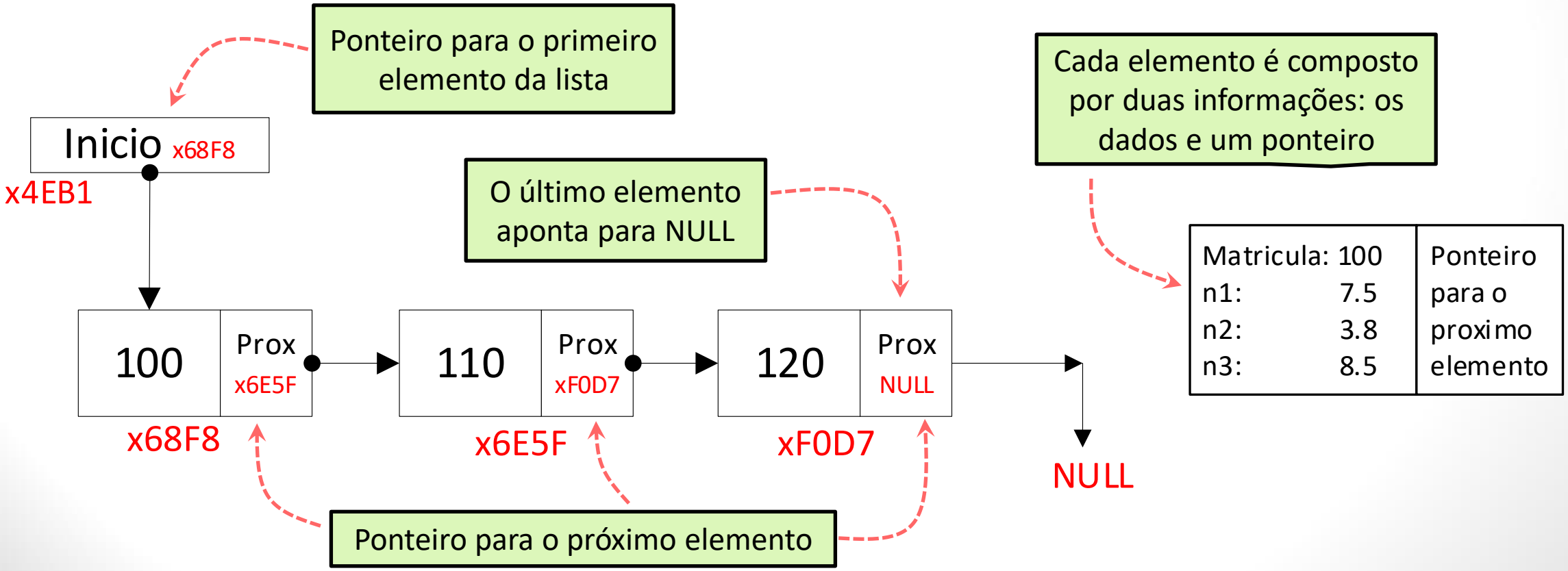
Estruturas de Dados 1

11 – Lista Sequencial Dinâmica – Linked List

Antonio Angelo de Souza Tartaglia
angelot@ifsp.edu.br

Lista Ligada

- Tipo de lista onde cada elemento que compõe a lisaponta para o seu sucessor na lista;
- Usa um ponteiro especial para o primeiro elemento da lista e uma indicação de final de Lista



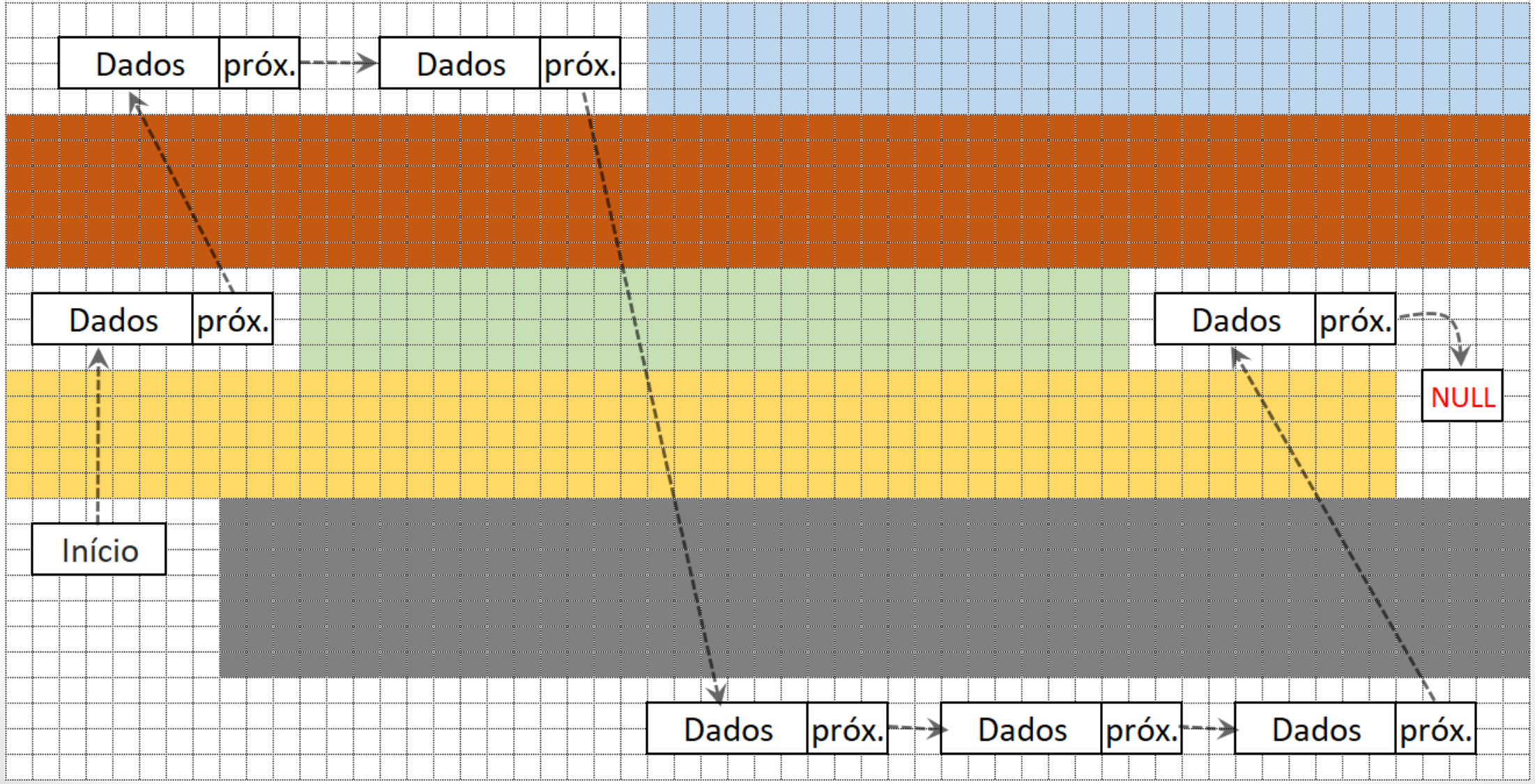
Estrutura de Dados 1

Lista Ligada

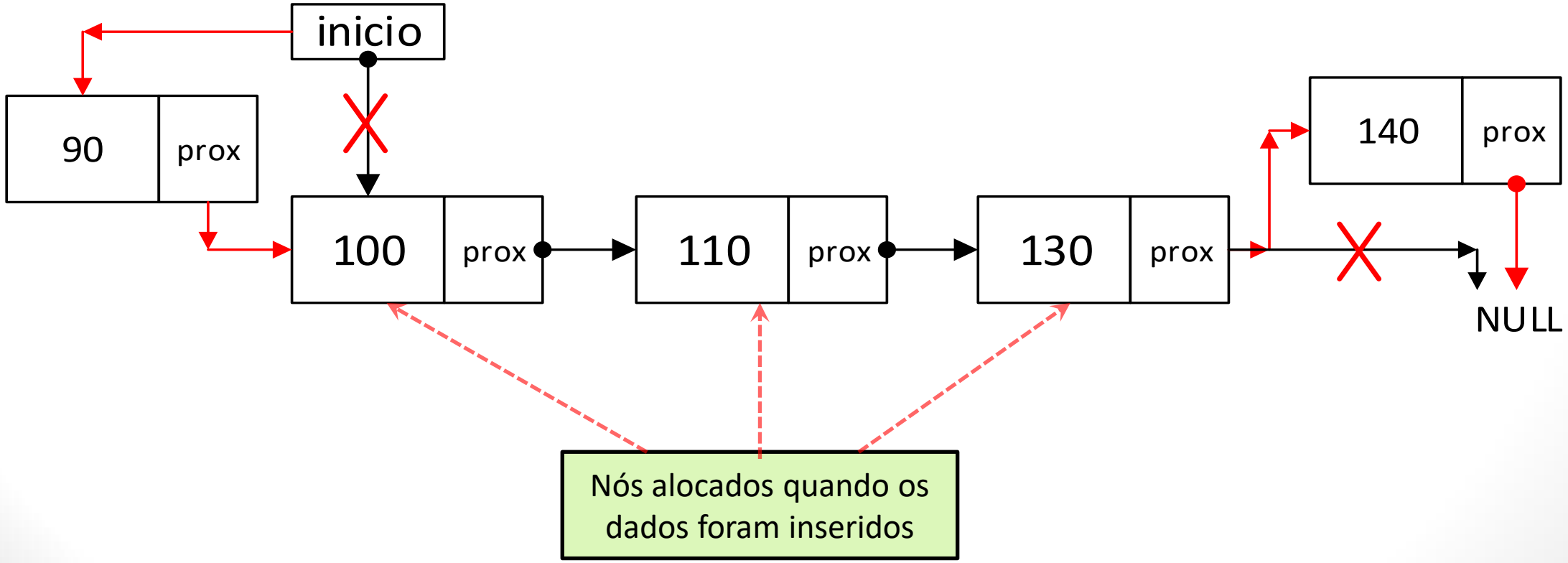
- Cada elemento é tratado como ponteiro que é alocado dinamicamente, a medida que os dados são inseridos;
- Para armazenar o primeiro elemento, utilizamos o ponteiro para ponteiro;
- Um ponteiro para ponteiro pode armazenar o endereço de outro ponteiro;
- Assim se torna fácil mudar o elemento que está no início da lista. Apenas muda-se o conteúdo do ponteiro que aponta para o outro ponteiro

