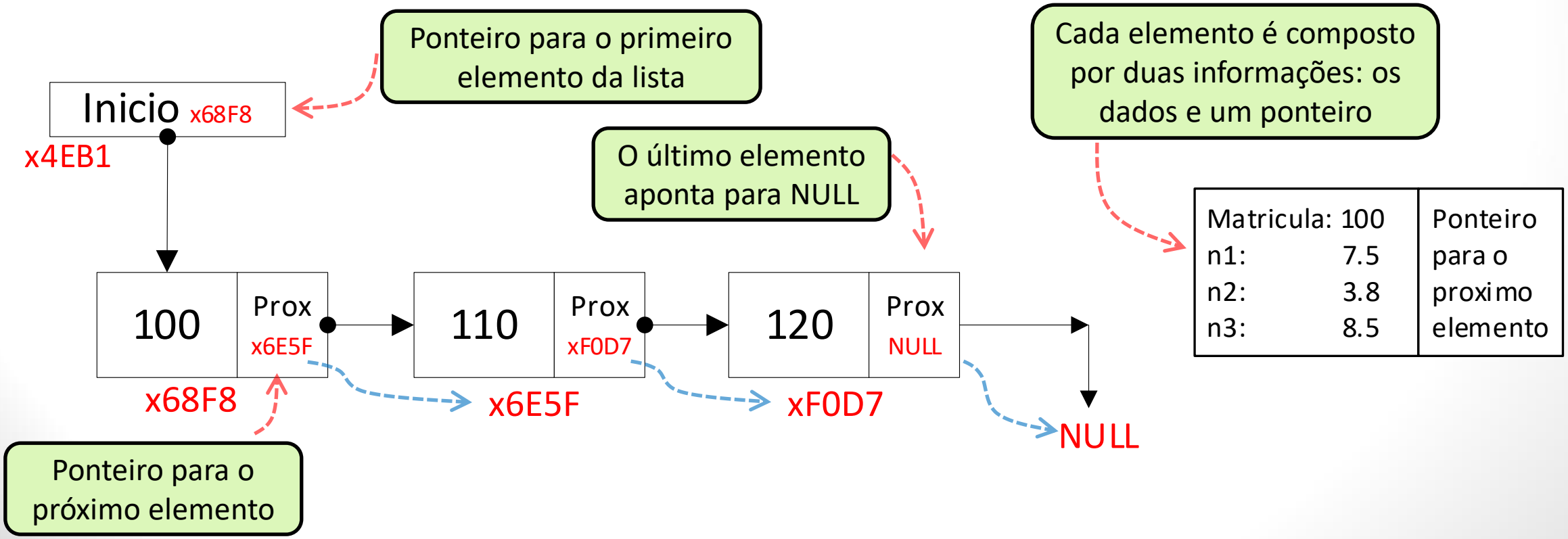
Estruturas de Dados 1

11 – Lista Sequencial Dinâmica – Linked List

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

Lista Ligada

- Tipo de estrutura de dados onde cada elemento que a compõe, aponta para o seu sucessor dentro da lista;
- Usa um ponteiro especial que aponta para o primeiro elemento da lista e uma indicação de final de Lista:

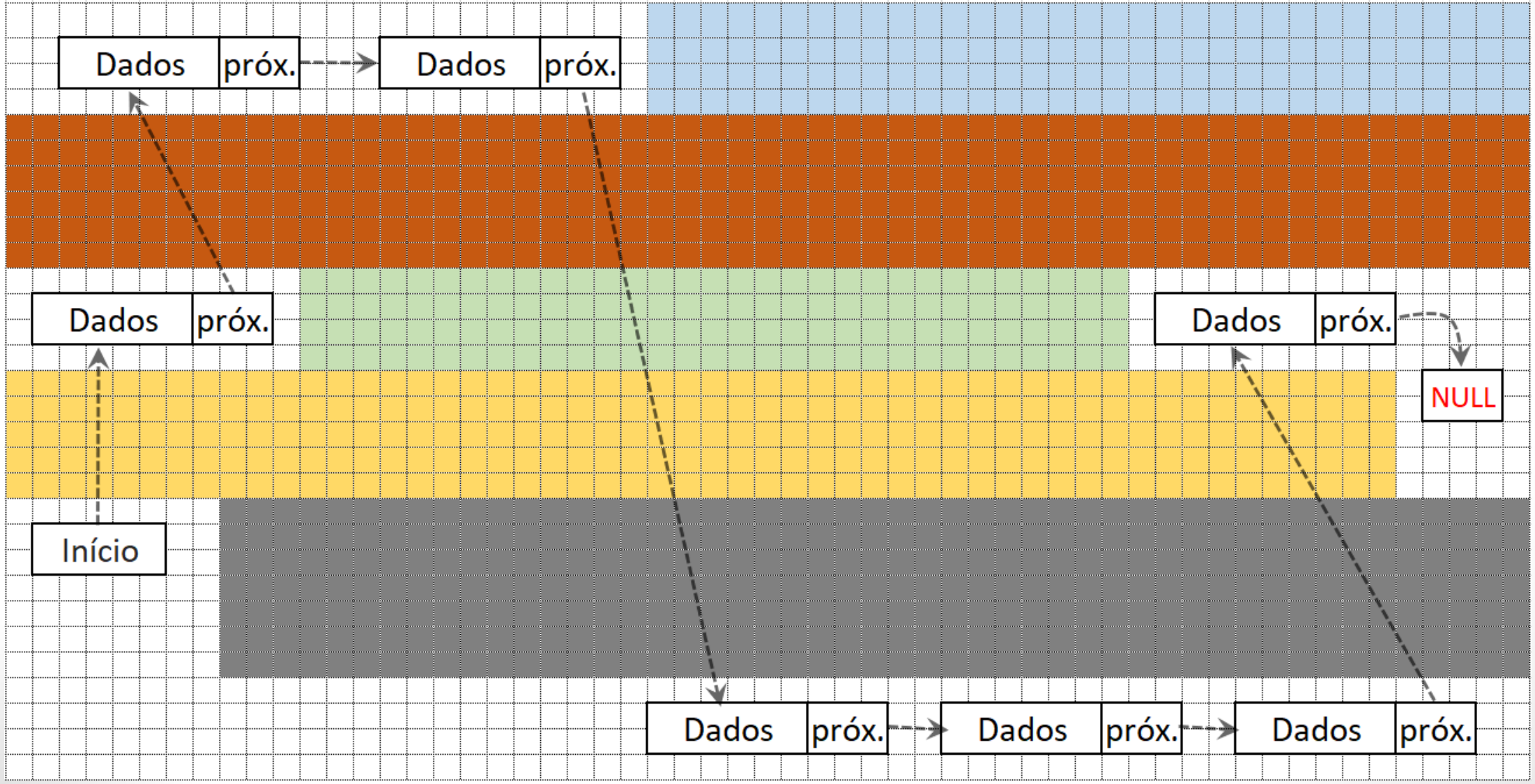


Lista Ligada

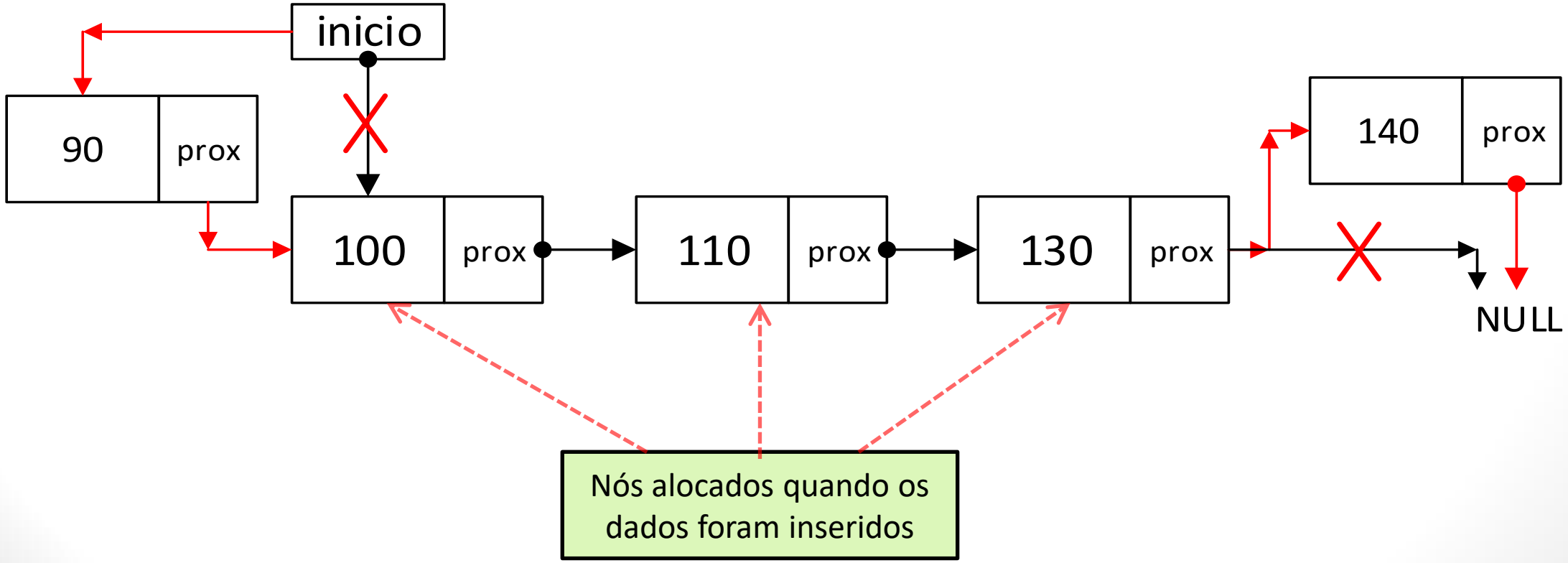
- **Cada elemento é tratado como ponteiro** que é alocado dinamicamente, a medida que os dados são inseridos;
- Para armazenar o primeiro elemento, utilizamos o ponteiro para ponteiro;
- Um ponteiro para ponteiro pode armazenar o endereço de outro ponteiro, e assim se torna fácil mudar o elemento que está no início da lista: apenas muda-se o conteúdo, ou seja, o endereço que está armazenado no ponteiro para ponteiro, que então passa a apontar para o outro ponteiro.

