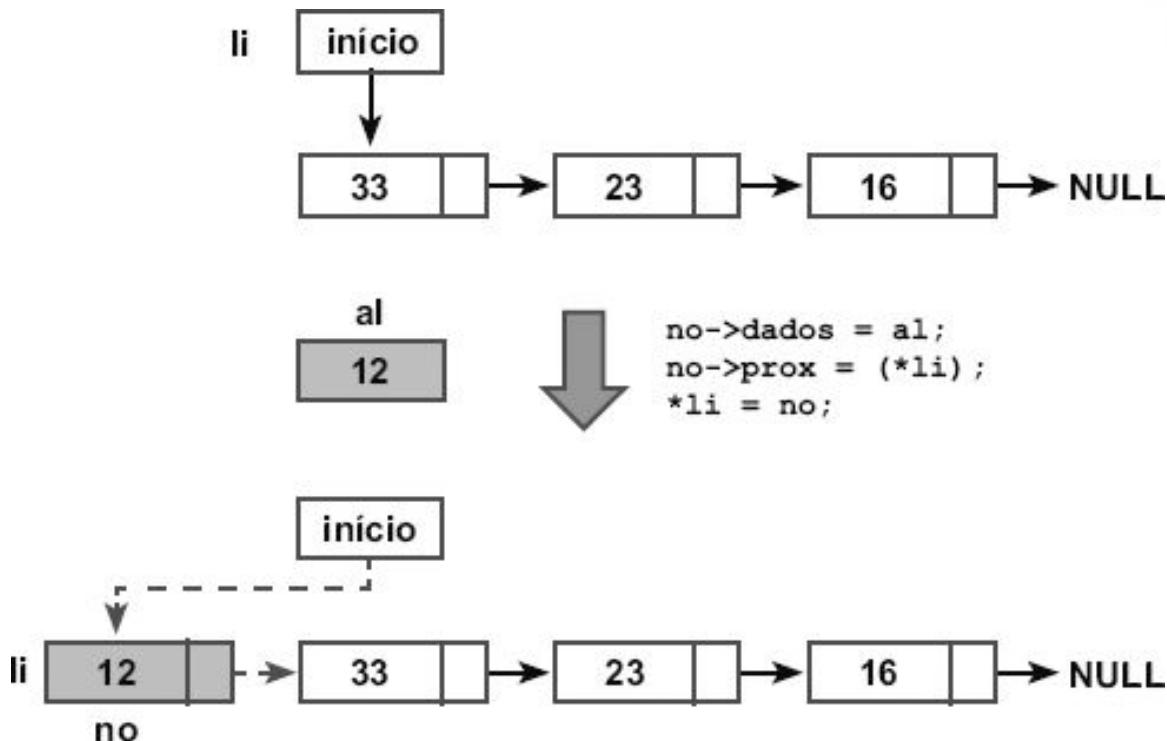
Lista Simp. Encadeada: inserção



```
no->dados = al;  
no->prox = (*li);  
*li = no;
```



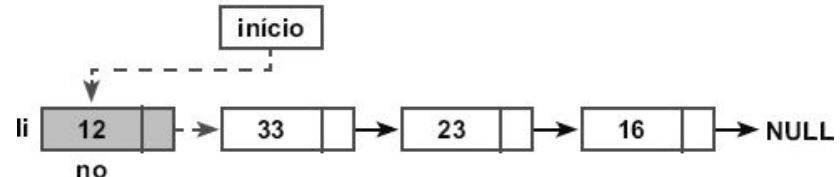
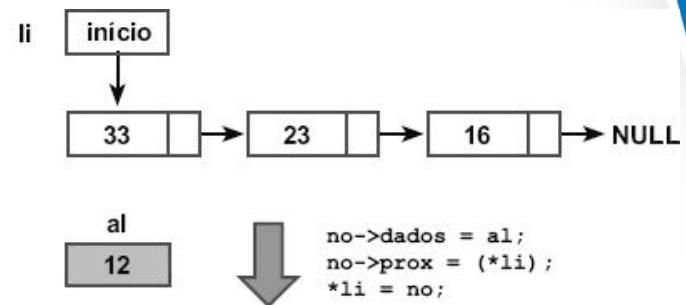
Lista Simp. Encadeada: inserção início



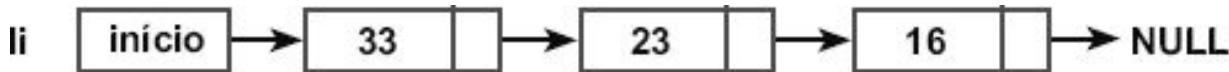
Lista Simp. Encadeada: inserção início

Inserindo um elemento no início da lista

```
01 int insere_lista_inicio(Lista* li, struct aluno al){  
02     if(li == NULL)  
03         return 0;  
04     Elem* no;  
05     no = (ELEM*) malloc(sizeof(ELEM));  
06     if(no == NULL)  
07         return 0;  
08     no->dados = al;  
09     no->prox = (*li);  
10     *li = no;  
11     return 1;  
12 }
```

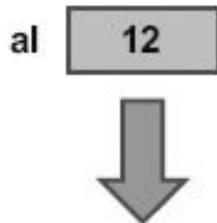


Lista Simp. Encadeada: inserção final



Busca onde inserir:

```
aux = *li;  
while (aux->prox != NULL) {  
    aux = aux->prox;  
}
```



Insere depois de "aux":

```
no->dados = al;  
no->prox = NULL;  
aux->prox = no;
```

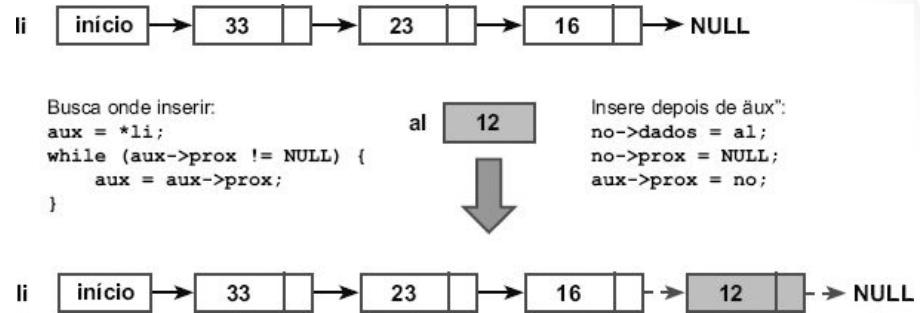


Lista Simp. Encadeada: inserção final

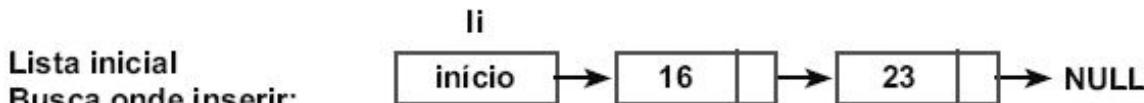
Inserindo um elemento no final da lista

```

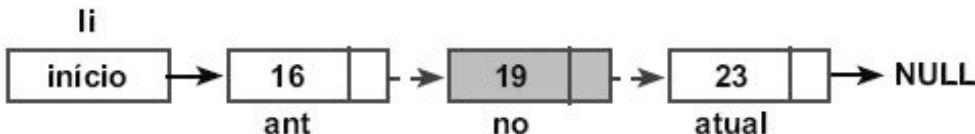
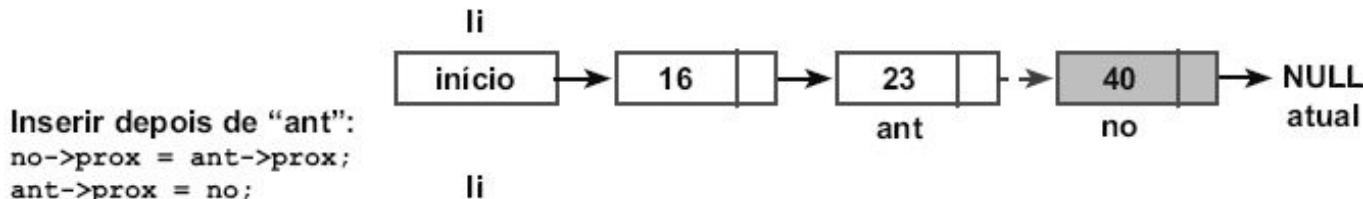
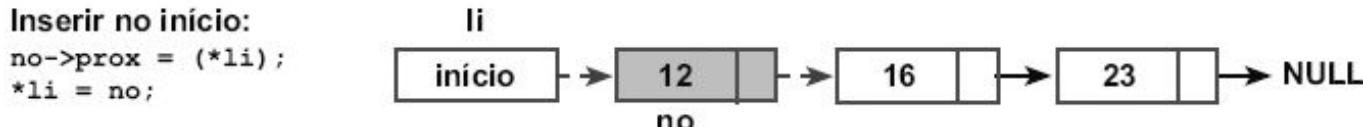
01  int insere_lista_final(Lista* li, struct aluno al){
02      if(li == NULL)
03          return 0;
04      ELEM *no;
05      no = (ELEM*) malloc(sizeof(ELEM));
06      if(no == NULL)
07          return 0;
08      no->dados = al;
09      no->prox = NULL;
10     if((*li) == NULL){//lista vazia: insere inicio
11         *li = no;
12     }else{
13         ELEM *aux;
14         aux = *li;
15         while(aux->prox != NULL){
16             aux = aux->prox;
17         }
18         aux->prox = no;
19     }
20     return 1;
21 }
```



Lista Simp. Encadeada: inserção meio



```
atual = *li;  
while(atual != NULL && atual->dados.matricula < al.matricula){  
    ant = atual;  
    atual = atual->prox;  
}
```



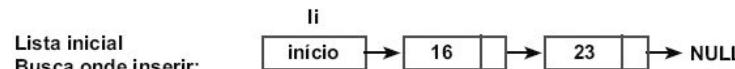
Lista Simp. Encadeada: inserção meio

Inserindo um elemento de forma ordenada na lista

```

01 int insere_lista_ordenada(Lista* li, struct aluno al){
02     if(li == NULL)
03         return 0;
04     Elemt *no;
05     no = (Elemt*) malloc(sizeof(Elemt));
06     if(no == NULL)
07         return 0;
08     no->dados = al;
09     if((*li) == NULL){//lista vazia: insere inicio
10         no->prox = NULL;
11         *li = no;
12         return 1;
13     }
14     else{
15         Elemt *ant, *atual = *li;
16         while(atual != NULL &
17               atual->dados.matricula < al.matricula){
18             ant = atual;
19             atual = atual->prox;
20         }
21         if(atual == *li){//insere inicio
22             no->prox = (*li);
23             *li = no;
24         }else{
25             no->prox = atual;
26             ant->prox = no;
27         }
28         return 1;
29     }
}

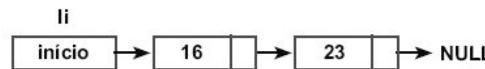
```



Busca onde inserir:

```

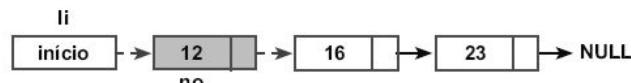
atual = *li;
while(atual != NULL && atual->dados.matricula < al.matricula){
    ant = atual;
    atual = atual->prox;
}
-----
```



Inserir no inicio:

```

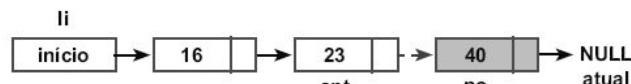
no->prox = (*li);
*li = no;
-----
```



Inserir depois de "ant":

```

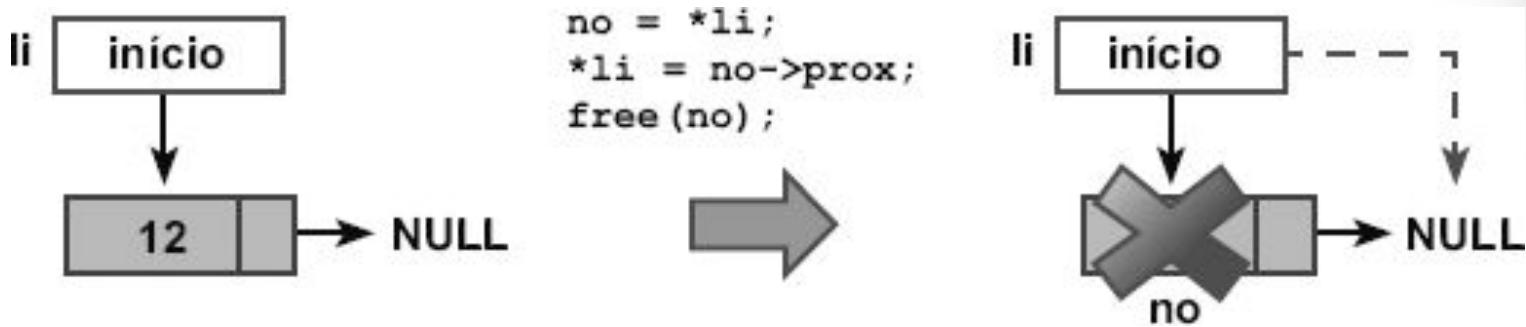
no->prox = ant->prox;
ant->prox = no;
-----
```



início

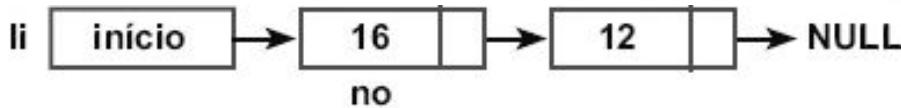
início → 16 → 19 → 23 → NULL

Lista Simp. Encadeada: remoção



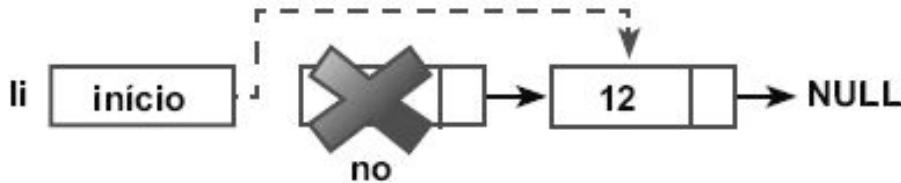
Lista Simp. Encadeada: remoção início

Lista inicial

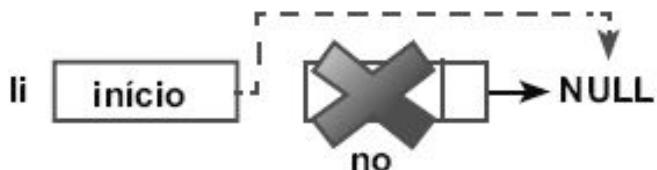


```
*li = no->prox;  
free(no);
```

Se a lista possui mais de um elemento, o `início` aponta para o segundo.



Se a lista possui um único elemento, ela fica vazia.



Lista Simp. Encadeada: remoção início

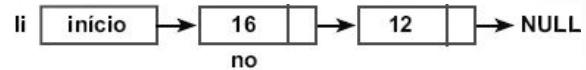
Removendo um elemento do início da lista

```

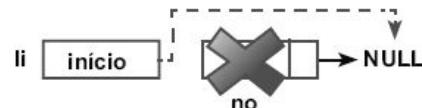
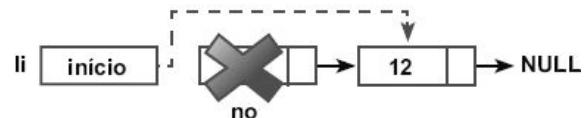
01 int remove_lista_inicio(Lista* li){
02     if(li == NULL)
03         return 0;
04     if((*li) == NULL)//lista vazia
05         return 0;
06
07     ELEM *no = *li;           Lista inicial
08     *li = no->prox;
09     free(no);
10     return 1;
11 }
```

Se a lista possui mais de um elemento, o início aponta para o segundo.

Se a lista possui um único elemento, ela fica vazia.



`*li = no->prox;
free(no);`

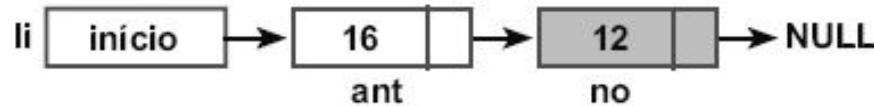


Lista Simp. Encadeada: remoção final

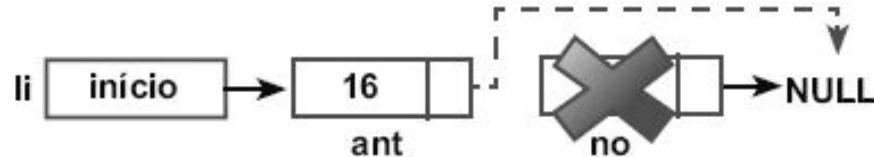
Lista inicial

Busca o último elemento:

```
no = *li;  
while(no->prox != NULL) {  
    ant = no;  
    no = no->prox;  
}
```



ant->prox = no->prox;
free(no);



Se a lista possui mais de um elemento, “ant” aponta para “NULL”.