Estrutura de Dados 1

Lista Ligada – Exemplo de ocupação em memória

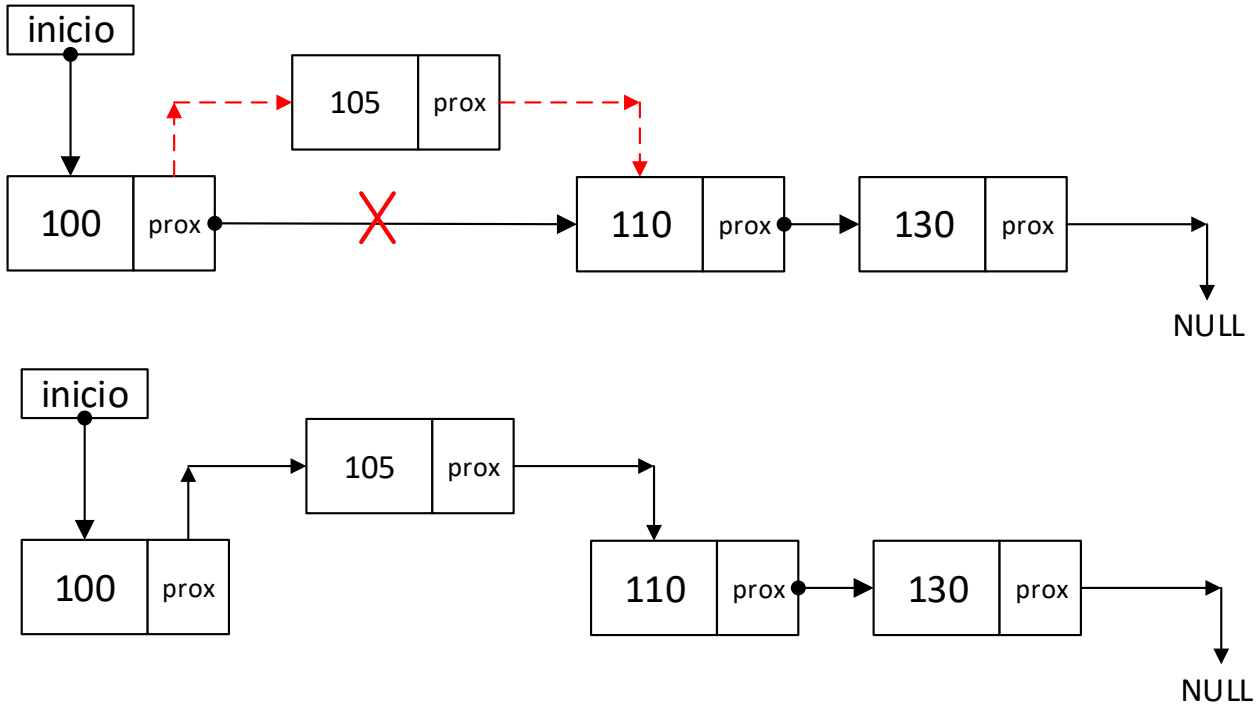


Lista Ligada



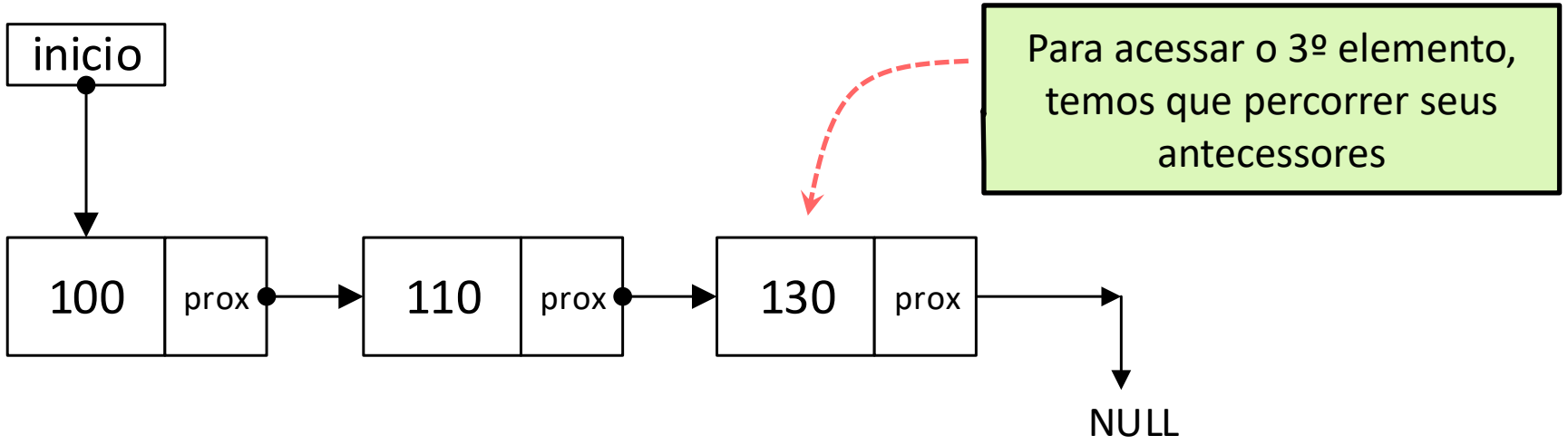
Lista Ligada

- Vantagens (em relação a lista estática):
  - Melhor utilização dos recursos de memória;
  - Não é necessário movimentar os elementos nas operações de inserção e remoção.



Lista Ligada

- Desvantagens:
  - Acesso indireto aos elementos;
  - Necessidade de percorrer a lista para acessar um elemento.



Lista Ligada

- Quando utilizar esta lista?
  - Quando não há necessidade de garantir um espaço mínimo para a execução do aplicativo;
  - Quando a inserção e remoção em lista ordenada são as operações mais frequentes.

É uma boa lista para inserções e remoções, pois não é necessário o deslocamento de nenhum elemento.

Lista Ligada - Implementação

- Implementação da Lista Ligada (ou Dinâmica Encadeada):
- Como a lista Estática, também neste tipo de lista implementaremos um TAD composto por 3 arquivos:
  - `main.c`
    - ✓ Gerenciará todo o programa;
  - `listaLigada.h`
    - ✓ Os protótipos das funções;
    - ✓ Tipo de dado armazenado na lista;
    - ✓ O ponteiro Lista.
  - `listaLigada.c`
    - ✓ O tipo de dados “Lista”;
    - ✓ Implementação de suas funções.

Protótipos das funções, disponíveis para o `main()`, que controlam o fluxo de informações armazenadas no dado encapsulado - Lista

Informações que serão armazenadas dentro do dado encapsulado - Lista

Endereço de memória da estrutura encapsulada Lista, que será devolvido ao `main()`

Tipo de dado Lista que está encapsulado

Funções disponibilizadas para armazenar dados em Lista

Lista Ligada - Implementação

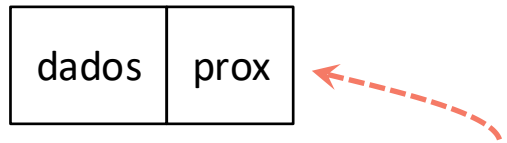
```c
//Arquivo listaLigada.h
typedef struct aluno{
    int matricula;
    float n1,n2,n3;
}ALUNO;

typedef struct elemento* Lista;
```

```c
//Arquivo main()
#include <stdio.h>
#include <stdlib.h>
#include "listaLigada.h"

int main(){
    Lista *li; //ponteiro para ponteiro
               //que está no arquivo
               //listaLigada.h
```

```c
//Arquivo listaLigada.c
#include <stdio.h>
#include <stdlib.h>
#include "listaLigada.h"
struct elemento{
    ALUNO dados;
    struct elemento *prox;
};
typedef struct elemento ELEM;
```



Lista Ligada – Criando a Lista

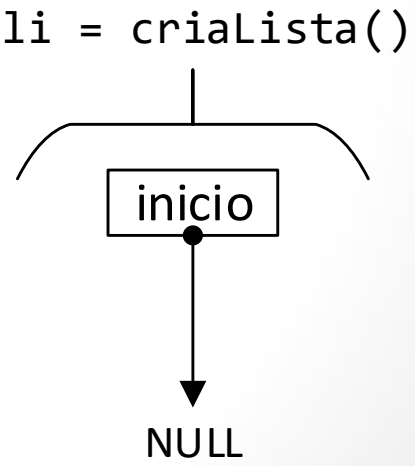
```c
//Arquivo listaLigada.h
Lista *criaLista();
```

```c
//Arquivo main()
li = criaLista();
```

```c
//Arquivo listaLigada.c
Lista *criaLista(){
    Lista *li;
    li = (Lista*) malloc(sizeof(Lista));
    if(li != NULL){
        *li = NULL;
    }
    return li;
}
```

Armazenará o endereço do primeiro elemento, e será devolvido ao `main()`

Se alocação ok, preenche o conteúdo com NULL



Lista Ligada – Liberando a Lista

```c
//Arquivo listaLigada.h
void apagaLista(Lista *li);
```

```c
//Arquivo main()
apagaLista(li);
```

Esta função deve ser a última a ser chamada pelo `main()`.