Estrutura de Dados 1

Lista Ligada – Exemplo de ocupação em memória

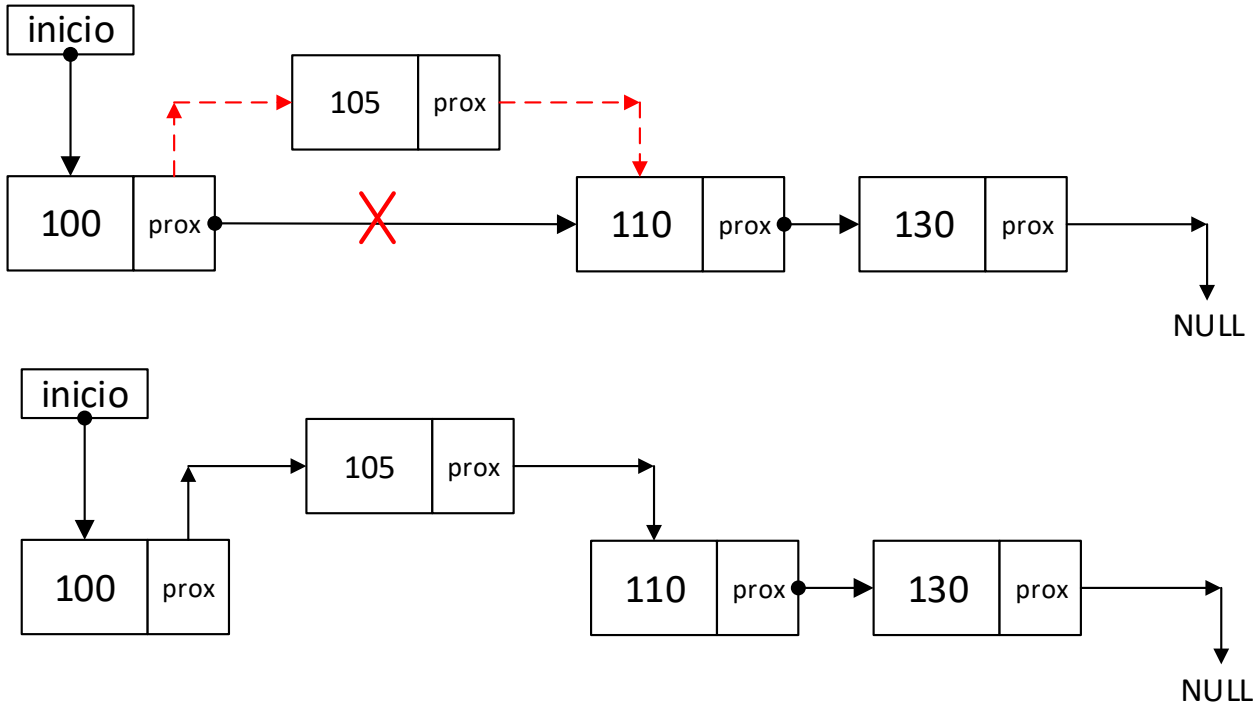


Lista Ligada



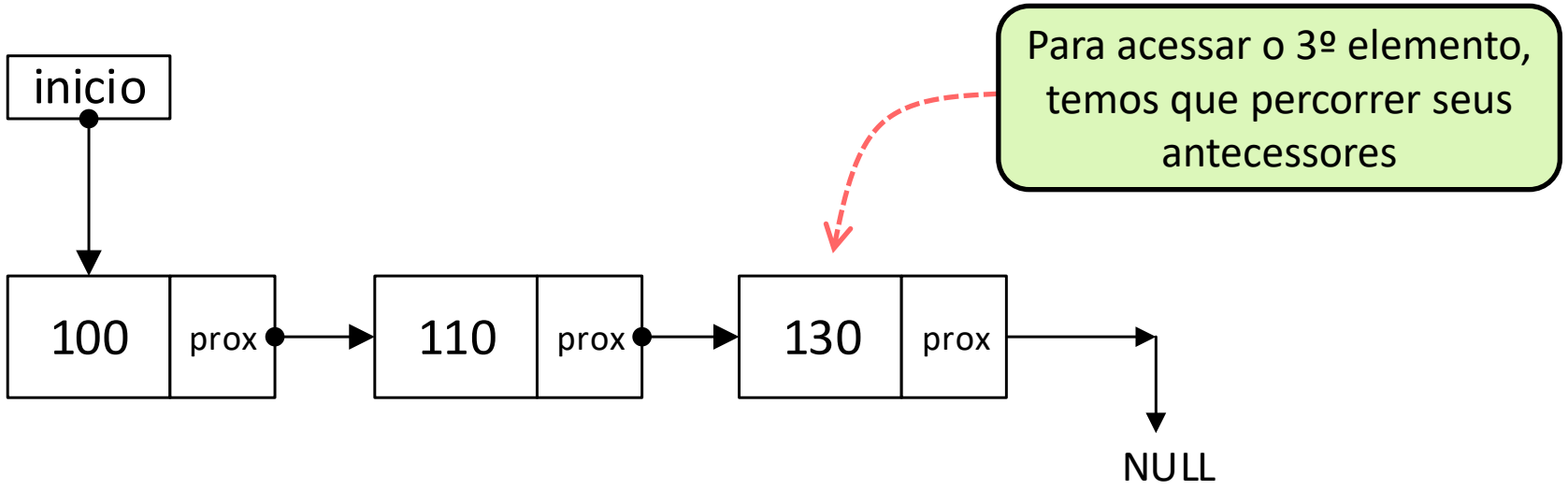
Lista Ligada

- 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.

Funções, disponibilizadas para o `main()`, que controlam o fluxo de informações armazenadas no dado encapsulado - `Lista`

Informações que serão armazenadas dentro do dado encapsulado - `Lista`

Armazenará o endereço de memória da estrutura encapsulada Lista, que será devolvido ao `main()`

Tipo de dado Lista que está encapsulado

Funções disponibilizadas para manipular os dados em Lista

Lista Ligada - Implementação

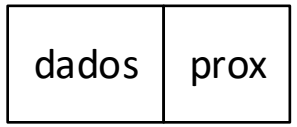
```c
//Arquivo listaLigada.h
typedef struct aluno{
    int matricula;
    float n1,n2,n3;
}ALUNO;

typedef struct elemento* Lista;
```