Se “no” é o único elemento da lista, a lista fica vazia.



Lista Simp. Encadeada: remoção final

Removendo um elemento do final da lista

```

01  int remove_lista_final(Lista* li){
02      if(li == NULL)
03          return 0;
04      if((*li) == NULL)//lista vazia
05          return 0;
06
07      ELEM *ant, *no = *li;
08      while(no->prox != NULL){
09          ant = no;
10          no = no->prox;
11      }
12
13      if(no == (*li))//remover o primeiro?
14          *li = no->prox;
15      else
16          ant->prox = no->prox;
17      free(no);
18      return 1;
19  }
```

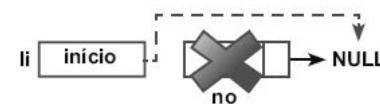
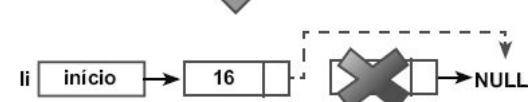
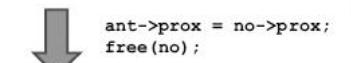
Lista inicial
Busca o último elemento:

```

no = *li;
while(no->prox != NULL){
    ant = no;
    no = no->prox;
}
```

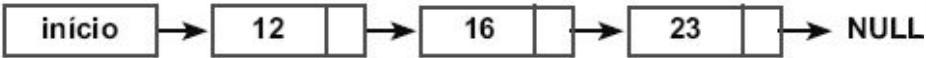
Se a lista possui mais de um elemento, "ant" aponta para "NULL".

Se "no" é o único elemento da lista, a lista fica vazia.



Lista Simp. Encadeada: remoção meio

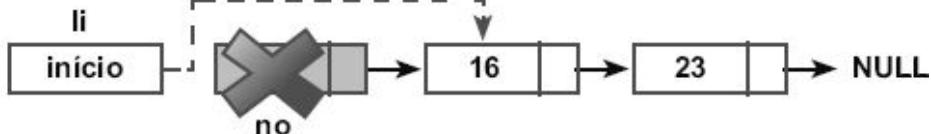
Lista inicial

Busca qual remover:  **início** → 12 → 16 → 23 → NULL

```
no = *li;
while(no != NULL && no->dados.matricula != mat){
    ant = no;
    no = no->prox;
}
```

Remover do início:

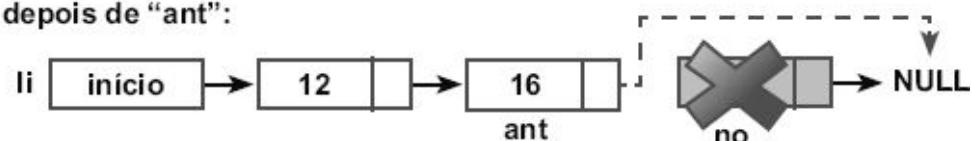
```
*li = no->prox;
free(no);
```



Remover do meio ou do fim

significa remover depois de “ant”:

```
ant->prox = no->prox;
free(no);
```



Lista Simp. Encadeada: remoção meio

Removendo um elemento específico da lista

```

01  int remove_lista(Lista* li, int mat) {
02      if(li == NULL)
03          return 0;
04      if((*li) == NULL)//lista vazia
05          return 0;
06      Elem *ant, *no = *li;
07      while(no != NULL && no->dados.matricula != mat) {
08          ant = no;
09          no = no->prox;
10      }
11      if(no == NULL)//não encontrado
12          return 0;
13
14      if(no == *li)//remover o primeiro?
15          *li = no->prox;
16      else
17          ant->prox = no->prox;
18      free(no);
19      return 1;
20  }

```

Lista inicial

Busca qual remover:

```

    li
    inicio → 12 → 16 → 23 → NULL
    no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }

```

Remover do início:

```

    li
    inicio → 16 → 23 → NULL
    *li = no->prox;
    free(no);

```

Remover do meio ou do fim
significa remover depois de "ant":

```

    li
    inicio → 12 → 16 → 23 → NULL
    ant->prox = no->prox;
    free(no);

```

Lista Simp. Encadeada: busca posição

Busca pela posição do elemento:

```
no = *li;
int i = 1;
while(no != NULL && i < pos){
    no = no->prox;
    i++;
}
```

Verifica se a posição foi encontrada e a retorna:

```
if(no == NULL) return 0;
else{
    *al = no->dados;
    return 1;
}
```



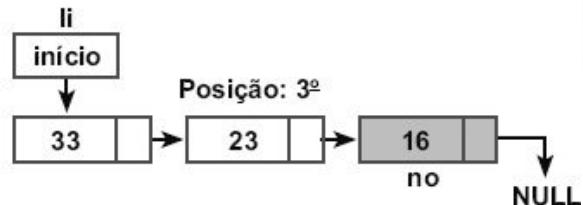
Lista Simp. Encadeada: busca posição

Busca um elemento por posição

```
01 int busca_lista_pos(Lista* li, int pos, struct aluno *al){  
02     if(li == NULL || pos <= 0)  
03         return 0;  
04     Elemt *no = *li;  
05     int i = 1;  
06     while(no != NULL && i < pos){  
07         no = no->prox;  
08         i++;  
09     }  
10     if(no == NULL)  
11         return 0;  
12     else{  
13         *al = no->dados;  
14         return 1;  
15     }  
16 }
```

Busca pela posição do elemento:

```
no = *li;  
int i = 1;  
while(no != NULL && i < pos){  
    no = no->prox;  
    i++;  
}  
Verifica se a posição foi  
encontrada e a retorna:  
if(no == NULL) return 0;  
else{  
    *al = no->dados;  
    return 1;  
}
```



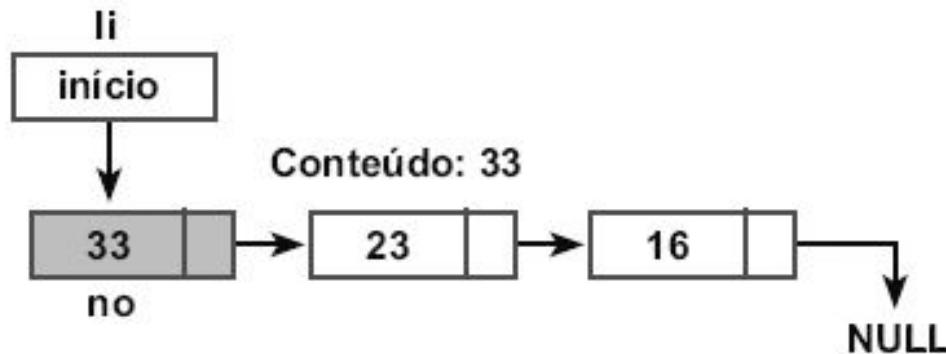
Lista Simp. Encadeada: busca conteúdo

Busca pelo conteúdo do elemento:

```
no = *li;  
while(no != NULL && no->dados.matricula != mat)  
    no = no->prox;
```

Verifica se o elemento foi encontrado e o retorna:

```
if(no == NULL) return 0;  
else{  
    *al = no->dados;  
    return 1;  
}
```



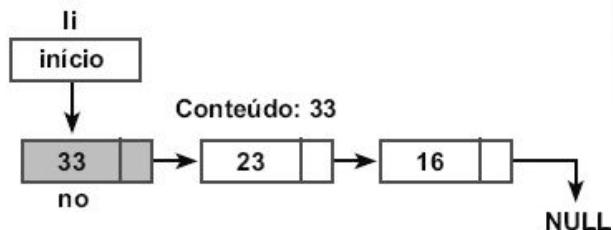
Lista Simp. Encadeada: busca conteúdo

Busca um elemento por conteúdo

```
01 int busca_lista_mat(Lista* li, int mat, struct aluno *al){  
02     if(li == NULL)  
03         return 0;  
04     Elemt *no = *li;  
05     while(no != NULL && no->dados.matricula != mat){  
06         no = no->prox;  
07     }  
08     if(no == NULL)  
09         return 0;  
10     else{  
11         *al = no->dados;  
12         return 1;  
13     }  
14 }
```

Busca pelo conteúdo do elemento:
no = *li;
while(no != NULL && no->dados.matricula != mat)
 no = no->prox;

Verifica se o elemento foi encontrado e o retorna:
if(no == NULL) return 0;
else{
 *al = no->dados;
 return 1;
}

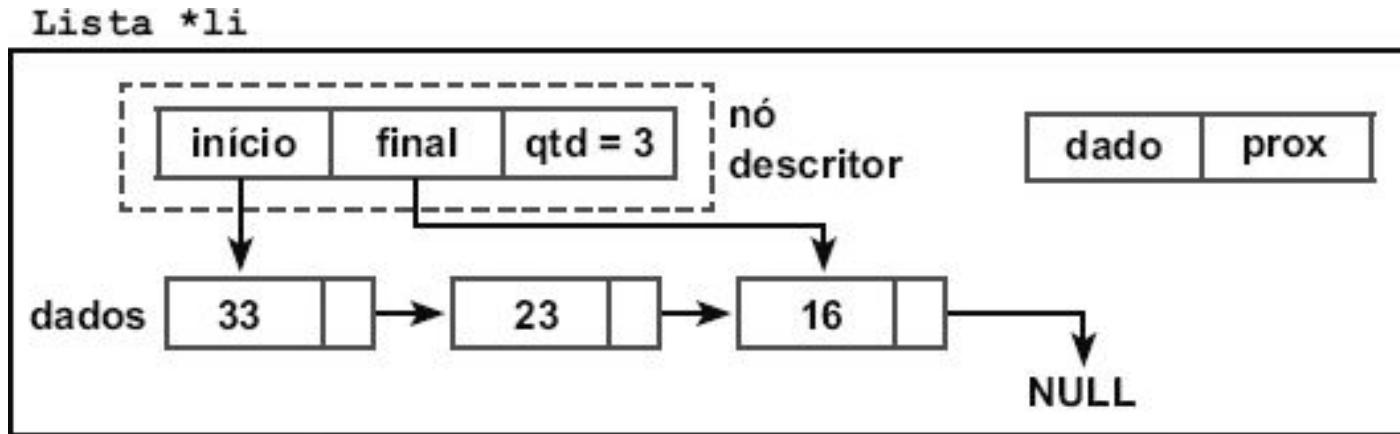




Listas Simplesmente Encadeadas com Nó Descriptor

Lista com Nó Descritor: estrutura

- Um nó descritor é uma estrutura que possui um campo que aponta para o primeiro elemento da lista, além de outras informações.
 - Lembra da fila dinâmica?



Lista com Nó Descritor

Lista *li;

Arquivo ListaDinEncadDesc.h

```
01 struct aluno{  
02     int matricula;  
03     char nome[30];  
04     float n1,n2,n3;  
05 };  
06 typedef struct descritor Lista;  
07  
08 Lista* cria_lista();  
09 void libera_lista(Lista* li);  
10 int insere_lista_final(Lista* li, struct aluno al);  
11 int insere_lista_inicio(Lista* li, struct aluno al);  
12 int remove_lista_inicio(Lista* li);  
13 int remove_lista_final(Lista* li);  
14 int tamanho_lista(Lista* li);  
15 int lista_vazia(Lista* li);  
16 int lista_cheia(Lista* li);  
17 int busca_lista_mat(Lista* li, int mat, struct aluno *al);  
18 int busca_lista_pos(Lista* li, int pos, struct aluno *al);
```

Lista com Nô Descritor

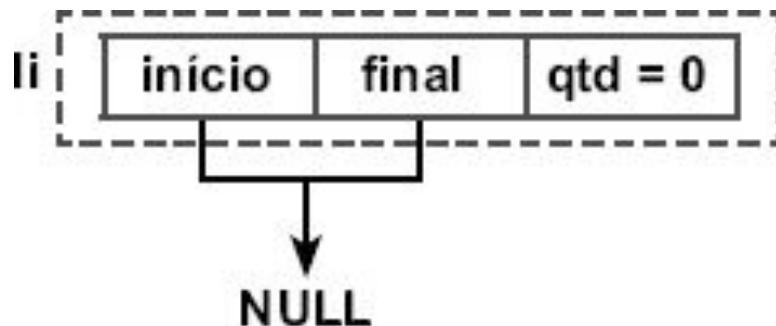
Arquivo ListaDinEncadDesc.c

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include "ListaDinEncadDesc.h" //inclui os protótipos
04  //Definição do tipo lista
05  struct elemento{
06      struct aluno dados;
07      struct elemento *prox;
08  };
09  typedef struct elemento Elem;
10
11 //Definição do Nô Descritor
12 struct descritor{
13     struct elemento *inicio;
14     struct elemento *final;
15     int tamanho;
16 }
```

Lista com Nô Descritor: criação

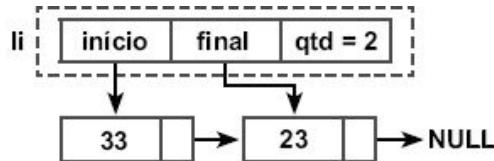
Criando uma lista

```
01  Lista* cria_lista(){
02      Lista* li = (Lista*) malloc(sizeof(Lista));
03      if(li != NULL){
04          li->inicio = NULL;
05          li->final = NULL;
06          li->tamanho = 0;
07      }
08      return li;
09 }
```



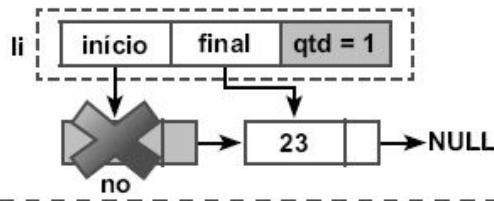
Lista com Nó Descritor: destruição

Lista inicial



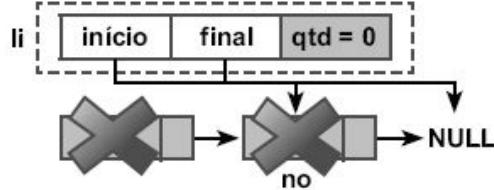
Passo 1:

```
no = li->inicio;  
li->inicio = li->inicio->prox;  
free(no);
```



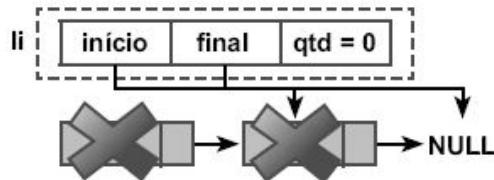
Passo 2:

```
no = li->inicio;  
li->inicio = li->inicio->prox;  
free(no);
```



Fim:

```
(li->inicio) == NULL  
free(li);
```



Lista com Nó Descritor: destruição

Destruindo uma lista