Recebe o endereço da lista na memória

Lista é válida se for diferente de NULL

```c
//Arquivo listaLigada.c
void apagaLista(Lista *li){
    if(li != NULL){
        ELEM *no;
        while((*li) != NULL){
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```

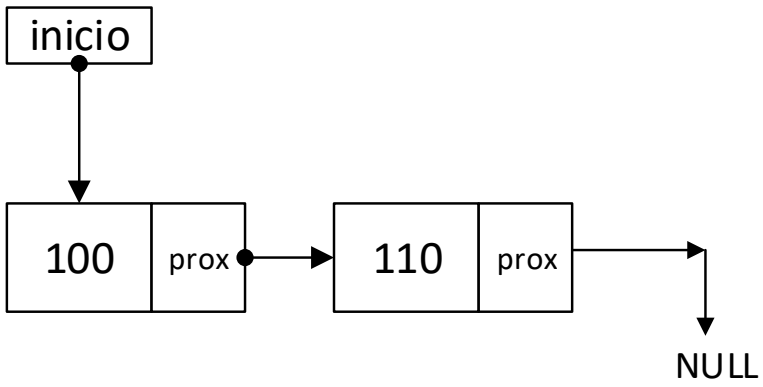
Enquanto o 1º elemento da lista for ≠

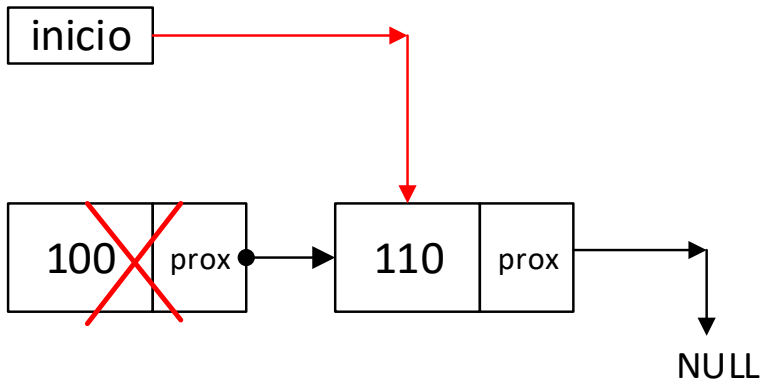
`while`: enquanto não estiver numa lista vazia, executa este conjunto de instruções, até que a lista esteja vazia apontando para NULL

Inicio da lista, aponta para próximo elemento da lista

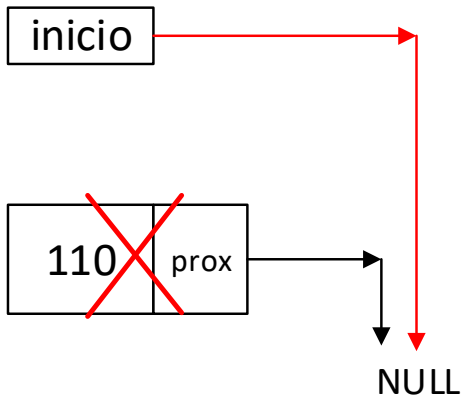
Ao final, libera a cabeça da lista (ponteiro especial), que aponta para o início

Lista Ligada – Liberando a Lista






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




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

Lista Ligada - Tamanho

- Obtendo informações básicas
  - Tamanho;
  - Se está cheia;
  - Se está vazia.

```c
//Arquivo listaLigada.h
int tamLista(Lista *li);
```

```c
//Arquivo main()
x = tamLista(li);
printf("O tamanho da lista e: %d", x);
```

```c
//Arquivo listaLigada.c
int tamLista(Lista *li){
    if(li == NULL){
        return 0;
    }
    int acum = 0;
    ELEM *no = *li;
    while(no != NULL){
        acum++;
        no = no->prox;
    }
    return acum;
}
```

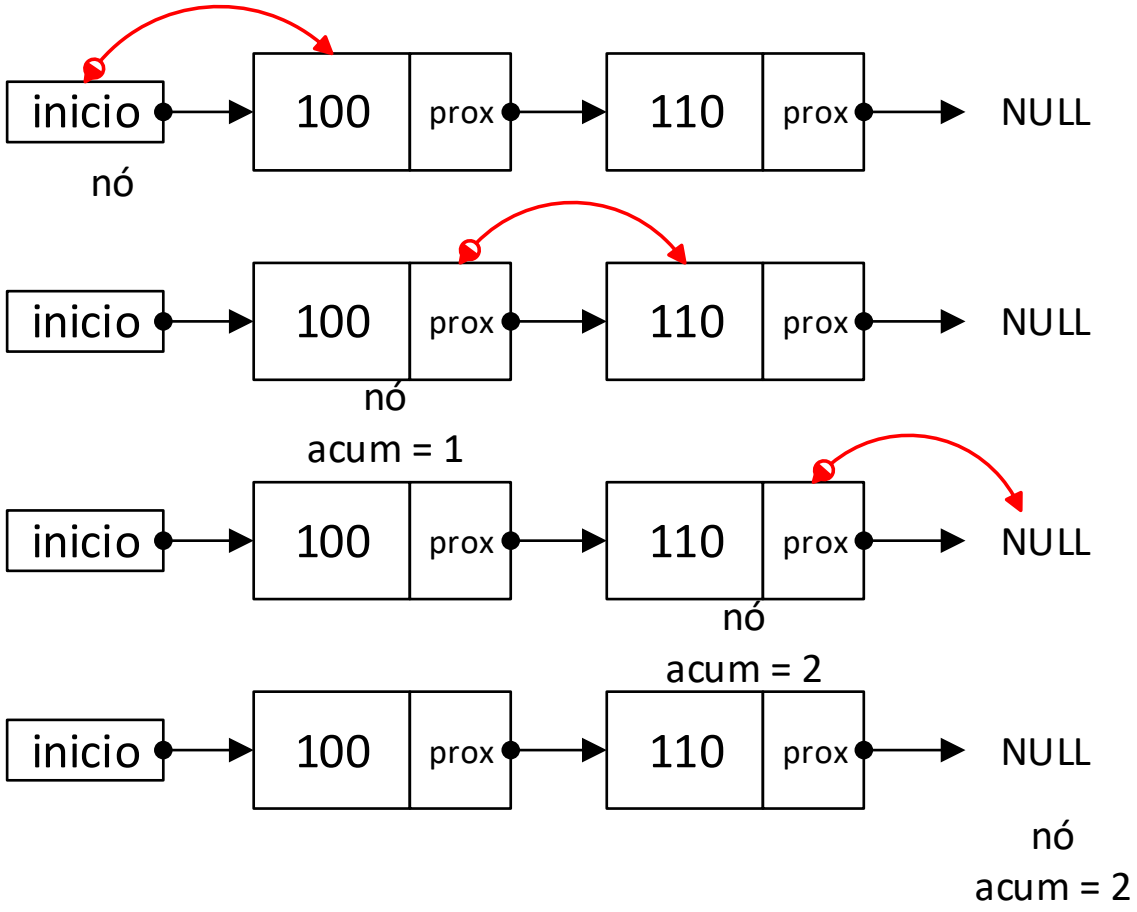
acum retorna a quantidade de elementos no que existem na lista

nó recebe 1º elemento da lista

Enquanto nó não for NULL, incrementa o acumulador e se desloca para o próximo nó

Nó é um auxiliar, e foi criado para preservar o inicio da lista, porque se andarmos com a cabeça da lista, perderemos o início da mesma. Sempre andamos por uma lista com elementos auxiliares para não perdermos informações.

Lista Ligada – Informações básicas




```
int acum = 0;
*no = *li;
```

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

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

```
Fim:
no == NULL;
```

Lista Ligada – Lista Cheia

- Em Listas Ligadas (dinâmicas encadeadas), não existe o conceito de lista cheia.
- A Lista estará cheia somente se toda a memória do computador estiver ocupada, cheia, ou seja acabar a memória disponível;
- Com Estruturas Dinâmicas, não faz sentido verificar se está cheia ou não. A função é mantida apenas por questões de compatibilidade com outras Estruturas do tipo Lista.

```
//Arquivo listaLigada.h
int listaCheia(Lista *li);
```

```
//Arquivo listaLigada.c
int listaCheia(Lista *li){
    return 0;
}
```

```
//Arquivo main()
if(listaCheia(li)){
    printf("\nLista esta cheia!");
}else{
    printf("\nLista esta vazia.");
}
```

Lista Ligada – Lista vazia