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

int main(){
    Lista *li = NULL; //ponteiro para ponteiro
                      //parte 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();
```

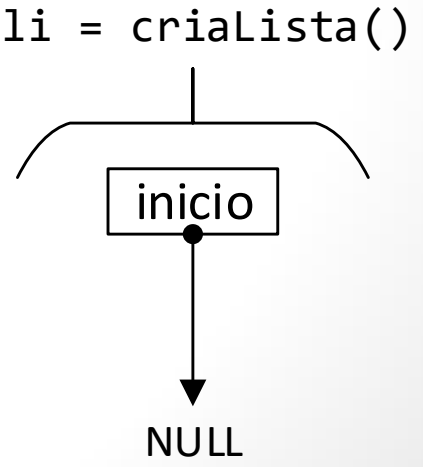
```c
//Arquivo main()
if((li = criaLista()) == NULL){
    abortaPrograma();
}
```

Se não for possível alocar o bloco de memória, o programa será então abortado.

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

Armazenará o endereço do início do bloco alocado, e será devolvido ao `main()`

Se alocação ok, preenche o conteúdo que foi alocado com NULL



Lista Ligada – Criando a Lista

- Este será um modelo básico de uma lista ligada para armazenamento de dados, que utiliza a arquitetura de Tipo Abstrato de Dados. Seu funcionamento só possível mediante a alocação de blocos de memória na *heap*.
- Em caso de impossibilidade de alocação, o programa perde todo o seu objetivo, sendo então abortado.

Em cada função que acessa diretamente a lista, é absolutamente necessário testar se a mesma foi alocada

```c
//Arquivo listaLigada.h
void abortaPrograma();
```

```c
//Arquivo listaLigada.c
void abortaPrograma(){
    printf("ERRO! Lista nao foi alocada, ");
    printf("programa sera encerrado...\n\n\n");
    system("pause");
    exit(1);
}
```

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()`.

```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);
    }
}
```

Recebe o endereço da lista na memória

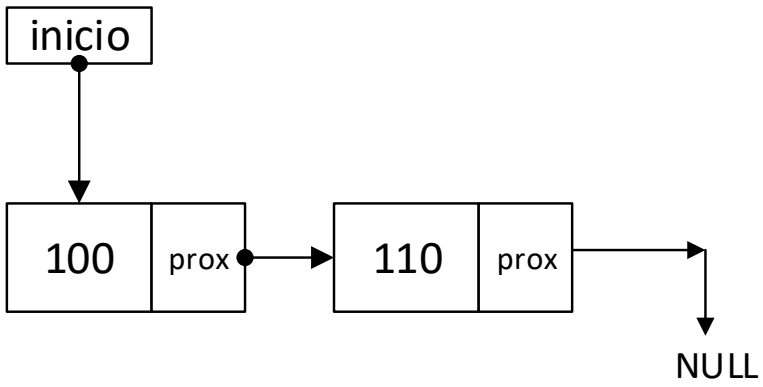
Lista será válida se **`li`** for diferente de NULL

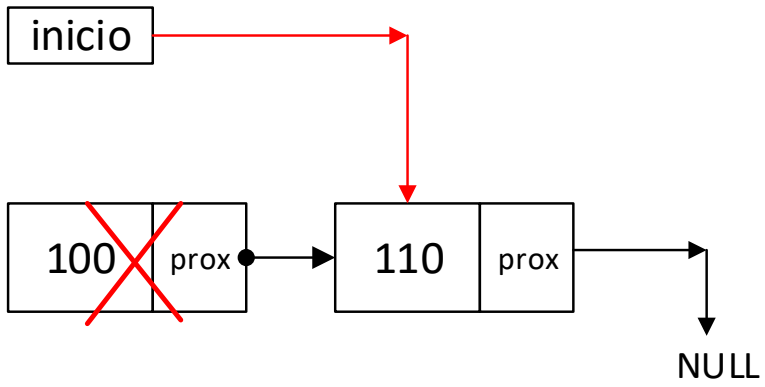
Enquanto o primeiro elemento da lista não for diferente de NULL, a lista não estará vazia. `while`: executa este conjunto de instruções, até que a cabeça da lista aponte para NULL, e assim a lista estará vazia.

Inicio da lista, aponta para próximo elemento da lista

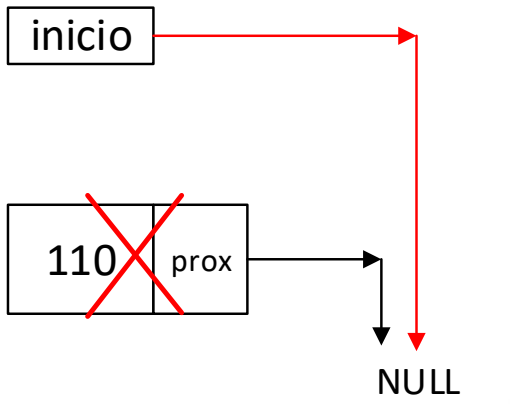
Ao final, libera a cabeça da lista (ponteiro especial), que aponta para o início

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 tamanhoLista(Lista *li);
```