```

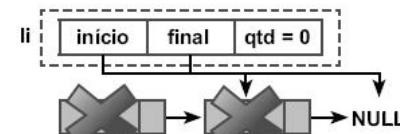
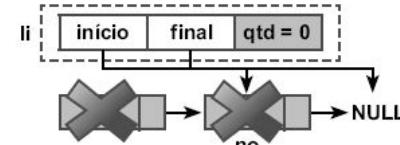
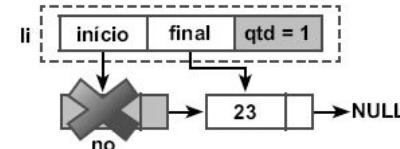
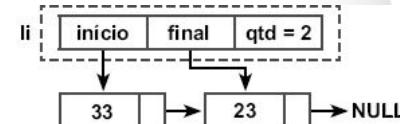
01 void libera_lista(Lista* li){
02     if(li != NULL){
03         Elem* no;
04         while((li->inicio) != NULL) {
05             no = li->inicio;
06             li->inicio = li->inicio->prox;
07             free(no);
08         }
09         free(li);
10     }
11 }
```

Lista inicial

Passo 1:
`no = li->inicio;`
`li->inicio = li->inicio->prox;`
`free(no);`

Passo 2:
`no = li->inicio;`
`li->inicio = li->inicio->prox;`
`free(no);`

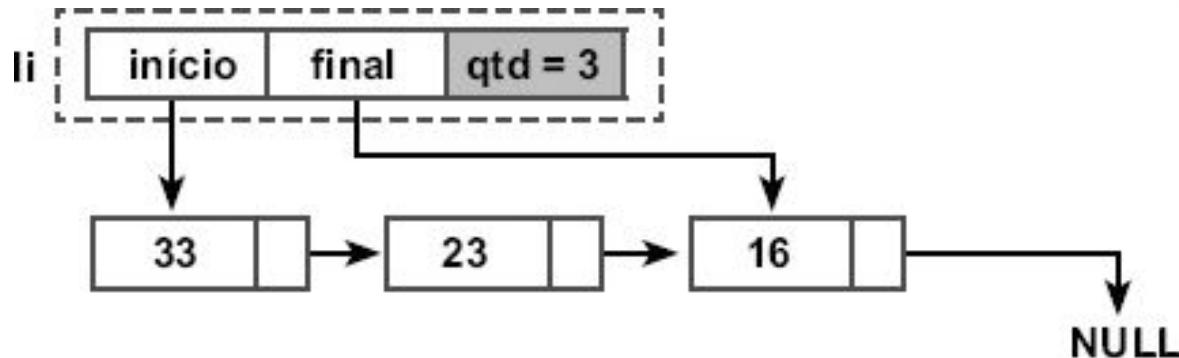
Fim:
`(li->inicio) == NULL`
`free(li);`



Lista com Nô Descritor: tamanho

Tamanho da lista

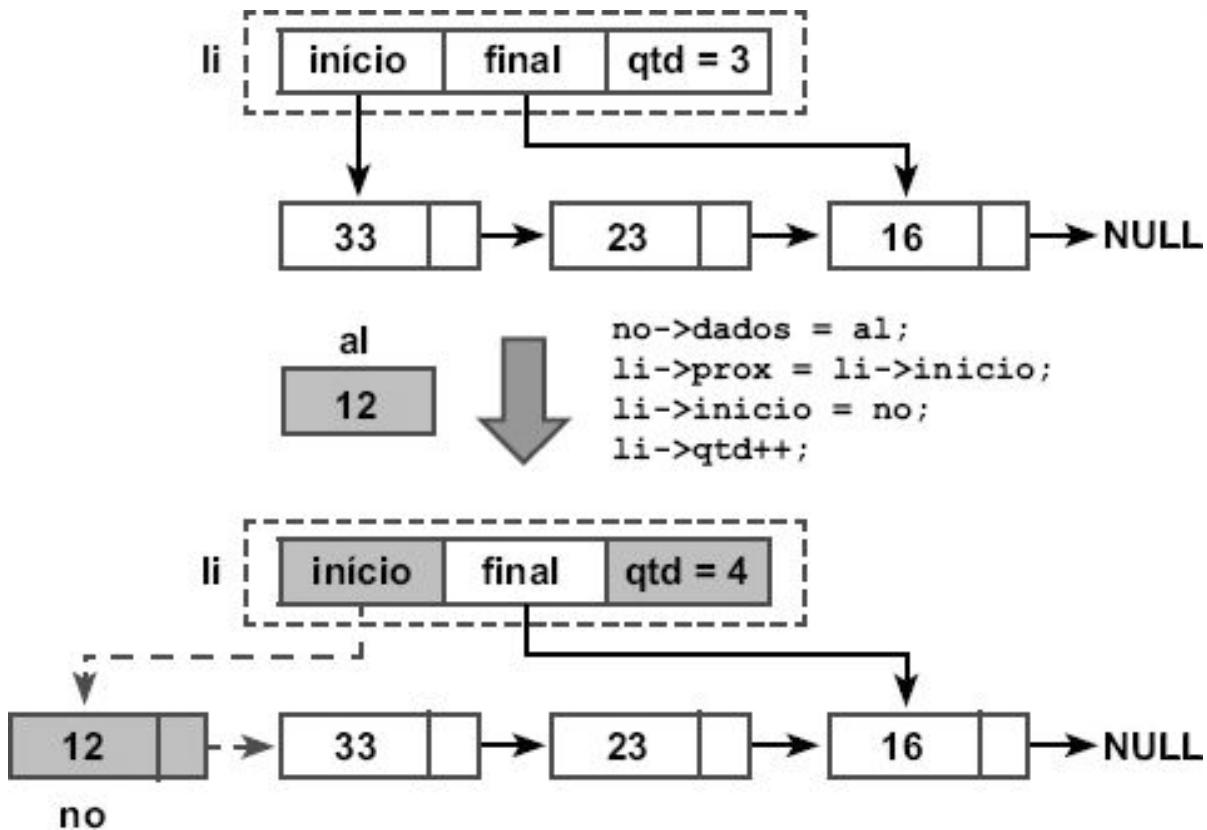
```
01 int tamanho_lista(Lista* li) {  
02     if(li == NULL)  
03         return 0;  
04     return li->tamanho;  
05 }
```



Lista com Nό Descritor: inserção



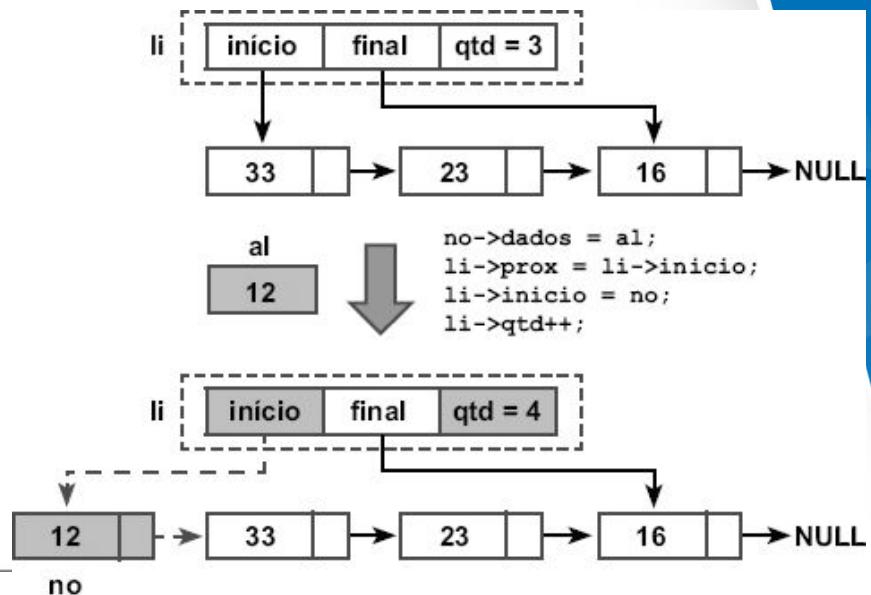
Lista com Nó Descritor: inserção início



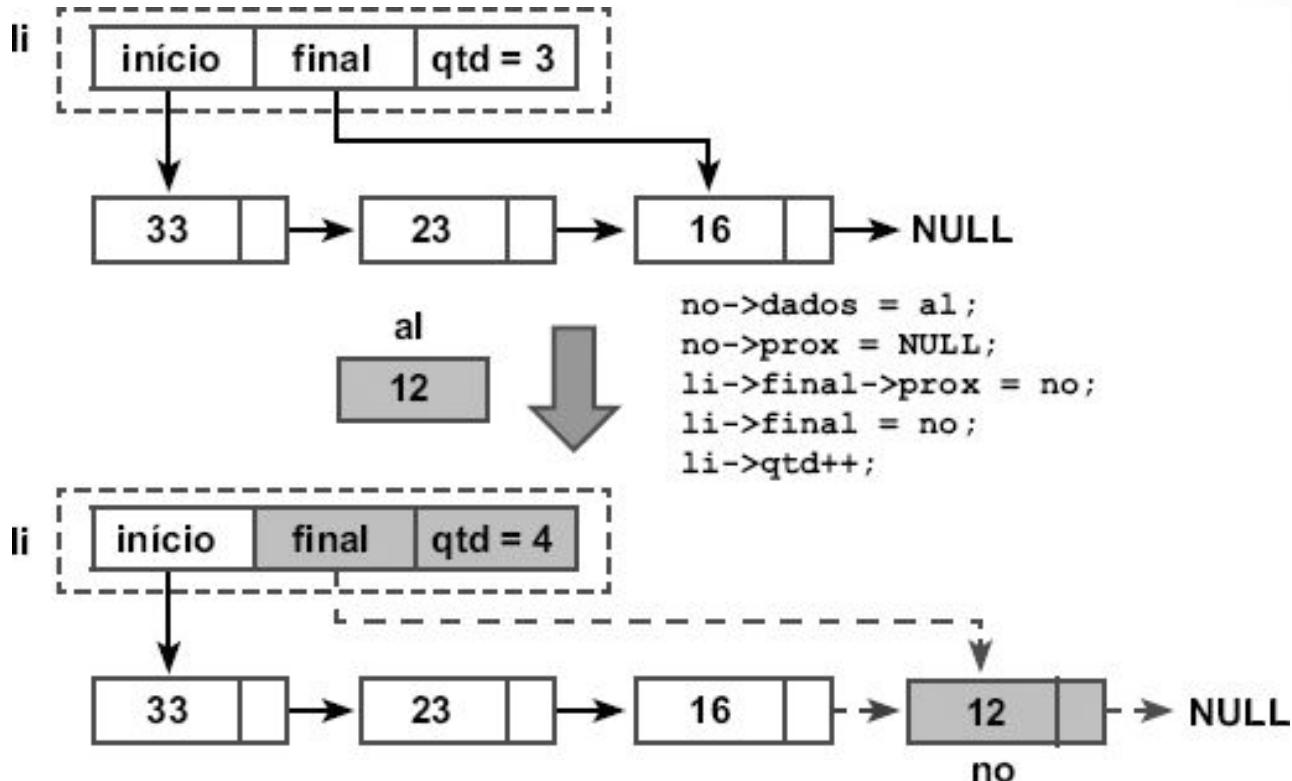
Lista com Nó Descritor: inserção início

Inserindo um elemento no início da lista

```
01 int insere_lista_inicio(Lista* li, struct aluno al){  
02     if(li == NULL)  
03         return 0;  
04     Elemt* no;  
05     no = (Elemt*) malloc(sizeof(Elemt));  
06     if(no == NULL)  
07         return 0;  
08     no->dados = al;  
09     no->prox = li->inicio;  
10     if(li->inicio == NULL)  
11         li->final = no;  
12     li->inicio = no;  
13     li->tamanho++;  
14     return 1;  
15 }
```



Lista com Nó Descritor: inserção final

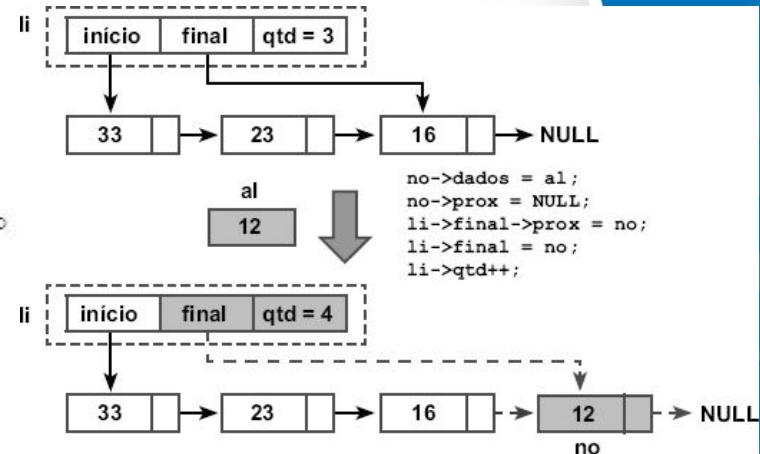


Lista com Nó Descritor: inserção final

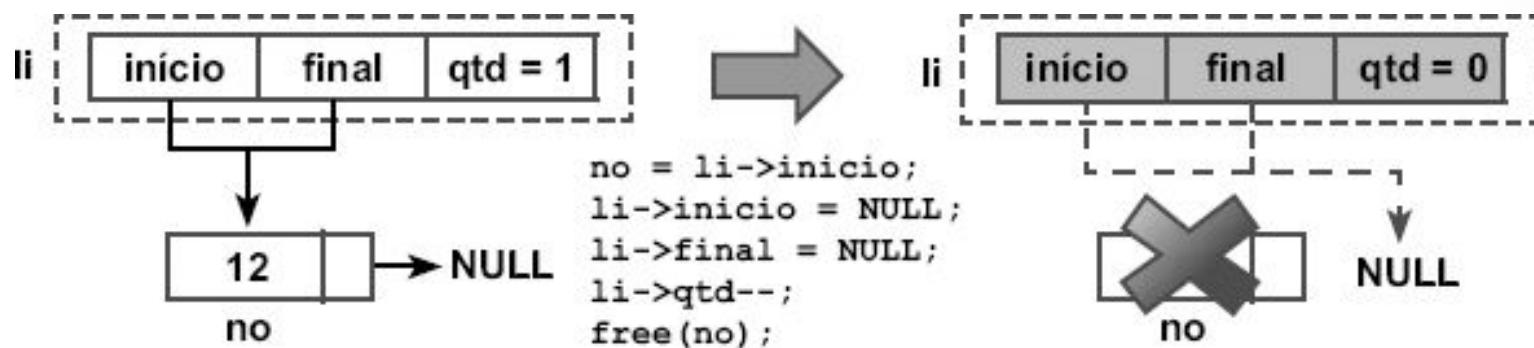
Inserindo um elemento no final da lista

```

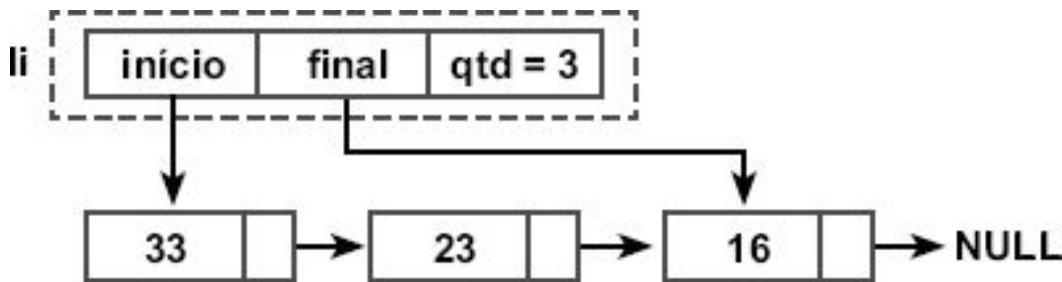
01 int insere_lista_final(Lista* li, struct aluno al){
02     if(li == NULL)
03         return 0;
04     Elemt *no;
05     no = (Elemt*) malloc(sizeof(Elemt));
06     if(no == NULL)
07         return 0;
08     no->dados = al;
09     no->prox = NULL;
10    if(li->inicio == NULL)//lista vazia: insere inicio
11        li->inicio = no;
12    else
13        li->final->prox = no;
14
15    li->final = no;
16    li->tamanho++;
17    return 1;
18 }
```



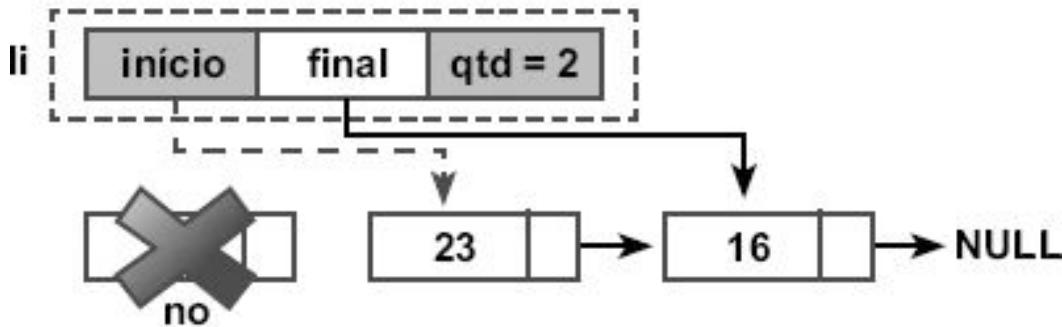
Lista com Nó Descritor: remoção



Lista com Nó Descritor: remoção início



↓
no = li->inicio;
li->inicio = no->prox;
free(no);
li->qtd--;



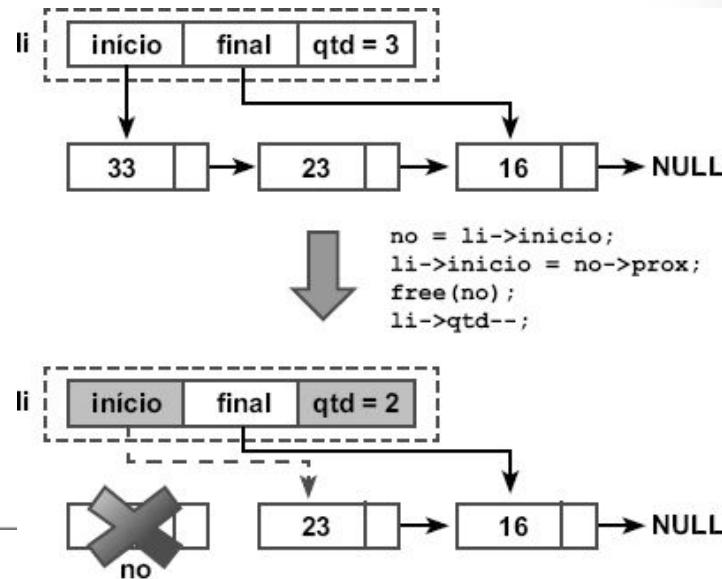
Lista com Nó Descritor: remoção início

Removendo um elemento do início da lista