```c
//Arquivo listaLigada.h
int listaVazia(Lista *li);
```

```c
//Arquivo main()
if(listaVazia(li)){
    printf("\nLista esta vazia!");
}else{
    printf("\nLista nao esta vazia.");
}
```

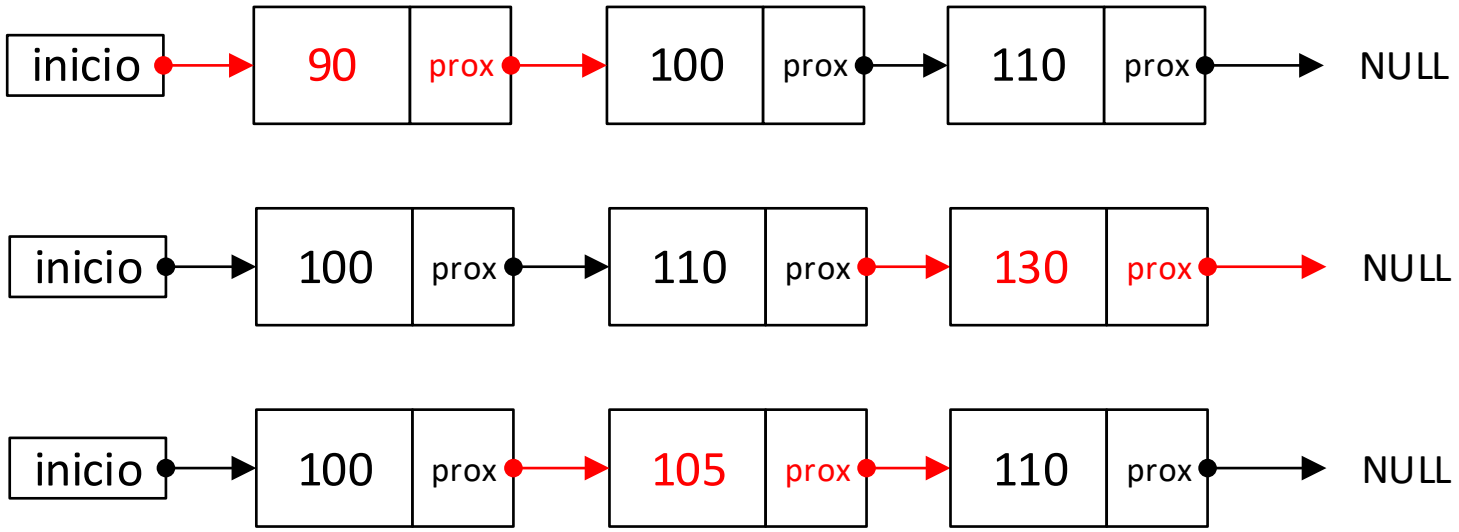
```c
//Arquivo listaLigada.c
int listaVazia(Lista *li){
    if(li == NULL){
        return 1;
    }
    if(*li == NULL){
        return 1;
    }
    return 0;
}
```

Se a lista for nula, ou seja não existir, informa que ela está vazia

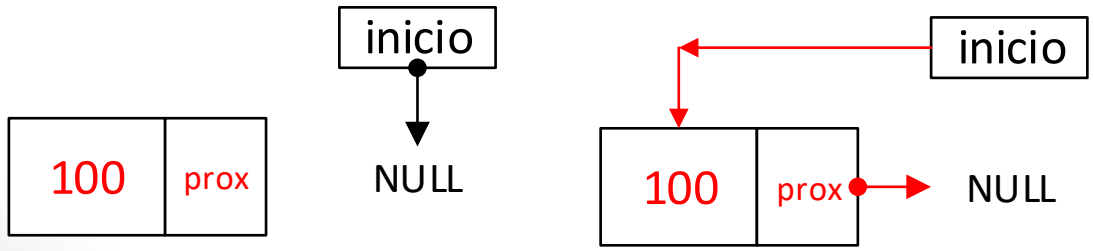
Se *li apontar para NULL, não existe nenhum elemento dentro da lista

Lista Ligada – Inserção

- Existem 3 formas de inserção:
  - Início;
  - Meio;
  - Fim.



- Também há o caso de inserção em Lista vazia:




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

Estrutura de Dados 1

- Dados para inserções e buscas. Devem ser criados no `main()`:

```
//Arquivo main()
//código de erro
int x = 0;
// para as buscas
int matricula = 130, posicao = 2;
//para popular a lista e al para consulta
ALUNO al1, al2, al3, al;

al1.matricula = 110;
al1.n1 = 5.6;
al1.n2 = 6.3;
al1.n3 = 7.9;
al2.matricula = 130;
al2.n1 = 9.2;
al2.n2 = 3.5;
al2.n3 = 8.1;
al3.matricula = 120;
al3.n1 = 6.6;
al3.n2 = 2.1;
al3.n3 = 9.2;
```

Receberá o código de erro

`matricula` será usado na busca por um elemento específico

`al1` será usado na inserção no início

`posicao` será usado na busca por um elemento em uma determinada posição

al - estrutura aluno ficará vazia e será utilizada para retorno de informações das buscas na lista

al2 será usado na inserção no final

al3 será usado na inserção ordenada

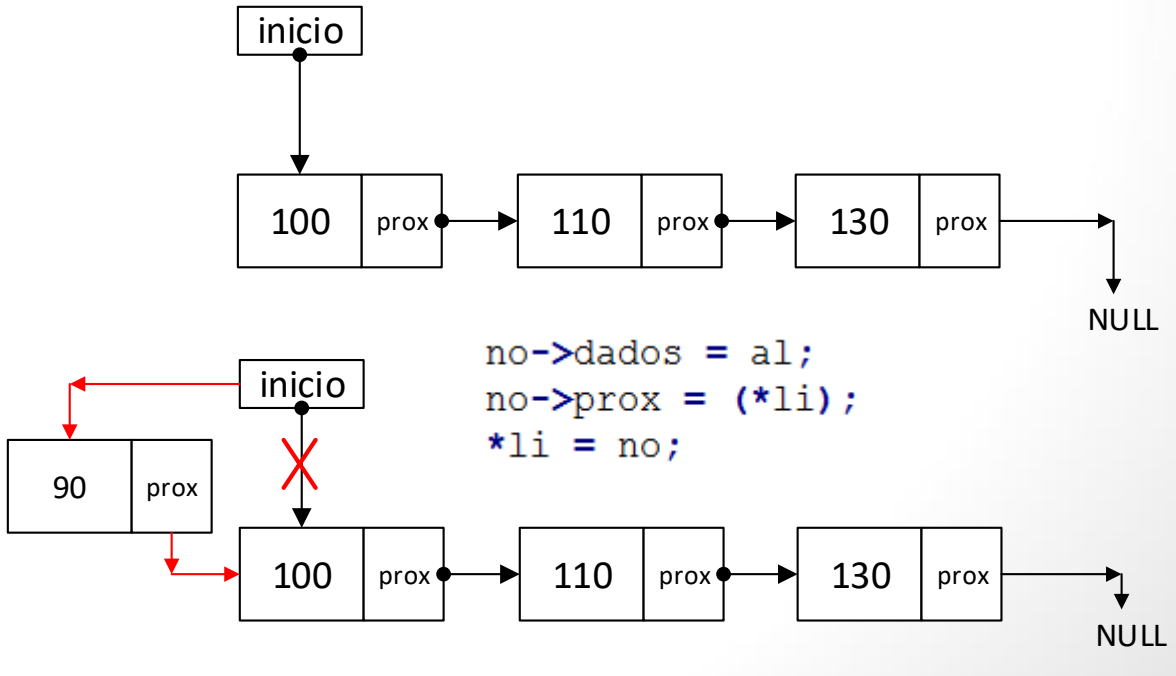
Lista Ligada – Inserção no início da Lista

```c
//Arquivo listaLigada.h
int insere_inicio_lista(Lista *li, ALUNO al);
```

```c
//Arquivo main()
x = insere_inicio_lista(li, al1);
if(x){
    printf("\nInserido no inicio com sucesso!");
}else{
    printf("\nNao foi possivel inserir no inicio.");
}
```

```c
//Arquivo listaLigada.c
int insere_inicio_lista(Lista *li, ALUNO al){
    if(li == NULL){
        return 0;
    }
    ELEM *no = (ELEM*) malloc(sizeof(ELEM));
    if(no == NULL){
        return 0;
    }
    no->dados = al;
    no->prox = (*li);
    *li = no;
    return 1;
}
```

Resolve inserção no início da lista e em uma lista vazia



Lista Ligada – Inserção no final da Lista

```c
//Arquivo listaLigada.c
int insere_final_lista(Lista *li, ALUNO al){
    if(li == NULL){
        return 0;
    }
    ELEM *no = (ELEM*) malloc(sizeof(ELEM));
    if(no == NULL){
        return 0;
    }
    no->dados = al;
    no->prox = NULL;
    if((*li) == NULL){//lista vazia, insere no inicio
        *li = no;
    }else{
        ELEM *aux = *li;
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        aux->prox = no;
    }
    return 1;
}
```

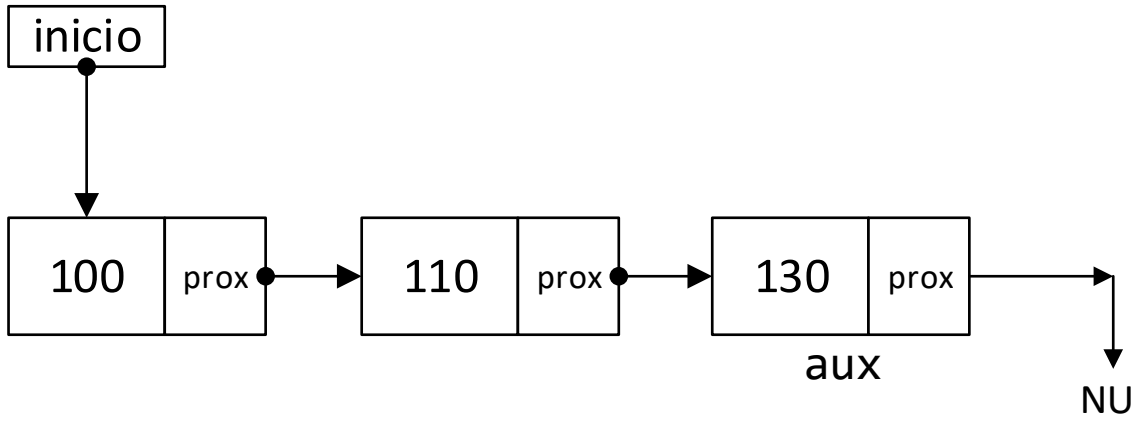
```c
//Arquivo listaLigada.h
int insere_final_lista(Lista *li, ALUNO al);
```