```

01 int remove_lista_inicio(Lista* li) {
02     if(li == NULL)
03         return 0;
04     if(li->inicio == NULL)//lista vazia
05         return 0;
06     Elem *no = li->inicio;
07     li->inicio = no->prox;
08     free(no);
09     if(li->inicio == NULL)
10         li->final = NULL;
11     li->tamanho--;
12     return 1;
13 }
14

```





Listas Circulares

Lista Circular

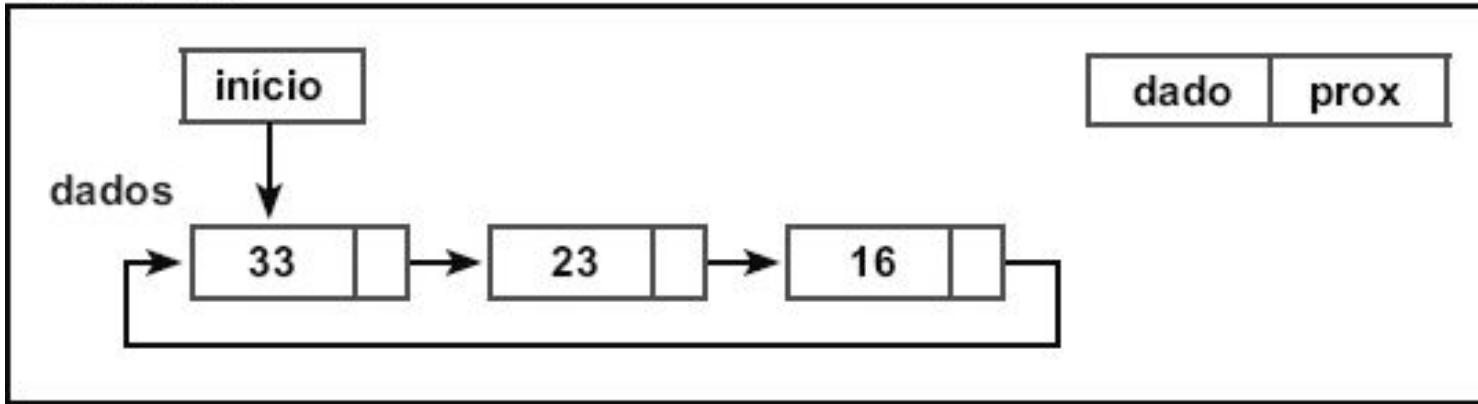
- Na lista dinâmica encadeada, após o último elemento não existe nenhum novo elemento alocado.
 - O último elemento da lista aponta para NULL.
- Na lista dinâmica encadeada circular, o último elemento antecede o primeiro elemento da lista.
 - A lista não tem fim.
 - Nunca chegaremos a uma posição final da lista.
 - Depois do último elemento volta-se para o primeiro, como em um círculo.

Listas circulares

- Tudo é um ciclo.
 - Tudo é um ciclo?
-
- Encerrando um ciclo (LinkedIn, 2024).
 - Ciclo encerra?

Lista Circular: estrutura

Lista *li



Lista Circular: estrutura

Arquivo ListaDinEncadCirc.h

```
01 struct aluno{  
02     int matricula;  
03     char nome[30];  
04     float n1,n2,n3;  
05 };  
06 typedef struct elemento* Lista;  
07  
08 Lista* cria_lista();  
09 void libera_lista(Lista* li);  
10 int busca_lista_pos(Lista* li, int pos, struct aluno *al);  
11 int busca_lista_mat(Lista* li, int mat, struct aluno *al);  
12 int insere_lista_final(Lista* li, struct aluno al);  
13 int insere_lista_inicio(Lista* li, struct aluno al);  
14 int insere_lista_ordenada(Lista* li, struct aluno al);  
15 int remove_lista(Lista* li, int mat);  
16 int remove_lista_inicio(Lista* li);  
17 int remove_lista_final(Lista* li);  
18 int tamanho_lista(Lista* li);  
19 int lista_vazia(Lista* li);  
20 int lista_cheia(Lista* li);
```

Lista Circular: estrutura

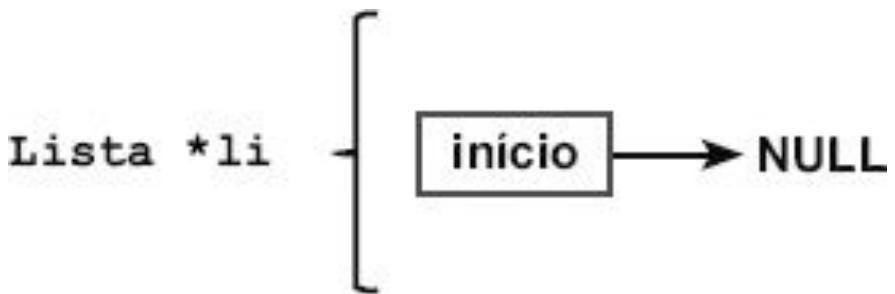
Arquivo ListaDinEncadCirc.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include "ListaDinEncadCirc.h" //inclui os Protótipos
04 //Definição do tipo lista
05 struct elemento{
06     struct aluno dados;
07     struct elemento *prox;
08 };
09 typedef struct elemento Elem;
```

Lista Circular: criação

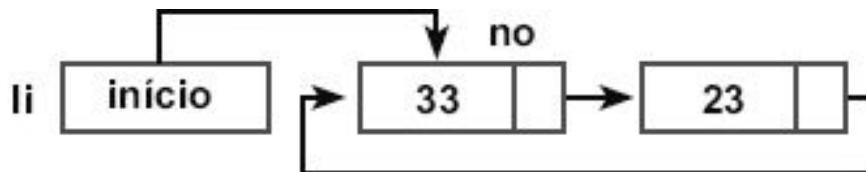
Criando uma lista

```
01     Lista* cria_lista(){
02         Lista* li = (Lista*) malloc(sizeof(Lista));
03         if(li != NULL)
04             *li = NULL;
05         return li;
06     }
```



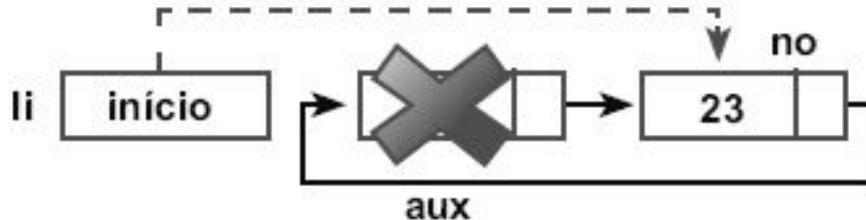
Lista Circular: destruição

Lista inicial



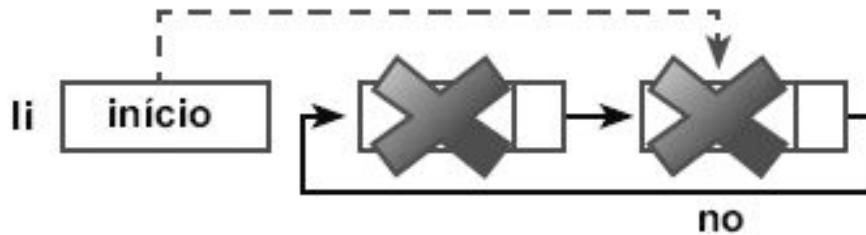
Passo 1:

```
aux = no;  
no = no->prox;  
free(aux);
```



Fim:

```
no->prox == *li;  
free(no);
```



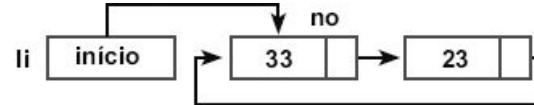
Lista Circular

Destruindo uma lista

```

01 void libera_lista(Lista* li){
02     if(li != NULL && (*li) != NULL) {
03         Elem *aux, *no = *li;
04         while((*li) != no->prox) {
05             aux = no;
06             no = no->prox;
07             free(aux);
08         }
09         free(no);
10         free(li);
11     }
12 }
```

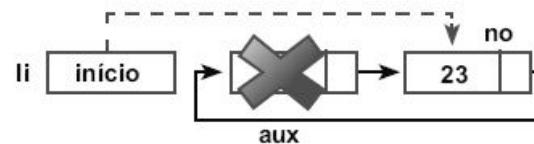
Lista inicial



Passo 1:

```

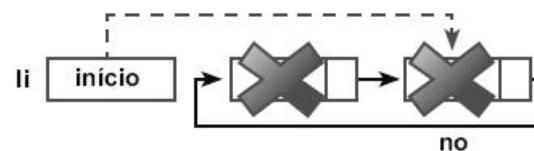
aux = no;
no = no->prox;
free(aux);
```



Fim:

```

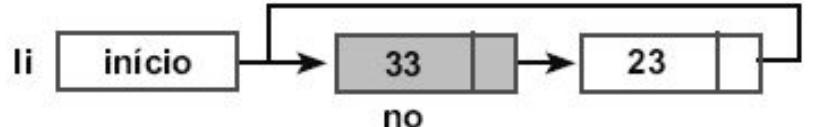
no->prox == *li;
free(no);
```



Lista Circular: tamanho

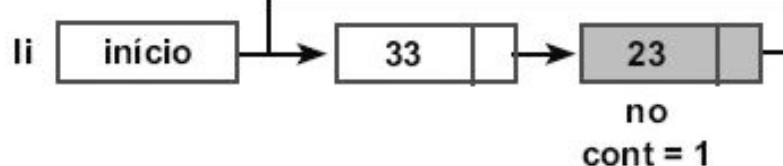
Lista inicial:

```
cont = 0;  
no = *li;
```



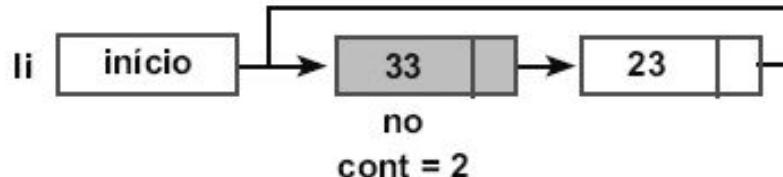
Passo 1:

```
cont++;  
no = no->prox;
```



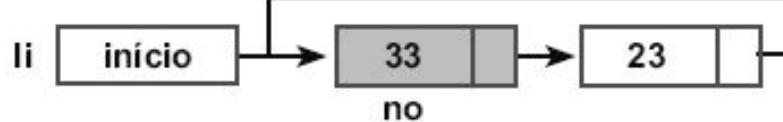
Passo 2:

```
cont++;  
no = no->prox;
```



Fim:

```
no == *li
```



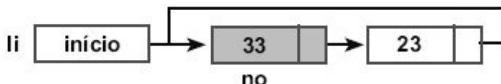
Lista Circular: tamanho

Tamanho da lista

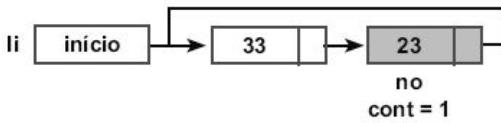
```

01 int tamanho_lista(Lista* li){
02     if(li == NULL || (*li) == NULL)
03         return 0;
04     int cont = 0;
05     Elemt* no = *li;
06     do{
07         cont++;
08         no = no->prox;
09     }while(no != (*li));
10     return cont;
11 }
```

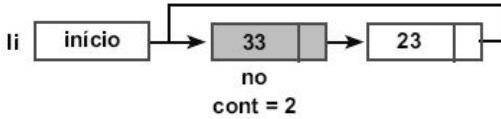
Lista inicial:
`cont = 0;`
`no = *li;`



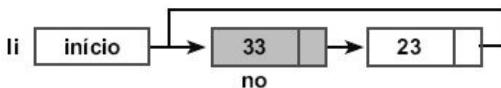
Passo 1:
`cont++;`
`no = no->prox;`



Passo 2:
`cont++;`
`no = no->prox;`



Fim:
`no == *li`



Lista Circular: lista cheia

Retornando se a lista está cheia

```
01 int lista_cheia(Lista* li) {  
02     return 0;  
03 }
```

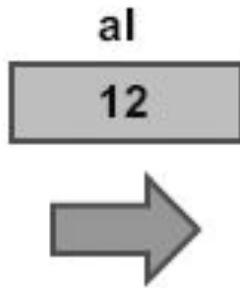
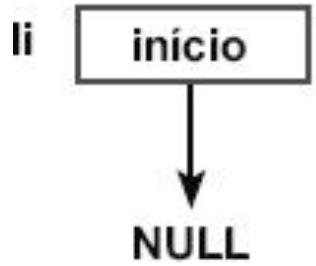


Lista Circular: lista vazia

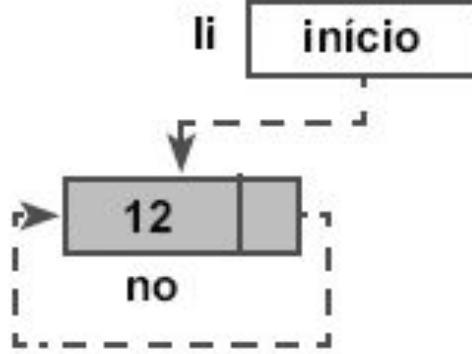
Retornando se a lista está vazia

```
01 int lista_vazia(Lista* li) {  
02     if(li == NULL)  
03         return 1;  
04     if(*li == NULL)  
05         return 1;  
06     return 0;  
07 }
```

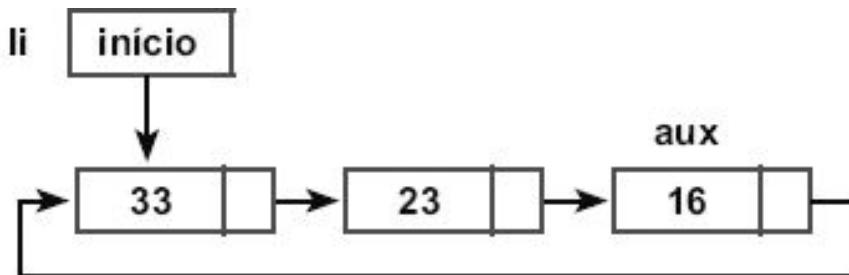
Lista Circular: inserção



```
no->dados = al;  
no->prox = no;  
*li = no;
```



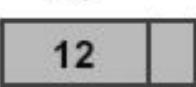
Lista Circular: inserção início



Busca último elemento “aux”:

```
aux = *li;  
while(aux->prox != (*li)){  
    aux = aux->prox;  
}
```

no



Insere depois de “aux”:

```
aux->prox = no;  
no->prox = *li;  
*li = no;
```

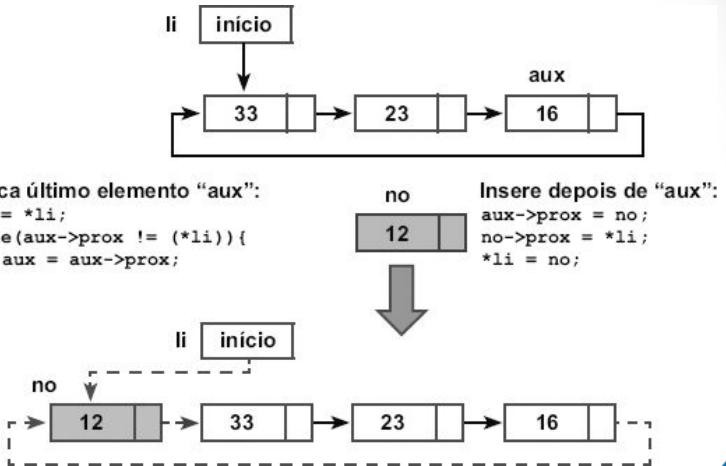


Lista Circular: inserção início

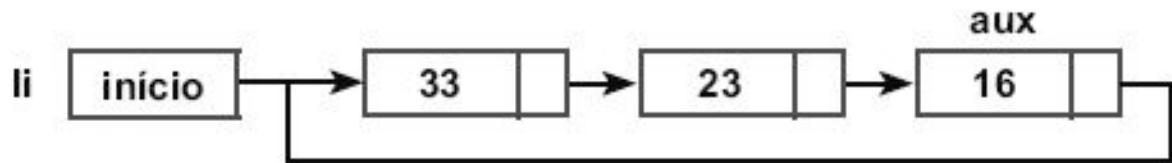
Inserindo um elemento no início da lista

```

01  int insere_lista_inicio(Lista* li, struct aluno al){
02      if(li == NULL)
03          return 0;
04      Elem *no = (ELEM*) malloc(sizeof(ELEM));
05      if(no == NULL)
06          return 0;
07      no->dados = al;
08      if((*li) == NULL){ //lista vazia: insere inicio
09          *li = no;
10          no->prox = no;
11      }else{
12          Elem *aux = *li;
13          while(aux->prox != (*li)){
14              aux = aux->prox;
15          }
16          aux->prox = no;
17          no->prox = *li;
18          *li = no;
19      }
20      return 1;
21  }
```