```c
//Arquivo main()
x = insere_final_lista(li, al2);
if(x){
    printf("\nInserido no final com sucesso!");
}else{
    printf("\nNao foi possivel inserir no final.");
}
```

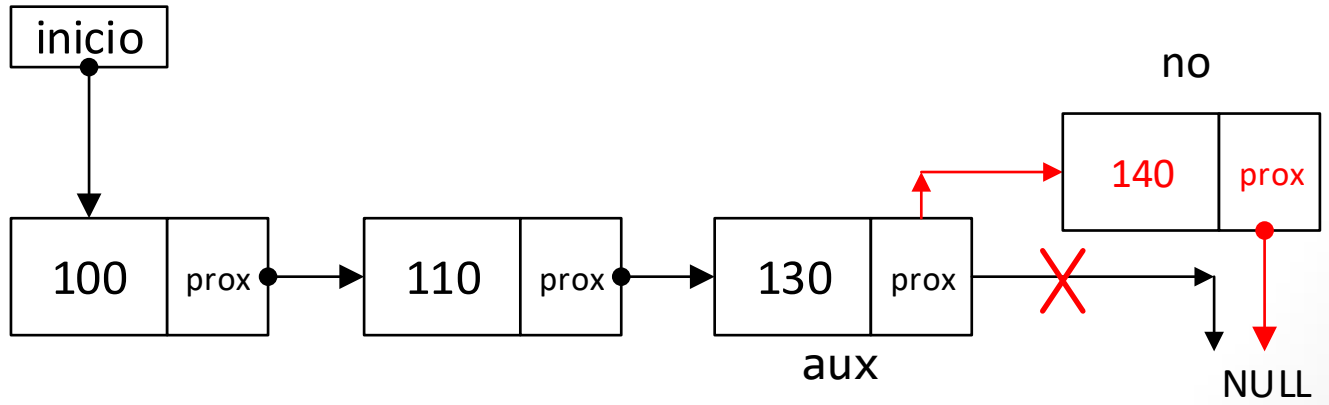
Percorre a lista com um nó auxiliar para preservar a cabeça da lista

Lista Ligada – Inserção no final da Lista

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




```
//insere dados depois de aux
no->dados = al;
no->prox = NULL;
aux->prox = no;
```



Estrutura de Dados 1

Lista Ligada – Inserção ordenada

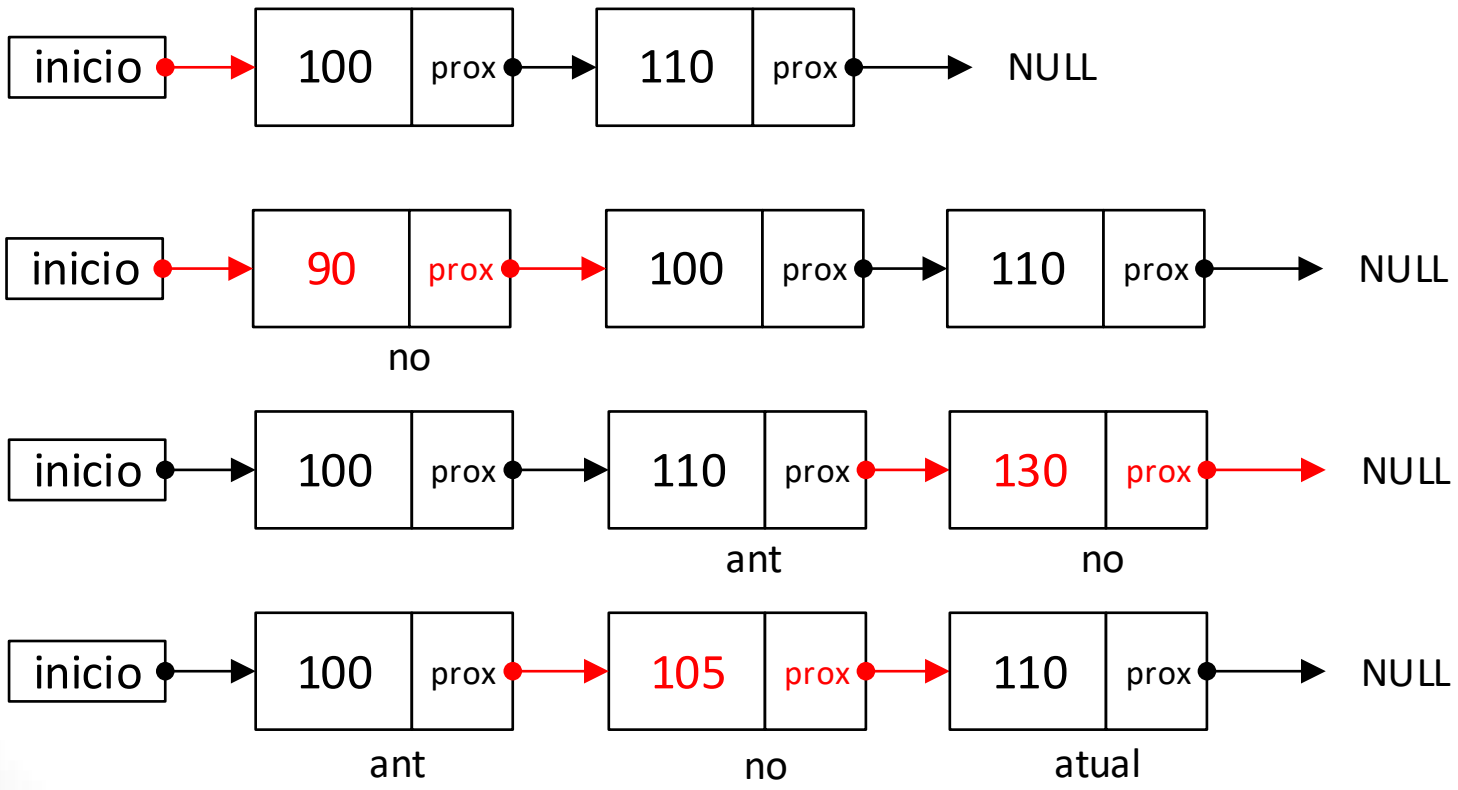
```c
//Arquivo listaLigada.h
int insere_lista_ordenada(Lista *li, ALUNO al);
```

```c
//Arquivo main()
x = insere_lista_ordenada(li, al3);
if(x){
    printf("\nInserido ordenadamente com sucesso!");
}else{
    printf("\nNao foi possivel inserir ordenadamente.");
}
```

```c
//Arquivo listaLigada.c
int insere_lista_ordenada(Lista *li, ALUNO al){
    if(li == NULL){
        return 0;
    }
    ELEM *no = (ELEM*) malloc(sizeof(ELEM));
    if(no == NULL){
        return 0;
    }
    no->dados = al;
    if(listaVazia(li)){//insere no inicio
        no->prox = (*li);
        *li = no;
        return 1;
    }else{
        ELEM *ant, *atual = *li;
        while(atual != NULL && atual->dados.matricula < al.matricula){
            ant = atual; // posiciona entre os nós
            atual = atual->prox;
        }
        if(atual == *li){ //insere se estiver na primeira posição
            no->prox = (*li);
            *li = no;
        }else{  //insere em qualquer outra posição
            no->prox = ant->prox;
            ant->prox = no;
        }