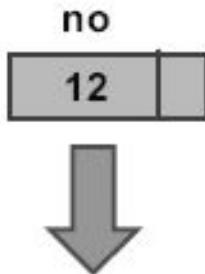


Lista Circular: inserção final



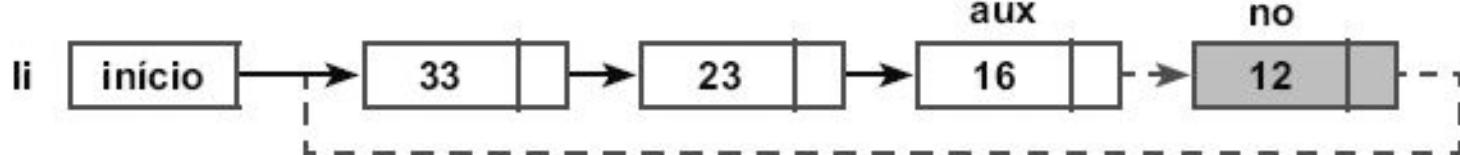
Busca último elemento “aux”:

```
aux = *li;  
while(aux->prox != (*li)){  
    aux = aux->prox;  
}  
//
```



Insere depois de “aux”:

```
aux->prox = no;  
no->prox = *li;
```



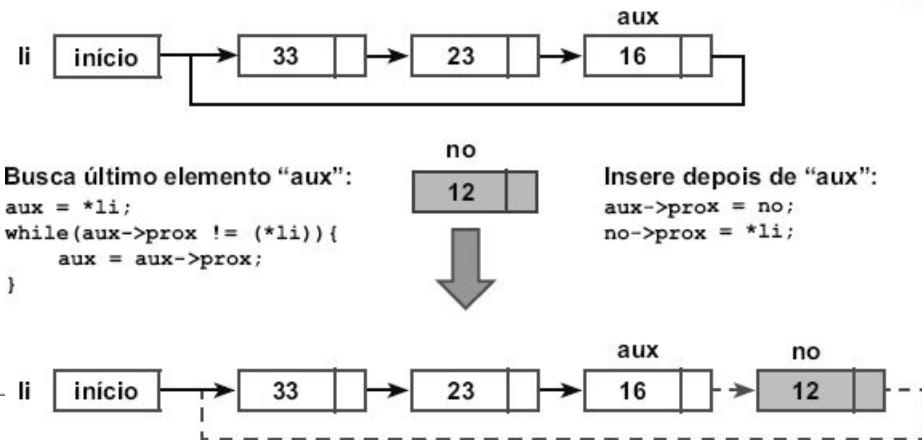
Lista Circular: inserção final

Inserindo um elemento no final da lista

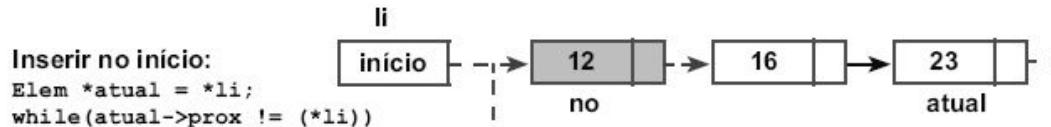
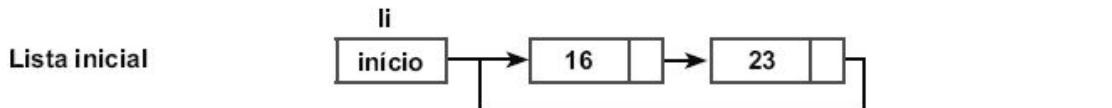
```

01  int insere_lista_final(Lista* li, struct aluno al){
02      if(li == NULL)
03          return 0;
04      Elem *no = (ELEM*) malloc(sizeof(ELEM));
05      if(no == NULL)
06          return 0;
07      no->dados = al;
08      if((*li) == NULL){//lista vazia: insere inicio
09          *li = no;
10          no->prox = no;
11      }else{
12          Elem *aux = *li;
13          while(aux->prox != (*li)){
14              aux = aux->prox;
15          }
16          aux->prox = no;
17          no->prox = *li;
18      }
19      return 1;
20  }

```



Lista Circular: inserção meio



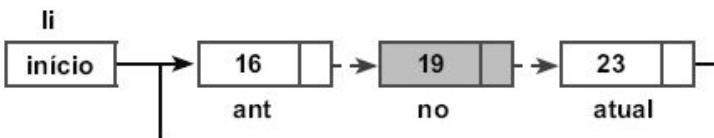
Inserir no meio ou no final

Busca onde inserir:

```
ant = *li;
atual = (*li)->prox;
while(atual != (*li) && atual->dados.matricula < al.matricula){
    ant = atual;
    atual = atual->prox;
}
```

Inserir depois de "ant":

```
ant->prox = no;
no->prox = atual;
```



Lista Circular: inserção meio

Inserindo um elemento de forma ordenada na lista

```

01  int insere_lista_ordenada(Lista* li, struct aluno al){
02      if(li == NULL)
03          return 0;
04      Elemt *no = (Elemt*) malloc(sizeof(Elemt));
05      if(no == NULL)
06          return 0;
07      no->dados = al;
08      if((*li) == NULL){//insere inicio
09          *li = no;
10          no->prox = no;
11          return 1;
12      }

```

Inserir no meio ou no final

Busca onde inserir:

```

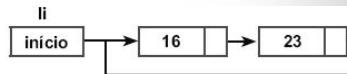
ant = *li;
atual = (*li)->prox;
while(atual != (*li) && atual->dados.matricula < al.matricula){
    ant = atual;
    atual = atual->prox;
}

```

Inserir depois de "ant":



Lista inicial



Inserir no inicio:

```

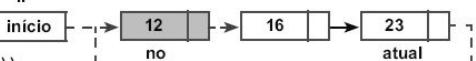
Elem *atual = *li;
while(atual->prox != (*li))
    atual = atual->prox;
no->prox = *li;
atual->prox = no;
*li = no;

```

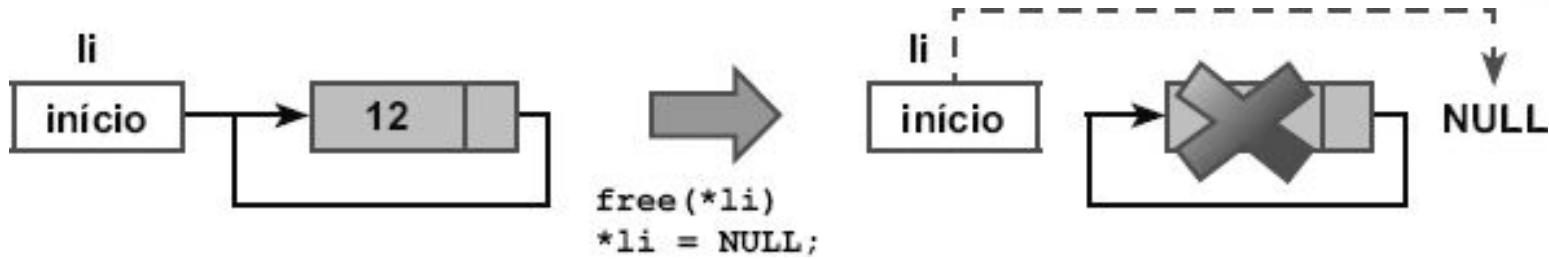
```

13     else{
14         if((*li)->dados.matricula > al.matricula){//início
15             Elemt *atual = *li;
16             while(atual->prox != (*li))//procura o último
17                 atual = atual->prox;
18             no->prox = *li;
19             atual->prox = no;
20             *li = no;
21             return 1;
22         }
23         Elemt *ant = *li, *atual = (*li)->prox;
24         while(atual != (*li) &&
25               atual->dados.matricula < al.matricula){
26             ant = atual;
27             atual = atual->prox;
28         }
29         ant->prox = no;
30         no->prox = atual;
31         return 1;
32     }

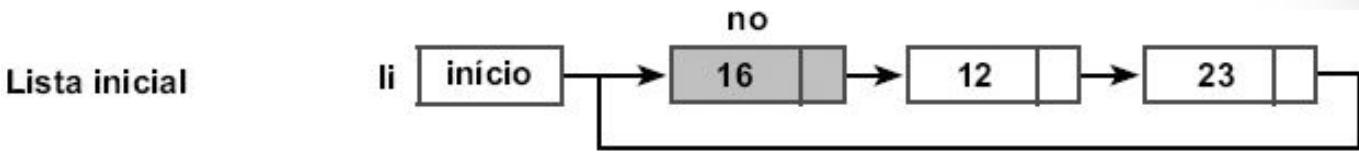
```



Lista Circular: remoção

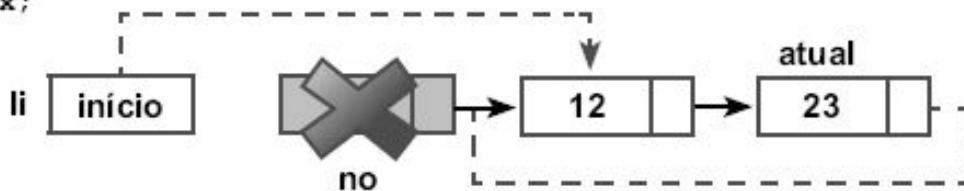


Lista Circular: remoção início



Busca último elemento: "atual"

```
atual = *li;
while(atual->prox != (*li))
    atual = atual->prox;
```

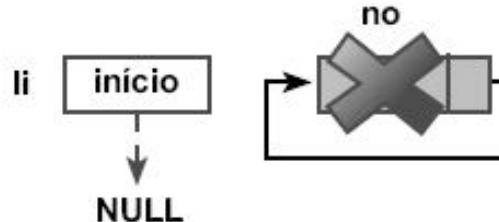


Remove o elemento:

```
no = *li;
atual->prox = no->prox;
*li = no->prox;
free(no);
```

**Se "no" é o único elemento
da lista, a lista fica vazia**

```
free(*li);
*li = NULL;
```



Lista Circular

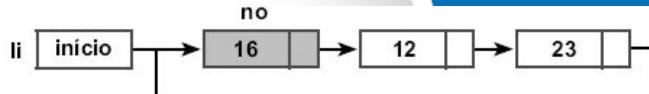
Removendo um elemento do início da lista

```

01  int remove_lista_inicio(Lista* li){
02      if(li == NULL)
03          return 0;
04      if((*li) == NULL)//lista vazia
05          return 0;
06
07      if((*li) == (*li)->prox){//lista fica vazia
08          free(*li);
09          *li = NULL;
10          return 1;
11      }
12      Elemt *atual = *li;
13      while(atual->prox != (*li))//procura o último
14          atual = atual->prox;
15
16      Elemt *no = *li;
17      atual->prox = no->prox;
18      *li = no->prox;
19      free(no);
20      return 1;
21  }

```

Lista inicial

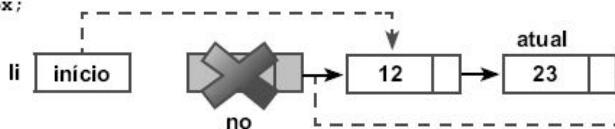


Busca último elemento: "atual"

```

atual = *li;
while(atual->prox != (*li))
    atual = atual->prox;

```



Remove o elemento:

```

no = *li;
atual->prox = no->prox;
*li = no->prox;
free(no);

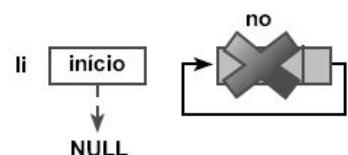
```

Se "no" é o único elemento
da lista, a lista fica vazia

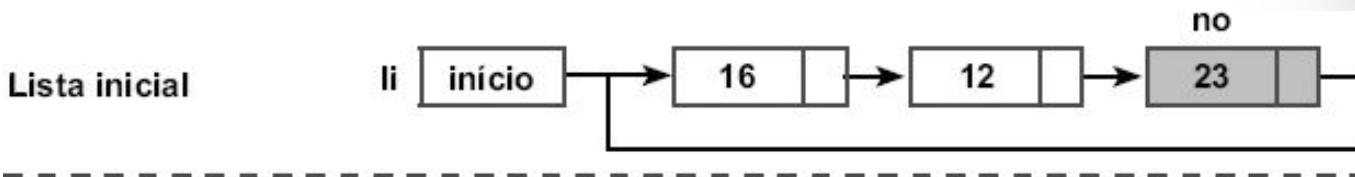
```

free(*li);
*li = NULL;

```



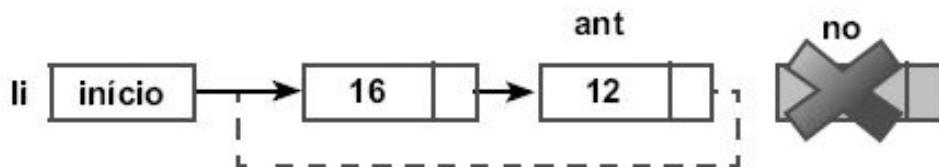
Lista Circular: remoção final



Busca último elemento: "no"

```

no = *li;
while(no->prox != (*li)){
    ant = no;
    no = no->prox;
}
  
```



Remove o elemento:

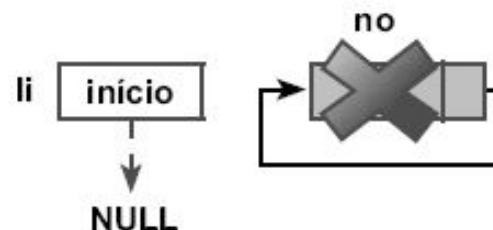
```

ant->prox = no->prox;
free(no);
  
```

Se "no" é o único elemento da lista, a lista fica vazia:

```

free(*li);
*li = NULL;
  
```



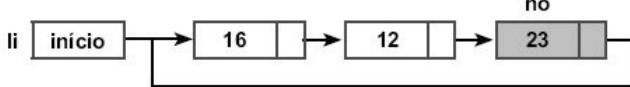
Lista Circular: remoção final

Removendo um elemento do final da lista

```

01 int remove_lista_final(Lista* li){
02     if(li == NULL)
03         return 0;
04     if((*li) == NULL)//lista vazia
05         return 0;
06
07     if((*li) == (*li)->prox){//lista fica vazia
08         free(*li);
09         *li = NULL;
10         return 1;
11     }
12     Elemt *ant, *no = *li;
13     while(no->prox != (*li)){//procura o último
14         ant = no;
15         no = no->prox;
16     }
17     ant->prox = no->prox;
18     free(no);
19     return 1;
20 }
```

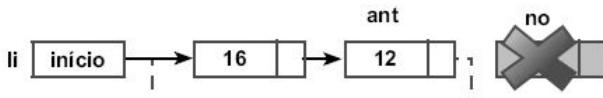
Lista inicial



Busca último elemento: "no"

```

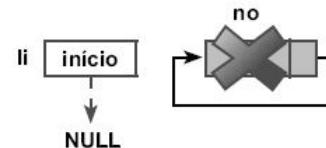
no = *li;
while(no->prox != (*li)){
    ant = no;
    no = no->prox;
}
Remove o elemento:
ant->prox = no->prox;
free(no);
```



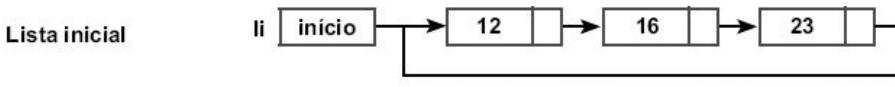
Se "no" é o único elemento
da lista, a lista fica vazia:

```

free(*li);
*li = NULL;
```



Lista Circular: remoção meio



Remover do início:

```

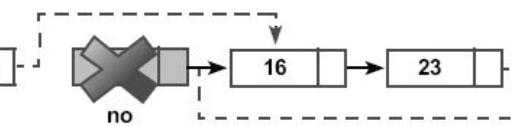
if(no == no->prox){//lista fica vazia
    free(no);
    *li = NULL;
    return 1;
} else{
    Elemt *ult = *li;
    while(ult->prox != (*li))
        ult = ult->prox;
    ult->prox = (*li)->prox;
    *li = (*li)->prox;
    free(no);
    return 1;
}
  
```

Remover do meio ou do final

Buscar onde remover:

```

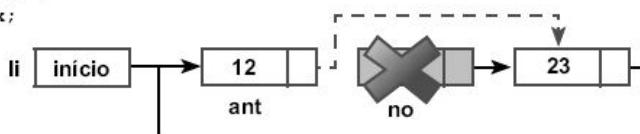
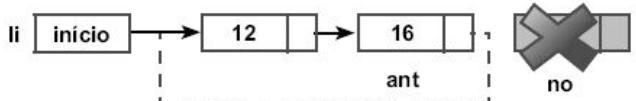
ant = no;
no = no->prox;
while(no != (*li) && no->dados.matricula != mat){
    ant = no;
    no = no->prox;
}
  
```



Remover depois de "ant":

```

ant->prox = no->prox;
free(no);
  
```



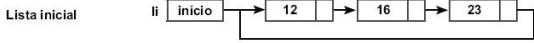
Lista Circular: remoção meio

Removendo um elemento específico da lista

```

01 int remove_lista(Lista* li, int mat){
02     if(li == NULL)
03         return 0;
04     if((*li) == NULL)//lista vazia
05         return 0;
06     Elem *no = *li;
07     if(no->dados.matricula == mat){//remover do inicio
08         if(no == no->prox){//lista fica vazia
09             free(no);
10             *li = NULL;
11             return 1;
12         }else{
13             Elem *ult = *li;
14             while(ult->prox != (*li))//procura o ultimo
15                 ult = ult->prox;
16                 ult->prox = (*li)->prox;
17                 *li = (*li)->prox;
18                 free(no);
19                 return 1;
20         }
21     }

```



Remover do inicio:

```

if(no == no->prox){//lista fica vazia
    free(no);
    *li = NULL;
    return 1;
}else{
    Elem *ult = *li;
    while(ult->prox != (*li))
        ult = ult->prox;
    ult->prox = (*li)->prox;
    *li = (*li)->prox;
    free(no);
    return 1;
}

```

Remover do meio ou do final

```

Buscar onde remover:
ant = no;
no = no->prox;
while(no != (*li) && no->dados.matricula != mat){
    ant = no;
    no = no->prox;
}

```

Remover depois de "ant":

```

ant->prox = no->prox;
free(no);

```

```

22     Elem *ant = no;
23     no = no->prox;
24     while(no != (*li) && no->dados.matricula != mat){
25         ant = no;
26         no = no->prox;
27     }
28     if(no == *li)//não encontrado
29         return 0;
30
31     ant->prox = no->prox;
32     free(no);
33     return 1;
34 }

```

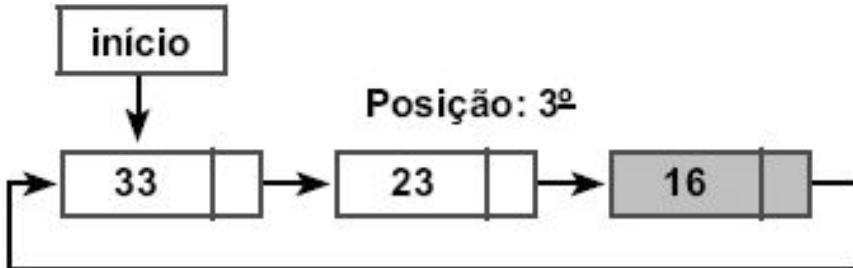
Lista Circular: busca posição

Busca pela posição do elemento:

```
no = *li;
int i = 1;
while(no->prox != (*li) && i < pos){
    no = no->prox;
    i++;
}
```

Verifica se a posição foi encontrada e a retorna:

```
if(i != pos) return 0;
else{
    *al = no->dados;
    return 1;
}
```



Lista Circular: busca posição

Busca um elemento por posição

```
01 int busca_lista_pos(Lista* li, int pos, struct aluno *al){  
02     if(li == NULL || (*li) == NULL || pos <= 0)  
03         return 0;  
04     Elemt *no = *li;  
05     int i = 1;  
06     while(no->prox != (*li) && i < pos){  
07         no = no->prox;  
08         i++;  
09     }  
10     if(i != pos)  
11         return 0;  
12     else{  
13         *al = no->dados;  
14         return 1;  
15     }  
16 }
```

Busca pela posição do elemento:

```
no = *li;  
int i = 1;  
while(no->prox != (*li) && i < pos){  
    no = no->prox;  
    i++;  
}
```

Verifica se a posição foi encontrada e a retorna:

```
if(i != pos) return 0;  
else{  
    *al = no->dados;  
    return 1;  
}
```



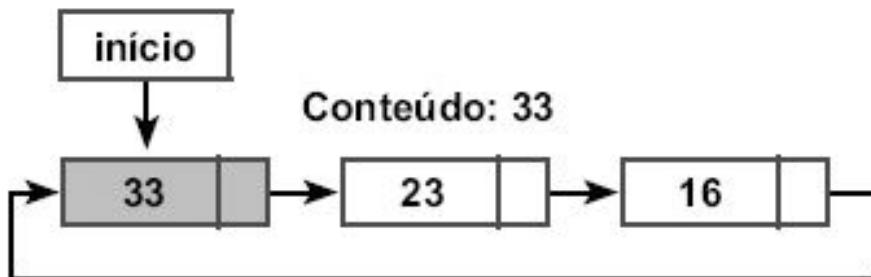
Lista Circular: busca elemento

Busca pelo conteúdo do elemento:

```
no = *li;
while(no->prox != (*li) && no->dados.matricula != mat)
    no = no->prox;
```

Verifica se o elemento foi
encontrado e o retorna:

```
if(no->dados.matricula != mat)
    return 0;
else{
    *al = no->dados;
    return 1;
}
```



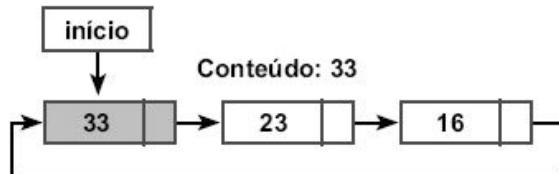
Lista Circular: busca elemento

Busca um elemento por conteúdo

```
01  int busca_lista_mat(Lista* li, int mat, struct aluno *al){  
02      if(li == NULL || (*li) == NULL)  
03          return 0;  
04      Elemt *no = *li;  
05      while(no->prox != (*li) && no->dados.matricula != mat)  
06          no = no->prox;  
07      if(no->dados.matricula != mat)  
08          return 0;  
09      else{  
10          *al = no->dados;  
11          return 1;  
12      }  
13  }
```

Busca pelo conteúdo do elemento:
no = *li;
while(no->prox != (*li) && no->dados.matricula != mat)
 no = no->prox;

Verifica se o elemento foi
encontrado e o retorna:
if(no->dados.matricula != mat)
 return 0;
else{
 *al = no->dados;
 return 1;
}





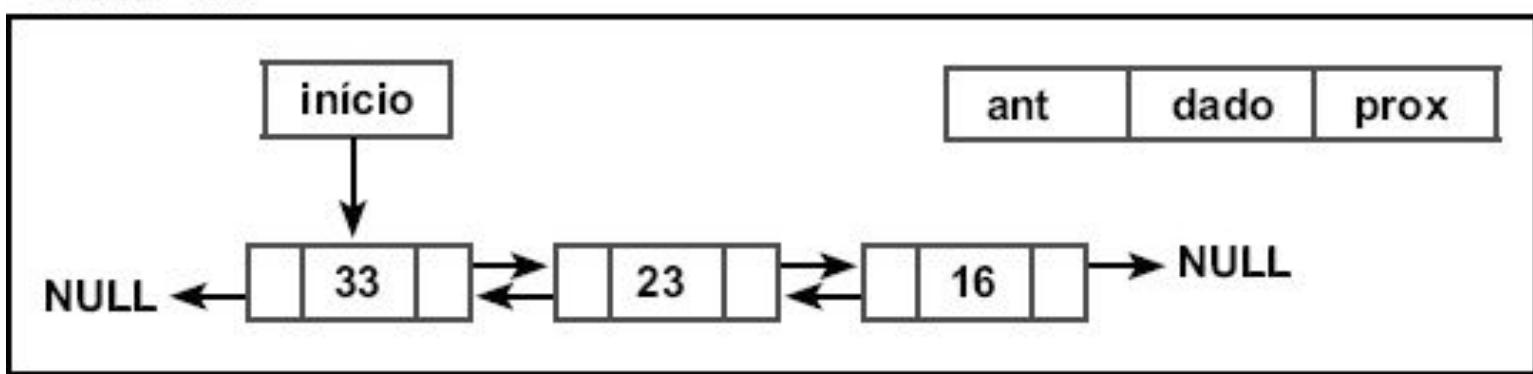
Listas Duplamente Encadeadas

Lista Duplamente Encadeada

- Diferente da lista dinâmica encadeada, esse tipo de lista não possui dois, mas três campos de informação dentro de cada elemento: os campos **dado**, **prox** e **ant**.
- É a existência dos campos **prox** e **ant** que garante que a lista é duplamente encadeada.
- Após o último elemento não existe nenhum novo elemento.
 - O último elemento da lista aponta o campo **prox** para NULL.
 - O primeiro elemento da lista aponta o campo **ant** para NULL.

Lista Duplamente Encadeada: estrutura

Lista *li



Lista Duplamente Encadeada: estrutura

Arquivo ListaDinEncadDupla.h

```
01 struct aluno{  
02     int matricula;  
03     char nome[30];  
04     float n1,n2,n3;  
05 };  
06 typedef struct elemento* Lista;  
07  
08 Lista* cria_lista();  
09 void libera_lista(Lista* li);  
10 int busca_lista_pos(Lista* li, int pos, struct aluno *al);  
11 int busca_lista_mat(Lista* li, int mat, struct aluno *al);  
12 int insere_lista_final(Lista* li, struct aluno al);  
13 int insere_lista_inicio(Lista* li, struct aluno al);  
14 int insere_lista_ordenada(Lista* li, struct aluno al);  
15 int remove_lista(Lista* li, int mat);  
16 int remove_lista_inicio(Lista* li);  
17 int remove_lista_final(Lista* li);  
18 int tamanho_lista(Lista* li);  
19 int lista_vazia(Lista* li);  
20 int lista_cheia(Lista* li);
```

Lista Duplamente Encadeada: estrutura

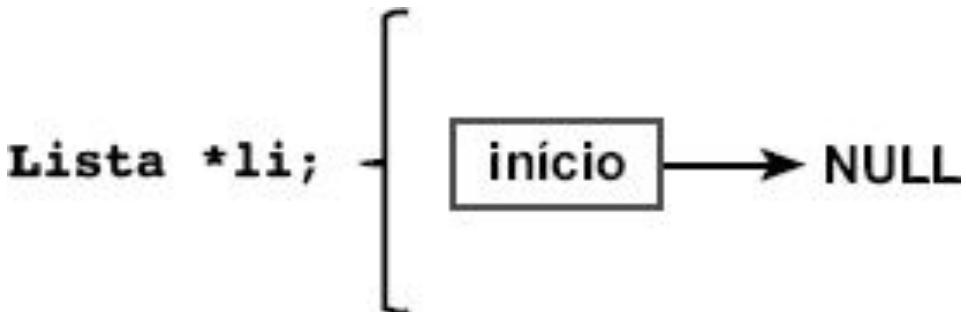
Arquivo ListaDinEncadDupla.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include "ListaDinEncadDupla.h" //inclui os protótipos
04 //Definição do tipo lista
05 struct elemento{
06     struct elemento *ant;
07     struct aluno dados;
08     struct elemento *prox;
09 };
10 typedef struct elemento Elem;
```

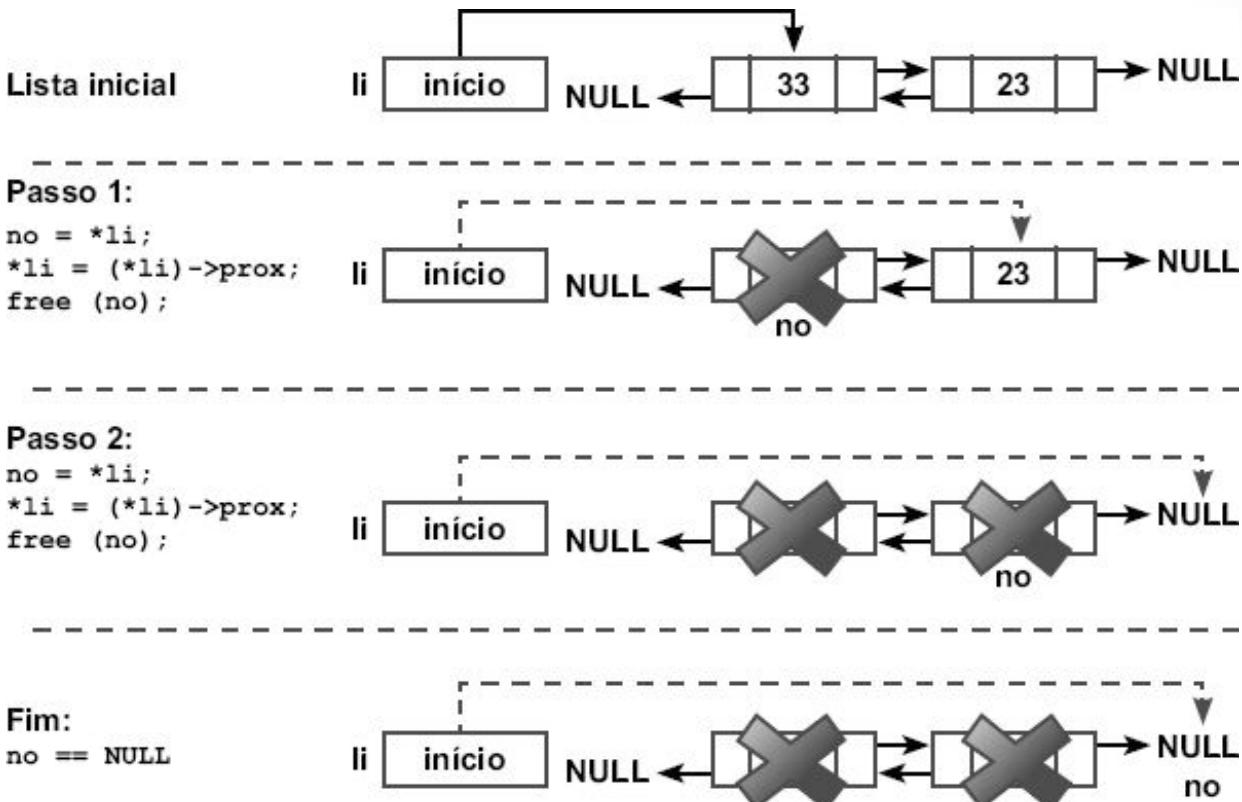
Lista Duplamente Encadeada: criação

Criando uma lista

```
01  Lista* cria_lista(){
02      Lista* li = (Lista*) malloc(sizeof(Lista));
03      if(li != NULL)
04          *li = NULL;
05      return li;
06  }
```



Lista Duplamente Encadeada: destruição

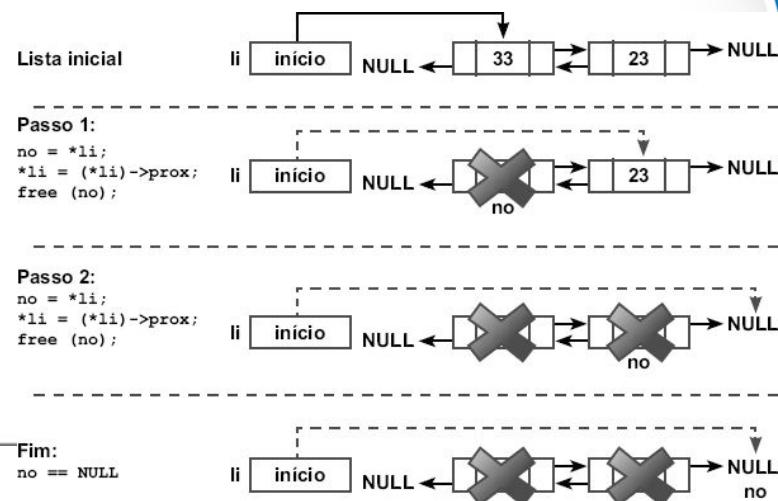


Lista Duplamente Encadeada: destruição

Destruidor uma lista