        return 1;
    }
}
```

Lista Ligada – Inserção ordenada



Busca onde inserir

Inserir no início

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

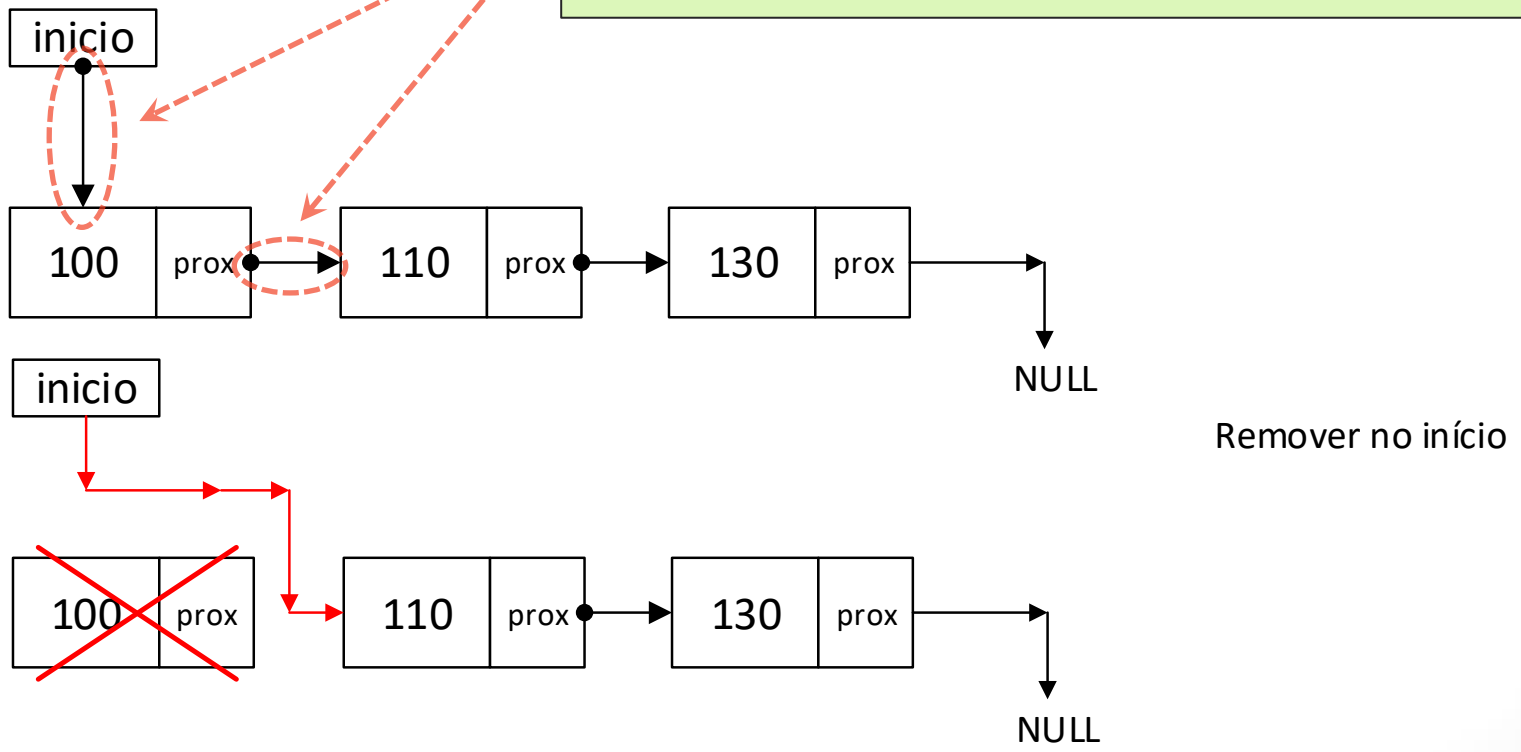
Inserir depois de ant

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

Lista Ligada – Remoção

- Existem 3 tipos de remoção:
  - Início
  - Meio
  - Final

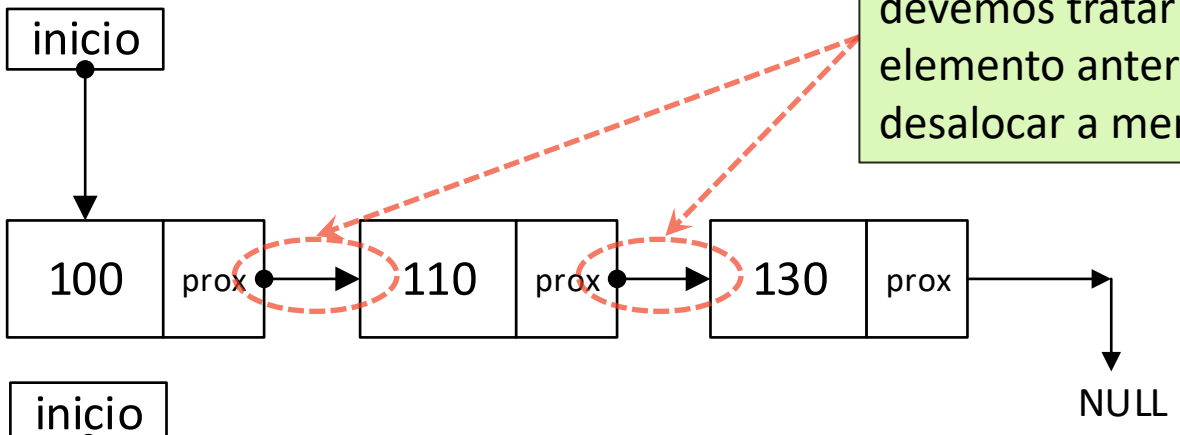
Em uma remoção no início, devemos fazer com que o ponteiro *início* aponte para o próximo elemento, e em seguida, desalocar a memória do elemento a ser excluído.



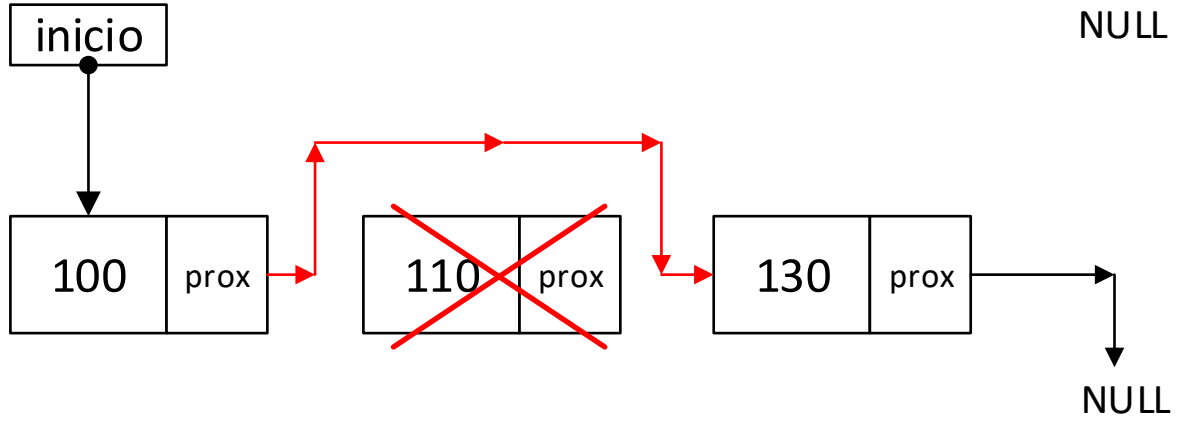
Remover no início

Lista Ligada – Remoção

Em uma remoção de um elemento qualquer, devemos tratar as ligações dos ponteiros do elemento anterior e do posterior, para depois desalocar a memória do elemento a se remover

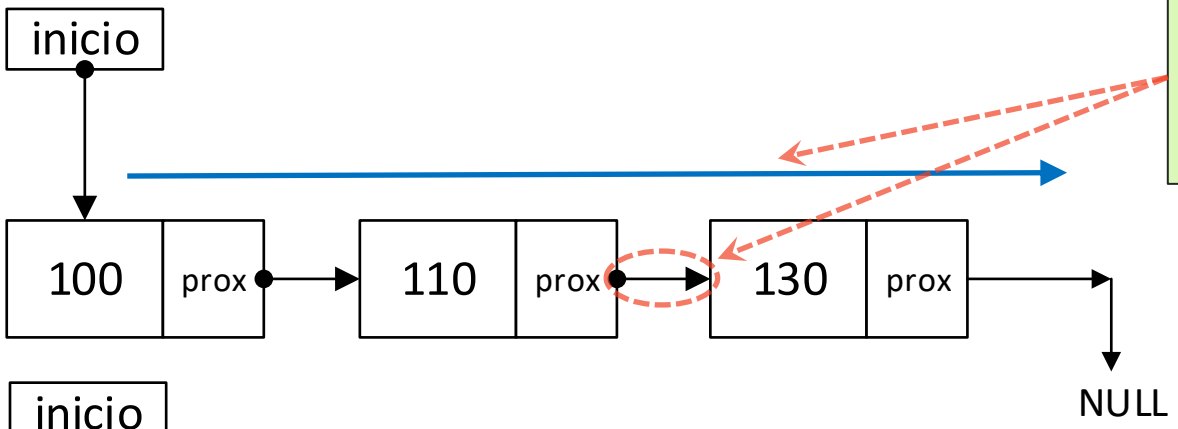


Remover no meio

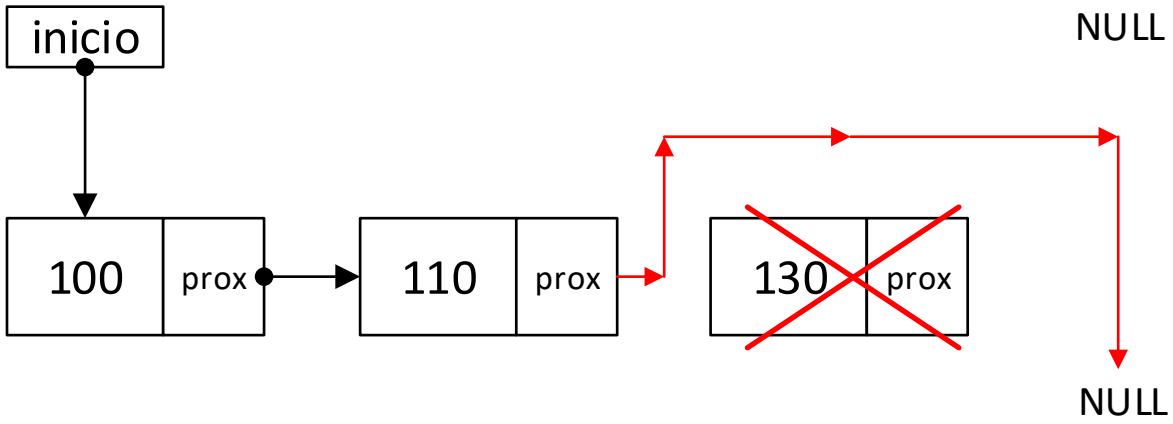


Lista Ligada – Remoção

Em uma remoção no final, é necessário percorrer toda a lista para encontrar o último elemento, tratar a ligação do elemento anterior para que aponte para NULL, e então remover o último.



Remover no final

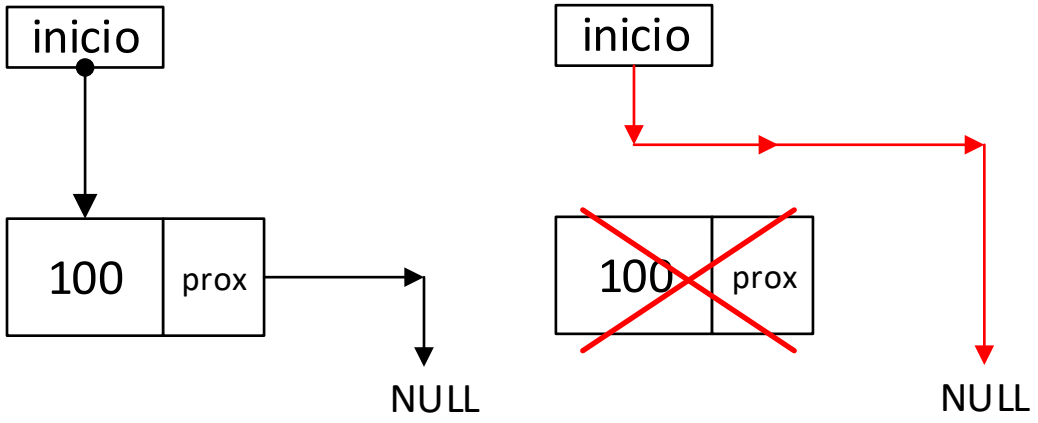


Lista Ligada – Remoção

- Os 3 tipos de remoção trabalham juntos. A remoção sempre remove um, e apenas um, elemento específico da lista, o qual pode estar no início, no meio ou no final da lista;

Cuidado; não se pode remover de uma lista vazia;

Removendo o último nó, a lista fica vazia, é necessário tratamento.



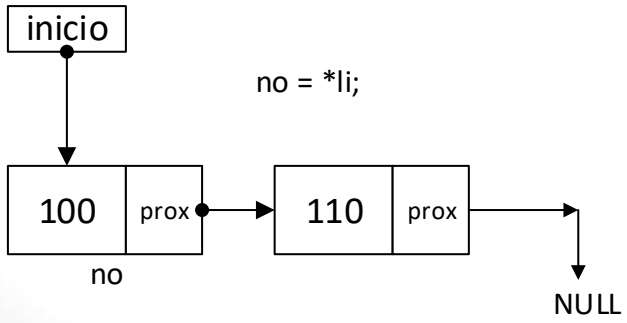
Lista Ligada – Remoção no início da Lista

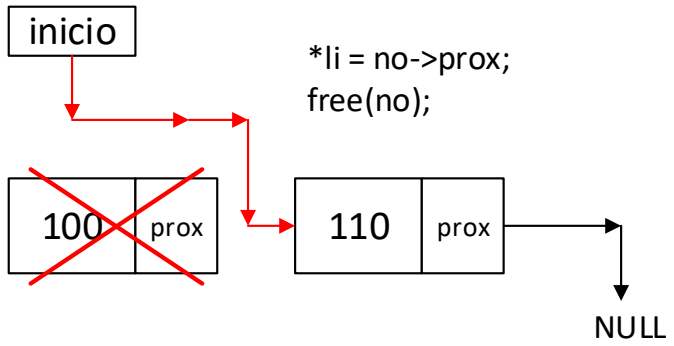
```c
//Arquivo listaLigada.c
int remove_inicio_lista(Lista *li){
    if(li == NULL){ //verifica se a lista existe
        return 0;
    }
    if(*li == NULL){//verifica se a lista está vazia
        return 0;
    }
    ELEM *no = *li;
    *li = no->prox;
    free(no);
    return 1;
}
```

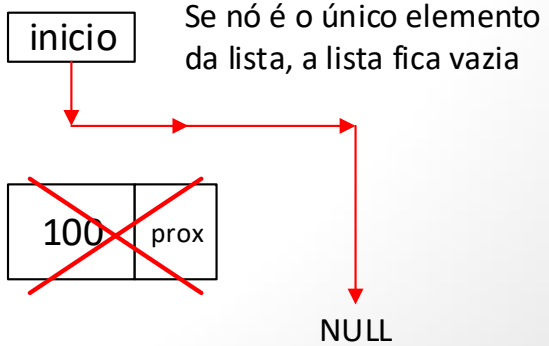
Resolve os dois casos, se a lista vai ficar vazia ou não.