```

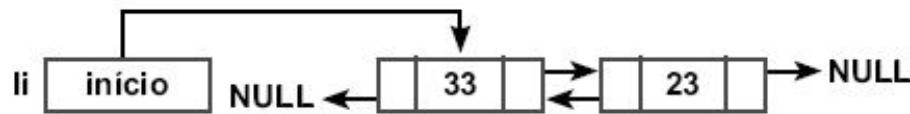
01 void libera_lista(Lista* li) {
02     if(li != NULL) {
03         ELEM* no;
04         while((*li) != NULL) {
05             no = *li;
06             *li = (*li)->prox;
07             free(no);
08         }
09         free(li);
10     }
11 }
```



Lista Duplamente Encadeada: tamanho

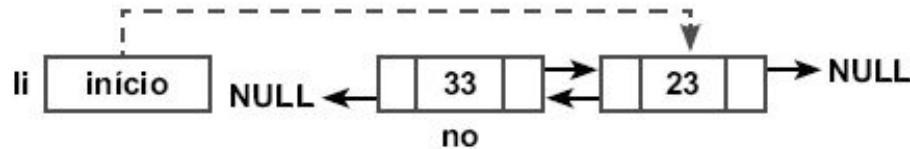
Lista inicial:

```
cont = 0;  
no = *li;
```



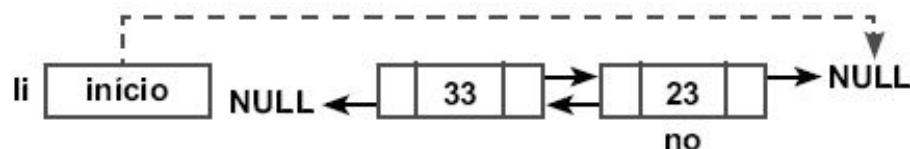
Passo 1:

```
cont++;  
no = no->prox;
```



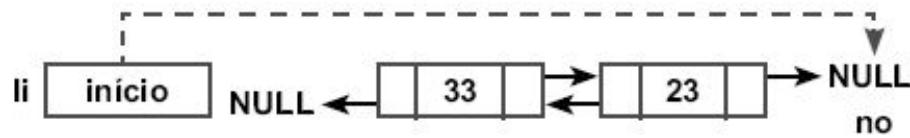
Passo 2:

```
cont++;  
no = no->prox;
```



Fim:

```
no == NULL
```



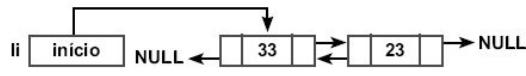
Lista Duplamente Encadeada: tamanho

Tamanho da lista

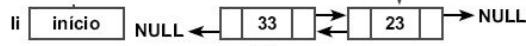
```

01 int tamanho_lista(Lista* li) {
02     if(li == NULL)
03         return 0;
04     int cont = 0;
05     Elem* no = *li;
06     while(no != NULL) {
07         cont++;
08         no = no->prox;
09     }
10     return cont;
11 }
```

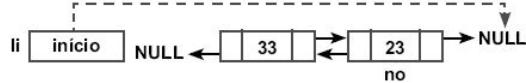
Lista inicial:
 cont = 0;
 no = *li;



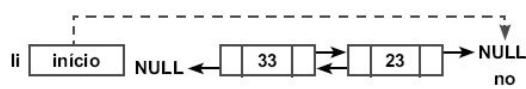
Passo 1:
 cont++;
 no = no->prox;



Passo 2:
 cont++;
 no = no->prox;



Fim:
 no == NULL



Lista Duplamente Encadeada: lista cheia

Retornando se a lista está cheia

```
01 int lista_cheia(Lista* li) {  
02     return 0;  
03 }
```

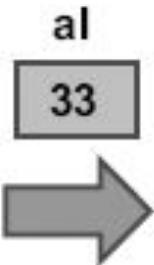
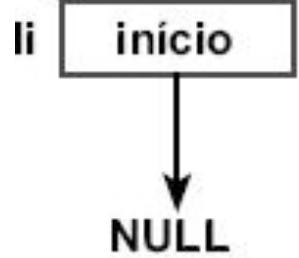


Lista Duplamente Encadeada: lista vazia

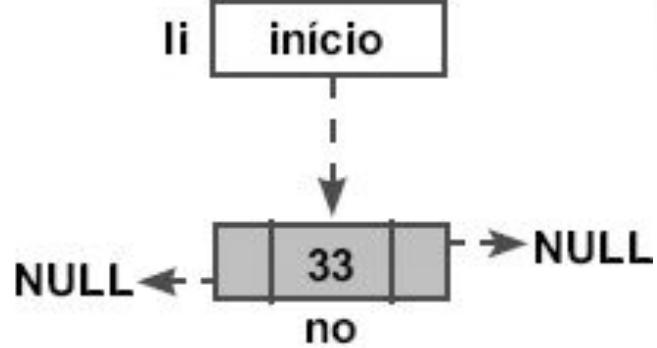
Retornando se a lista está vazia

```
01 int lista_vazia(Lista* li) {  
02     if(li == NULL)  
03         return 1;  
04     if(*li == NULL)  
05         return 1;  
06     return 0;  
07 }
```

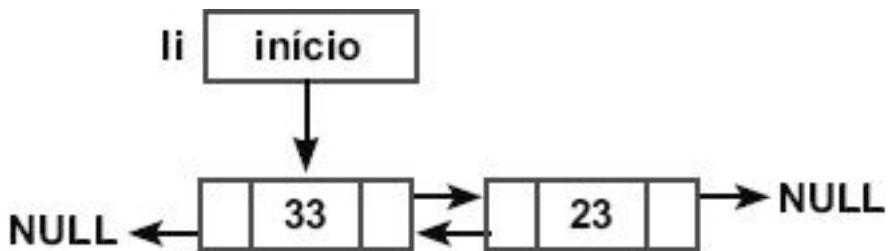
Lista Duplamente Encadeada: inserção



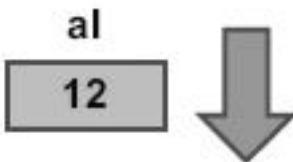
```
no->dados = al;  
no->prox = NULL;  
no->ant = NULL;  
*li = no;
```



Lista Duplamente Encadeada: inserção início

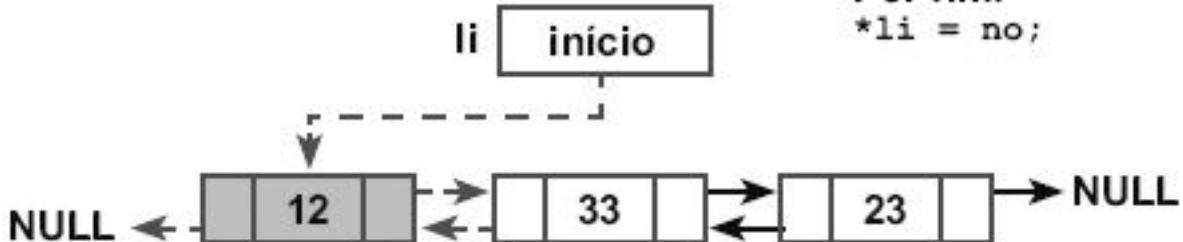


```
no->dados = al;  
no->prox = (*li);  
no->ant = NULL;
```



Lista não estava vazia:
`(*li)->ant = no;`

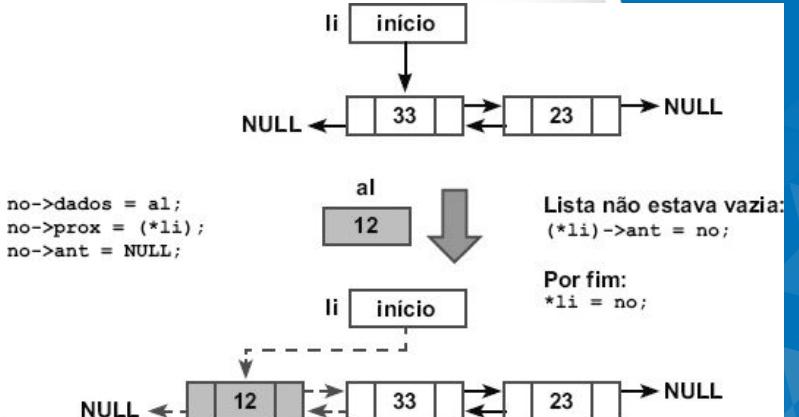
Por fim:
`*li = no;`



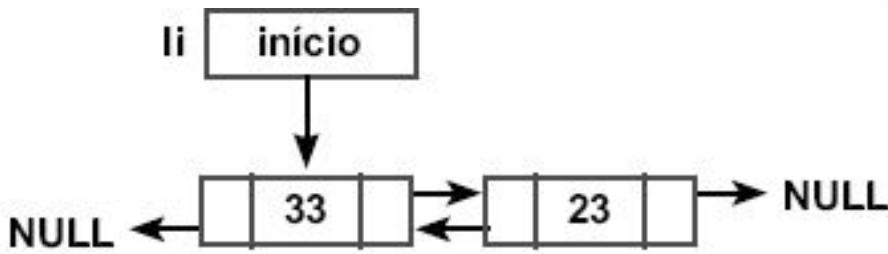
Lista Duplamente Encadeada: inserção início

Inserindo um elemento no início da lista

```
01 int insere_lista_inicio(Lista* li, struct aluno al) {
02     if(li == NULL)
03         return 0;
04     ELEM* no;
05     no = (ELEM*) malloc(sizeof(ELEM));
06     if(no == NULL)
07         return 0;
08     no->dados = al;
09     no->prox = (*li);
10    no->ant = NULL;
11    //lista não vazia: apontar para o anterior!
12    if(*li != NULL)
13        (*li)->ant = no;
14    *li = no;
15    return 1;
16 }
```

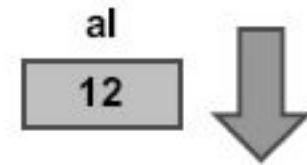


Lista Duplamente Encadeada: inserção final



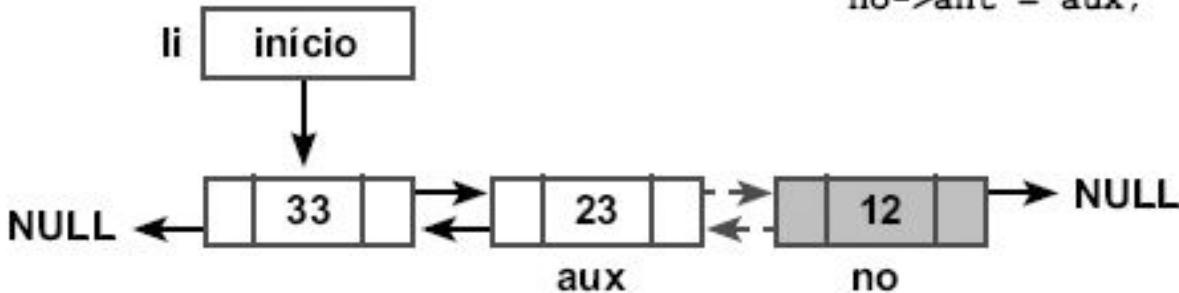
Busca onde inserir:

```
aux = *li;
while(aux->prox != NULL) {
    aux = aux->prox;
}
```



Insere depois de “aux”:

```
no->dados = al;
no->prox = NULL;
aux->prox = no;
no->ant = aux;
```

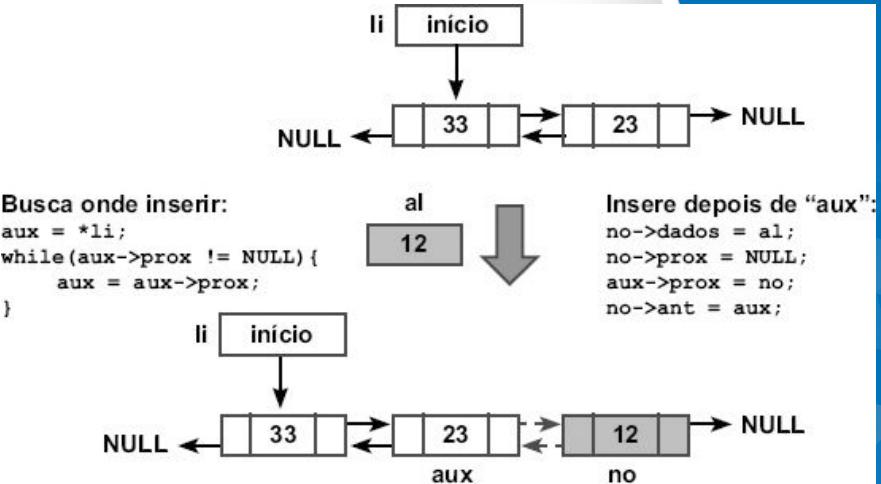


Lista Duplamente Encadeada: inserção final

Inserindo um elemento no final da lista

```

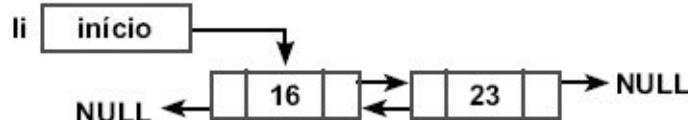
01  int insere_lista_final(Lista* li, struct aluno al){
02      if(li == NULL)
03          return 0;
04      ELEM *no;
05      no = (ELEM*) malloc(sizeof(ELEM));
06      if(no == NULL)
07          return 0;
08      no->dados = al;
09      no->prox = NULL;
10     if((*li) == NULL){ //lista vazia: insere inicio
11         no->ant = NULL;
12         *li = no;
13     }else{
14         ELEM *aux = *li;
15         while(aux->prox != NULL){
16             aux = aux->prox;
17         }
18         aux->prox = no;
19         no->ant = aux;
20     }
21     return 1;
22 }
```



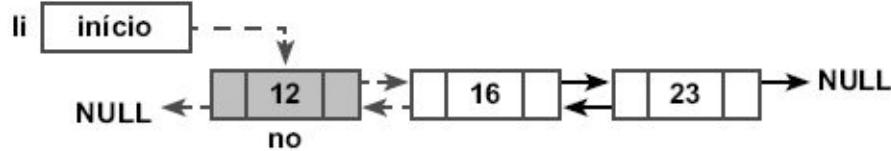
Lista Duplamente Encadeada: inserção meio

Lista inicial
Busca onde inserir:

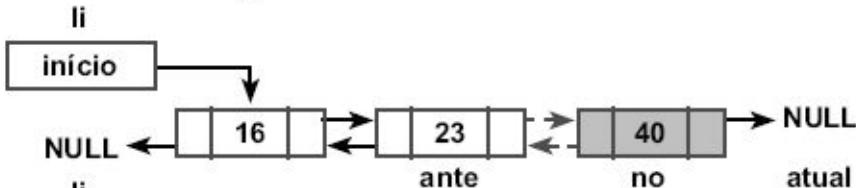
```
atual = *li;
while(atual!=NULL && atual->dados.matricula < al.matricula) {
    ante = atual;
    atual = atual->prox;
}
```



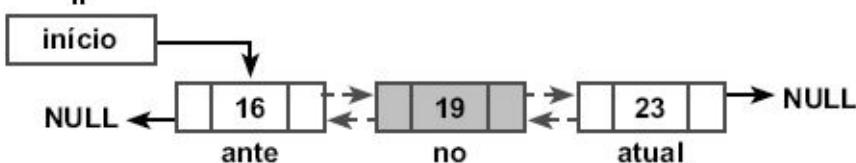
Inserir no início:
`no->ant = NULL;
(*li)->ant = no;
no->prox = (*li);
*li = no;`



Inserir depois de "ante":
`no->prox=ante->prox;
no->ant = ante;
ante->prox = no;`



Não é final da lista:
`atual->ant = no;`



Lista Duplamente Encadeada: inserção meio

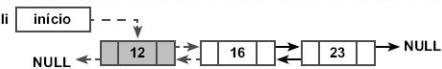
Inserindo um elemento de forma ordenada na lista