```c
//Arquivo listaLigada.h
int remove_inicio_lista(Lista *li);
```

```c
//Arquivo main()
x = remove_inicio_lista(li);
if(x){
    printf("\nRemovido do inicio com sucesso!");
}else{
    printf("\nNao foi possivel remover do inicio.");
}
```







Lista Ligada – Remoção no final da Lista

```c
//Arquivo listaLigada.c
int remove_final_lista(Lista *li){
    if(li == NULL){
        return 0;
    }
    if((*li) == NULL){//lista vazia
        return 0;
    }
    ELEM *ant, *no = *li;
    while(no->prox != NULL){
        ant = no;
        no = no->prox;
    }
    if(no == (*li)){//remove o primeiro?
        *li = no->prox; //o proximo é NULL
    }else{
        ant->prox = no->prox;
    }
    free(no);
    return 1;
}
```

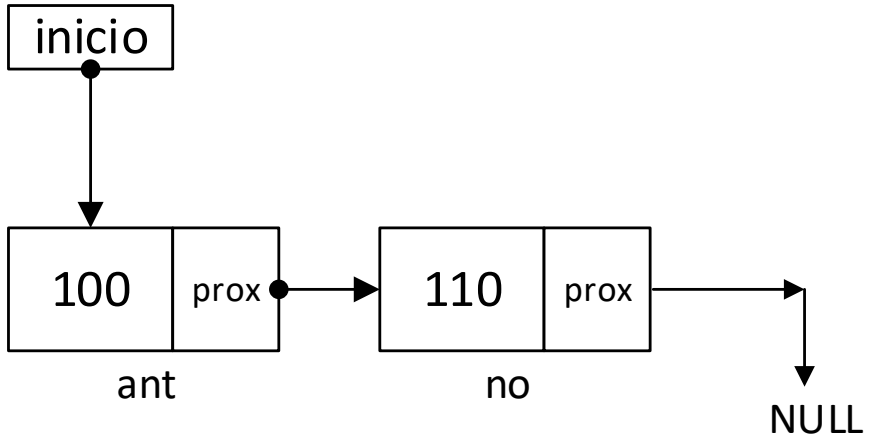
Percorre a lista até o final.

Nó ant->prox passa a apontar para onde no->prox aponta

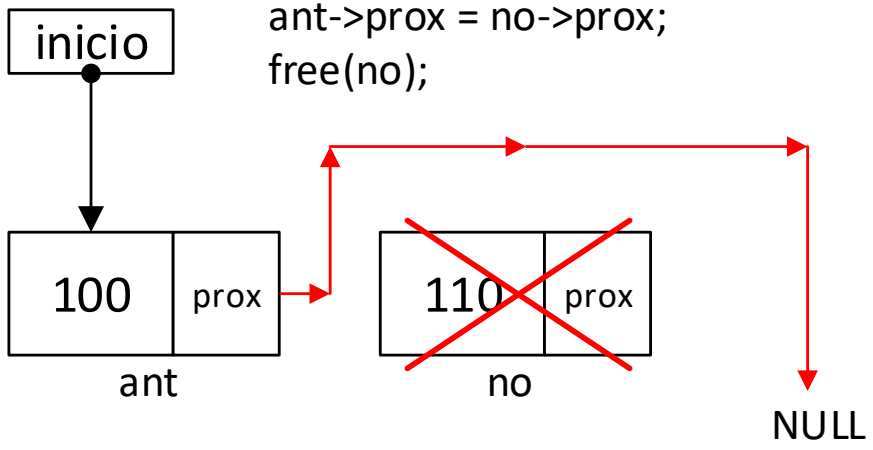
```c
//Arquivo listaLigada.h
int remove_final_lista(Lista *li);
```

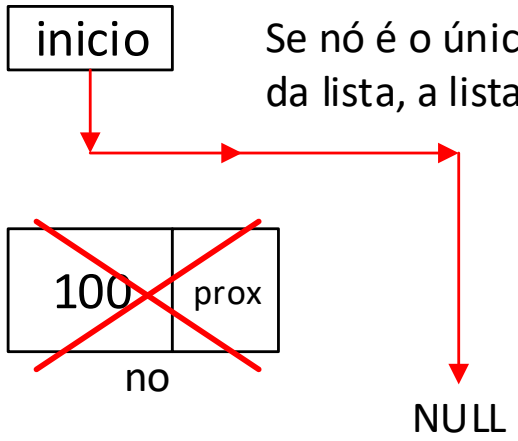
```c
//Arquivo main()
x = remove_final_lista(li);
if(x){
    printf("\nRemovido do final com sucesso!");
}else{
    printf("\nNao foi possivel remover do final.");
}
```

Lista Ligada – Remoção no final da Lista




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





Lista Ligada – Remoção de qualquer elemento

```c
//Arquivo listaLigada.c
int remove_lista(Lista *li, int mat){
    if(li == NULL){
        return 0;
    }
    ELEM *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    //sai do while por 2 motivos: lista chegou ao fim/vazia
    //ou encontrou elemento a remover
    if(no == NULL){
        return 0;
    }
    //se no parado na 1ª posição, remover o primeiro?
    if(no == *li){
        *li = no->prox;
    }else{
        //ant vai apontar para elemento seguinte ao nó
        ant->prox = no->prox;
    }
    free(no);
    return 1;
}
```

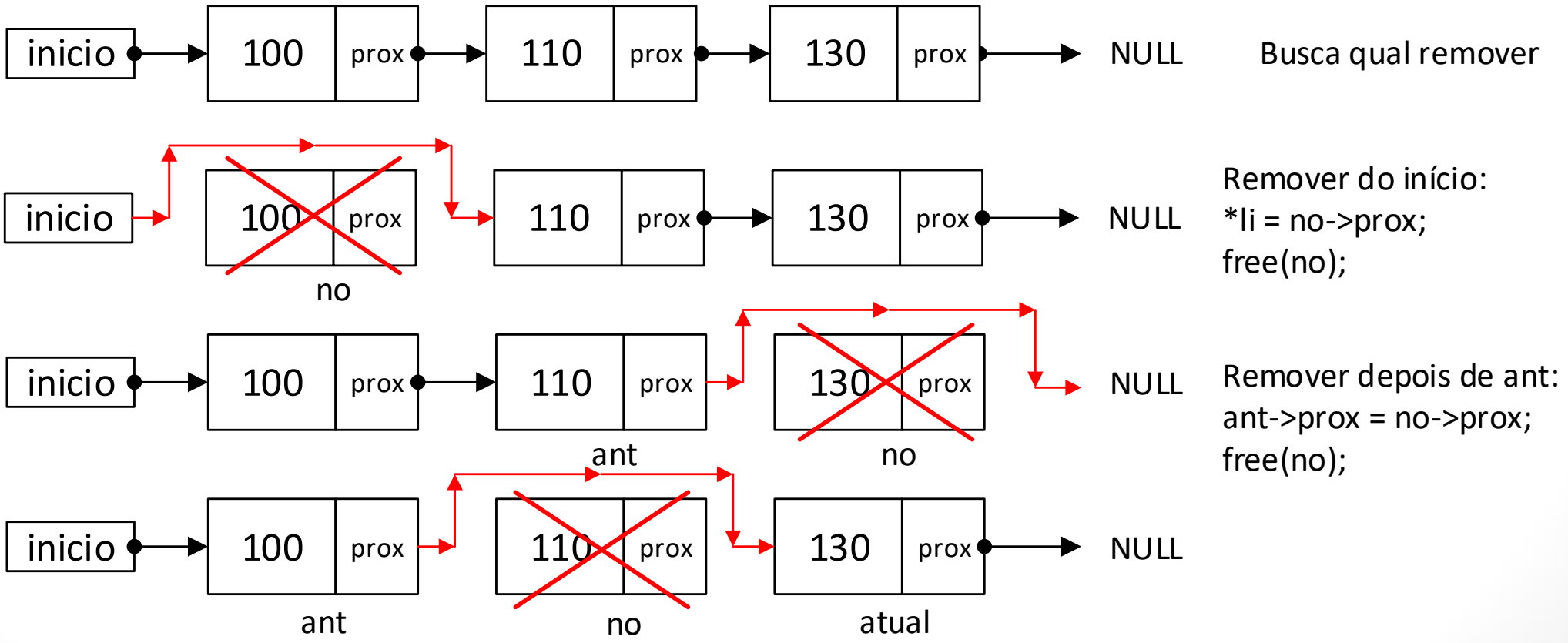
Remove no início, meio e fim