```

01 int insere_lista_ordenada(Lista* li, struct aluno al) {
02     if(li == NULL)
03         return 0;
04     Elemt *no;
05     no = (Elemt*) malloc(sizeof(Elemt));
06     if(no == NULL)
07         return 0;
08     no->dados = al;
09     if((*li) == NULL){//lista vazia: insere inicio
10         no->prox = NULL;
11         no->ant = NULL;
12         *li = no;
13         return 1;
14     }

```

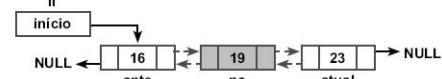
Inserir no inicio:
 no->ant = NULL;
 (*li)->ant = no;
 no->prox = (*li);
 *li = no;



Inserir depois de "ante":
 no->prox=ante->prox;
 no->ant = ante;
 ante->prox = no;



Não é final da lista:
 atual->ant = no;



```

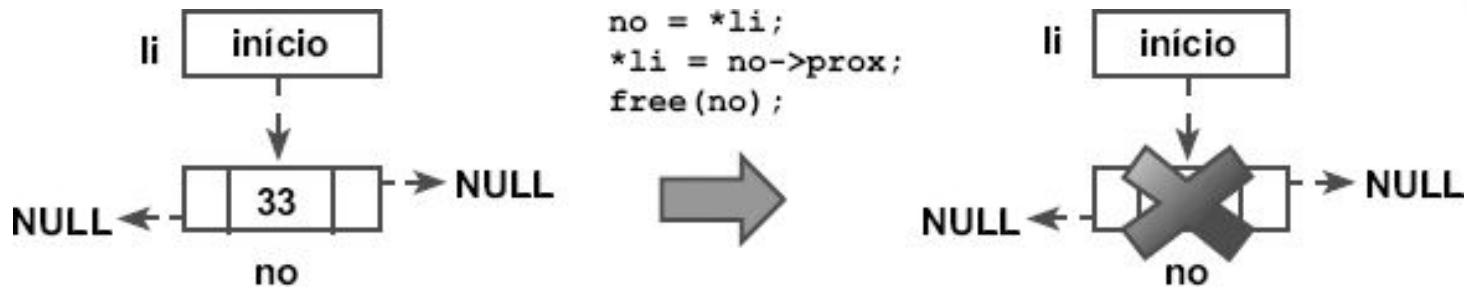
15     else{
16         Elemt *ante, *atual = *li;
17         while(atual != NULL &&
18               atual->dados.matricula < al.matricula){
19             ante = atual;
20             atual = atual->prox;
21         }
22         if(atual == *li){//insere inicio
23             no->ant = NULL;
24             (*li)->ant = no;
25             no->prox = (*li);
26             *li = no;
27         }else{
28             no->prox = ante->prox;
29             no->ant = ante;
30             ante->prox = no;
31             if(atual != NULL)
32                 atual->ant = no;
33         }
34     }
35 }

```

Lista inicial
 Busca onde inserir:
 atual = *li;
 while(atual!=NULL && atual->dados.matricula < al.matricula){
 ante = atual;
 atual = atual->prox;
 }

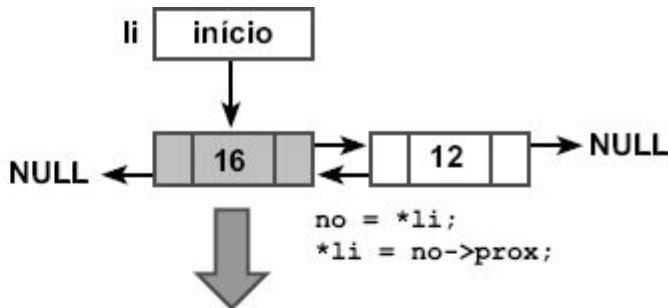


Lista Duplamente Encadeada: remoção



Lista Duplamente Encadeada: remoção início

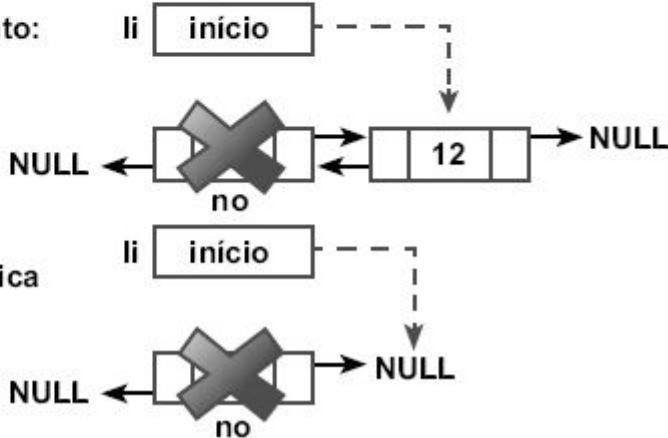
Lista inicial



Se a lista possui mais de um elemento:
`no->prox->ant = NULL;`

Por fim:
`free(no);`

Se “no” é o único elemento, a lista fica vazia.



Lista Duplamente Encadeada: remoção início

Removendo um elemento do início da lista

```

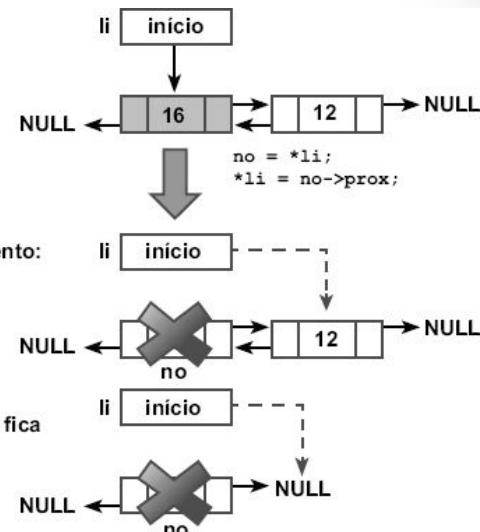
01  int remove_lista_inicio(Lista* li){
02      if(li == NULL)
03          return 0;
04      if((*li) == NULL)//lista vazia
05          return 0;
06
07      Elem *no = *li;
08      *li = no->prox;
09      if(no->prox != NULL)
10          no->prox->ant = NULL;
11
12      free(no);
13      return 1;
14  }
  
```

Lista inicial

Se a lista possui mais de um elemento:
no->prox->ant = NULL;

Por fim:
free(no);

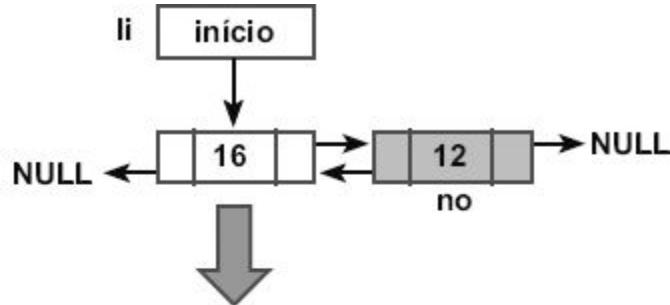
Se "no" é o único elemento, a lista fica vazia.



Lista Duplamente Encadeada: remoção final

Procura último elemento da lista:

```
no = *li;  
while(no->prox != NULL)  
    no = no->prox;
```



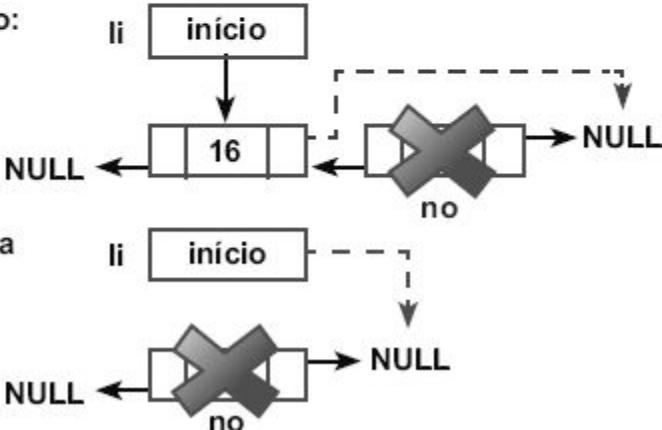
Se a lista possui mais de um elemento:

```
no->ant->prox = NULL;
```

Por fim:

```
free(no);
```

Se "no" é o único elemento, a lista fica vazia.



Lista Duplamente Encadeada: remoção final

Removendo um elemento do final da lista

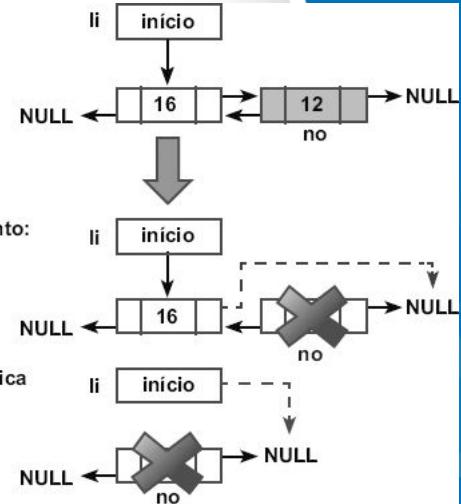
```

01  int remove_lista_final(Lista* li){
02      if(li == NULL)
03          return 0;
04      if((*li) == NULL)//lista vazia
05          return 0;
06
07      ELEM *no = *li;
08      while(no->prox != NULL)
09          no = no->prox;
10
11     if(no->ant == NULL)//remover o primeiro e único
12         *li = no->prox;
13     else
14         no->ant->prox = NULL;
15
16     free(no);
17     return 1;
18 }
```

a último elemento da lista:
 li;
 no->prox != NULL)
 i = no->prox;

sta possui mais de um elemento:
 i->prox = NULL;
 n:
 o;

" é o único elemento, a lista fica



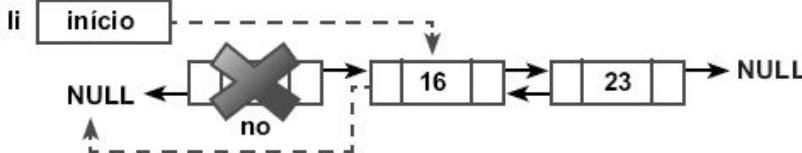
Lista Duplamente Encadeada: remoção meio

Lista inicial
Busca qual remover:

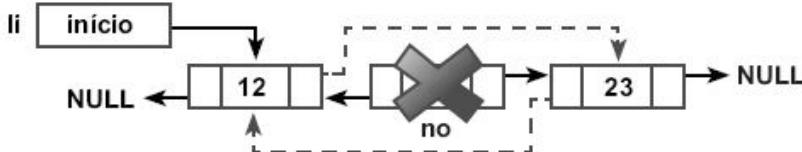
```
no = *li;
while(no != NULL && no->dados.matricula != mat) {
    no = no->prox;
}
```



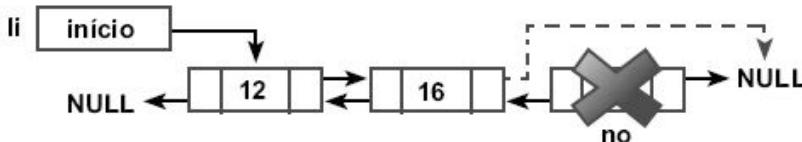
Remover do início:
`*li = no->prox;`



**Não está removendo
do final:**
`no->prox->ant = no->ant;`



**Remover do meio
ou do final:**
`no->ant->prox=no->prox;`
Por fim:
`free(no);`



Lista Duplamente Encadeada: remoção meio

Removendo um elemento específico da lista

```

01 int remove_lista(Lista* li, int mat){
02     if(li == NULL)
03         return 0;
04     if((*li) == NULL)//lista vazia
05         return 0;
06     ELEM *no = *li;
07     while(no != NULL & no->dados.matricula != mat){
08         no = no->prox;
09     }
10     if(no == NULL)//não encontrado
11         return 0;
12
13     if(no->ant == NULL)//remover o primeiro
14         *li = no->prox;
15     else
16         no->ant->prox = no->prox;
17
18     if(no->prox != NULL)//não é o último
19         no->prox->ant = no->ant;
20
21     free(no);
22     return 1;
23 }
```

Lista inicial
Busca qual remover:

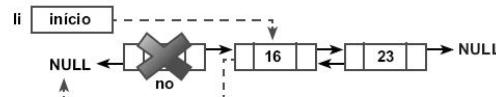
```

no = *li;
while(no != NULL && no->dados.matricula != mat){
    no = no->prox;
}

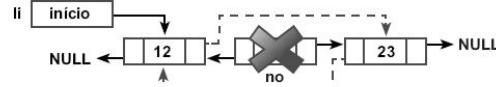
```



Remover do inicio:
*li = no->prox;



Não está removendo
do final:
no->prox->ant = no->ant;



Remover do meio
ou do final:
no->ant->prox=no->prox;
Por fim:
free(no);



Lista Duplamente Encadeada: busca posição

Busca pela posição do elemento:

```
no = *li;
int i = 1;
while(no != NULL && i < pos) {
    no = no->prox;
    i++;
}
```



Verifica se o elemento foi
encontrado e o retorna:

```
if(no == NULL) return 0;
else{
    *al = no->dados;
    return 1;
}
```

Lista Duplamente Encadeada: busca posição

Busca um elemento por posição

```

01 int busca_lista_pos(Lista* li, int pos, struct aluno *al) {
02     if(li == NULL || pos <= 0)
03         return 0;
04     Elemt *no = *li;
05     int i = 1;
06     while(no != NULL && i < pos){
07         no = no->prox;
08         i++;
09     }
10     if(no == NULL)
11         return 0;
12     else{
13         *al = no->dados;
14         return 1;
15     }
16 }
```

Busca pela posição do elemento:
`no = *li;
int i = 1;
while(no != NULL && i < pos){
 no = no->prox;
 i++;
}`

Verifica se o elemento foi encontrado e o retorna:
`if(no == NULL) return 0;
else{
 *al = no->dados;
 return 1;
}`



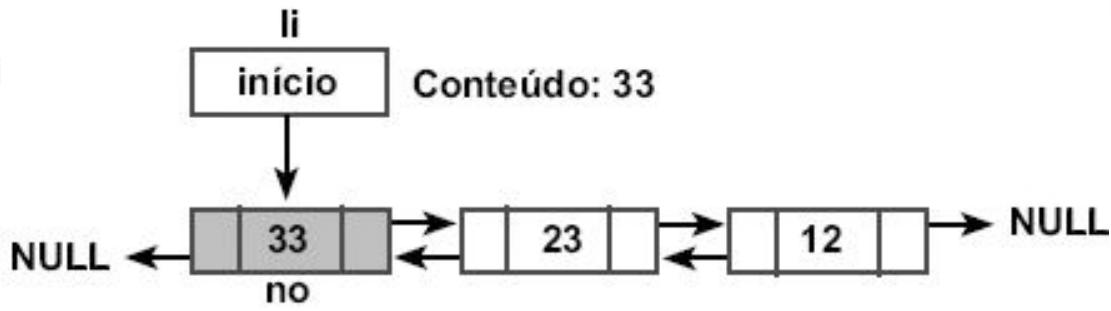
Lista Duplamente Encadeada: busca conteúdo

Busca pelo conteúdo do elemento

```
no = *li;
while(no != NULL && no->dados.matricula != mat)
    no = no->prox;
```

Verifica se o elemento foi
encontrado e o retorna

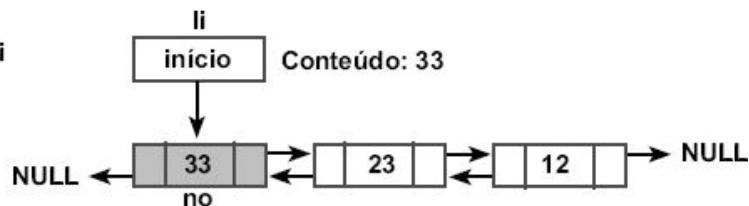
```
if(no == NULL)
    return 0;
else{
    *al = no->dados;
    return 1;
}
```



Lista Duplamente Encadeada: busca conteúdo

Busca um elemento por conteúdo

```
01 int busca_lista_mat(Lista* li, int mat, struct aluno *al){  
02     if(li == NULL)  
03         return 0;  
04     ELEM *no = *li;  
05     while(no != NULL && no->dados.matricula != mat){  
06         no = no->prox;  
07     }  
08     if(no == NULL)  
09         return 0;          Busca pelo conteúdo do elemento  
10    else{                no = *li;  
11        *al = no->dados;  while(no != NULL && no->dados.matricula != mat)  
12        return 1;          no = no->prox;  
13    }                      Verifica se o elemento foi  
14 }                          encontrado e o retorna  
                           if(no == NULL)  
                           return 0;  
                           else{  
                           *al = no->dados;  
                           return 1;  
                           }
```



Referências

- BACKES, André Ricardo. **Algoritmos e Estruturas de Dados em C**. Rio de Janeiro: LTC, 2023.

Obrigado!

raphaelguedes@ufg.br
raphaelguedes@inf.ufg.br