```c
//Arquivo listaLigada.h
int remove_lista(Lista *li, int mat);
```

Nesta função não é necessário testar se a lista estava vazia para remover, o `while` faz esse tratamento.

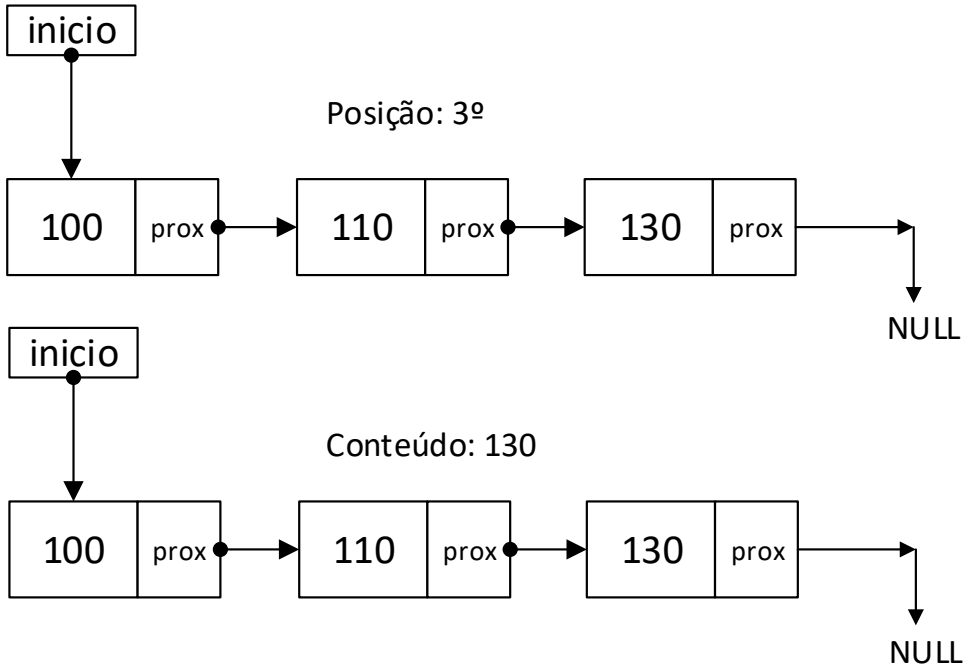
```c
//Arquivo main()
x = remove_lista(li, matricula);
if(x){
    printf("\nRemovido elemento com sucesso!");
}else{
    printf("\nNao foi possivel remover o elemento.");
}
```

Lista Ligada – Remoção de qualquer elemento



Lista Ligada – Consultas

- Existem 2 maneiras de consultar um elemento de uma lista:
  - Pela posição;
  - Pelo conteúdo.
- Ambas dependem de busca, portanto, temos que percorrer os elementos até encontrarmos o desejado:



Lista Ligada – Consultas por posição

```c
//Arquivo listaLigada.h
int consulta_lista_pos(Lista *li, int posicao, ALUNO *al);
```

```c
//Arquivo listaLigada.c
int consulta_lista_pos(Lista *li, int posicao, ALUNO *al){
    if(li == NULL || posicao <= 0){
        return 0;
    }
    ELEM *no = *li;
    int i = 1;
    //percorre a lista com i, a procura do elemento:
    while (no != NULL && i < posicao){
        no = no->prox;
        i++;
    }
    //trata-se lista vazia, ou não encontrou elemento:
    if(no == NULL){
        return 0;
    }else{
        //se nó != de NULL, então encontrou elemento:
        *al = no->dados;
        return 1;
    }
}
```

```c
//Arquivo main()
x = consulta_lista_pos(li, posicao, &al);
if(x){
    printf("\n\nConteudo na posicao %d:", posicao);
    printf("\n%d", al.matricula);
    printf("\n%.2f", al.n1);
    printf("\n%.2f", al.n2);
    printf("\n%.2f", al.n3);
}else{
    printf("\nElemento %d nao encontrado.", posicao);
}
```

Lista Ligada – Consultas por conteúdo

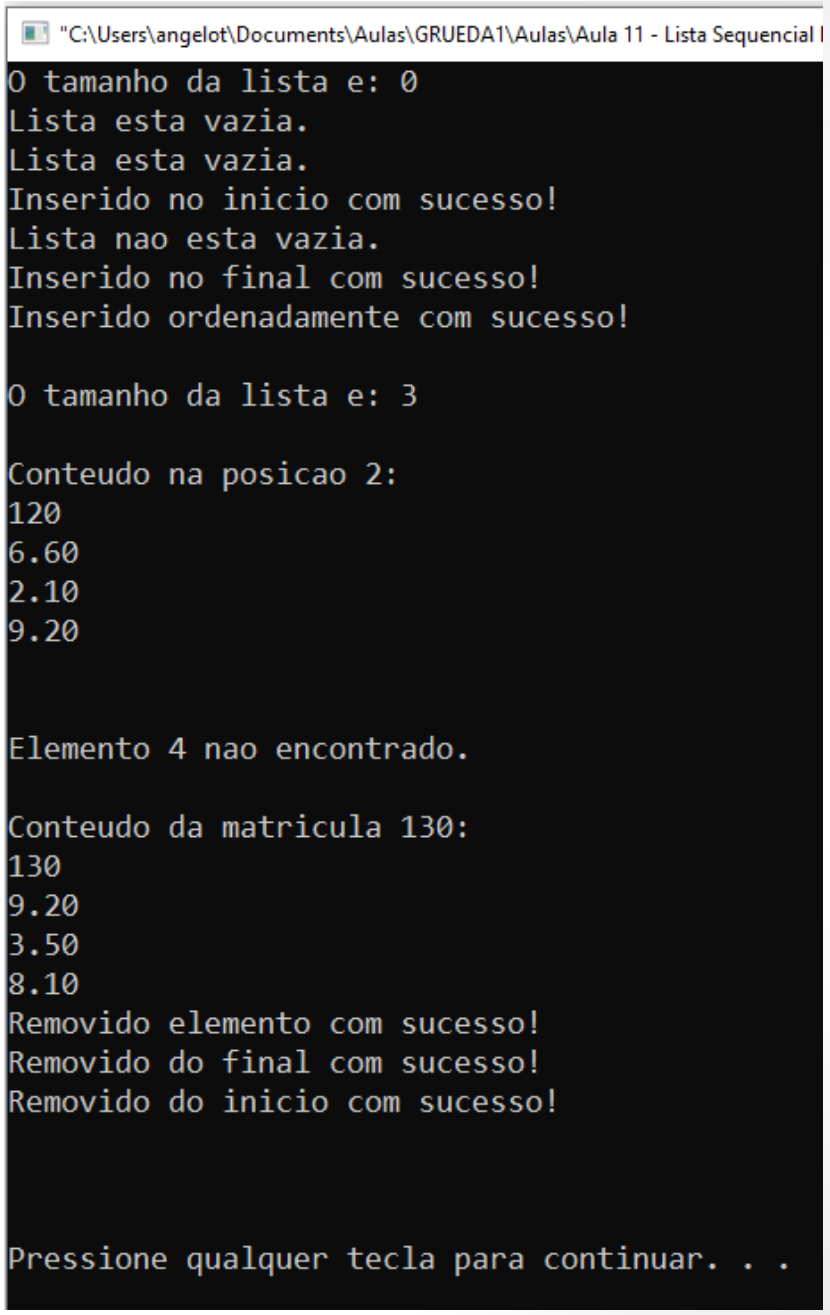
```c
//Arquivo listaLigada.h
int consulta_lista_mat(Lista *li, int matricula, ALUNO *al);
```

```c
//Arquivo listaLigada.c
int consulta_lista_mat(Lista *li, int mat, ALUNO *al){
    if(li == NULL){
        return 0;
    }
    ELEM *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        no = no->prox;
    }
    if(no == NULL){
        return 0;
    }else{
        *al = no->dados;
        return 1;
    }
}
```

```c
//Arquivo main()
x = consulta_lista_mat(li, matricula, &al);
if(x){
    printf("\n\nConteudo da matricula %d:", matricula);
    printf("\n%d", al.matricula);
    printf("\n%.2f", al.n1);
    printf("\n%.2f", al.n2);
    printf("\n%.2f", al.n3);
}else{
    printf("\nMatricula %d nao encontrada.", matricula);
}
```

Estrutura de Dados 1

- Execução do programa Lista-Ligada:



Atividade

- Monte o programa com todas as funções apresentadas pertencente à lista ligada.
- Implemente nesta lista a mesma função `coletadados()`, do exercício anterior (lista estática).
- Crie um menu que funcione ininterruptamente com 4 opções:
  - Incluir um elemento de forma ordenada utilizando a função `coletadados();`
  - Buscar um elemento por conteúdo (matrícula);
  - Excluir um elemento de forma ordenada, e;
  - Encerrar o programa.
- Entregue os arquivos (zipados) na plataforma Moodle como atividade 1.