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

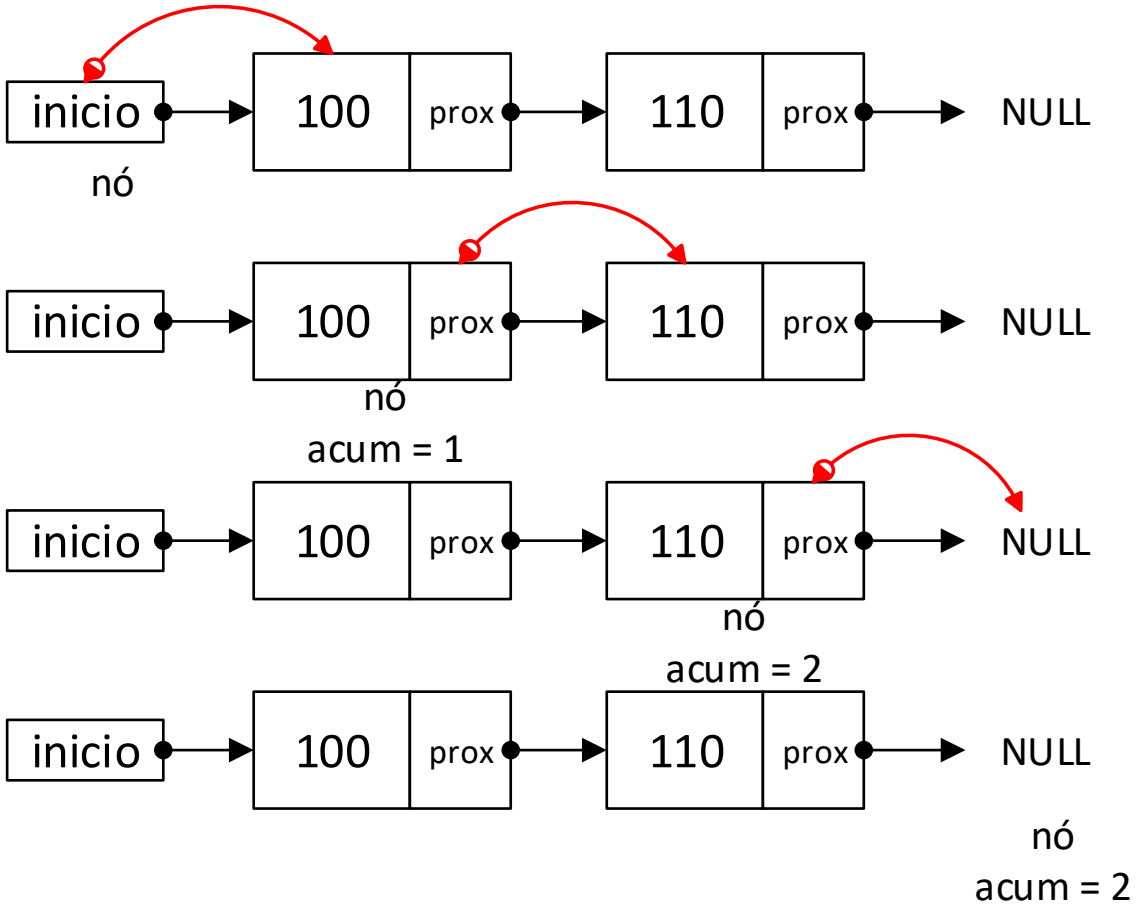
```c
//Arquivo listaLigada.c
int tamanhoLista(Lista *li){
    if(li == NULL){
        abortaPrograma();
    }
    int acum = 0;
    ELEM *no = *li;
    while(no != NULL){
        acum++;
        no = no->prox;
    }
    return acum;
}
```

acum retorna a quantidade de elementos que existem na lista.

nó é um ponteiro auxiliar e recebe o 1º elemento da Lista. nó foi criado para se preservar o inicio da lista, porque se andarmos com a cabeça da lista, perderemos seu endereço de início. Sempre percorremos uma lista com elementos auxiliares, para não perdermos informações.

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

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 para o programa;
- Quando se trabalha com estruturas dinâmicas, não faz sentido verificar se está cheia ou não. Esta função é mantida apenas por questões de compatibilidade com outras estruturas do tipo Lista.

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

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

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

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.");
}
```

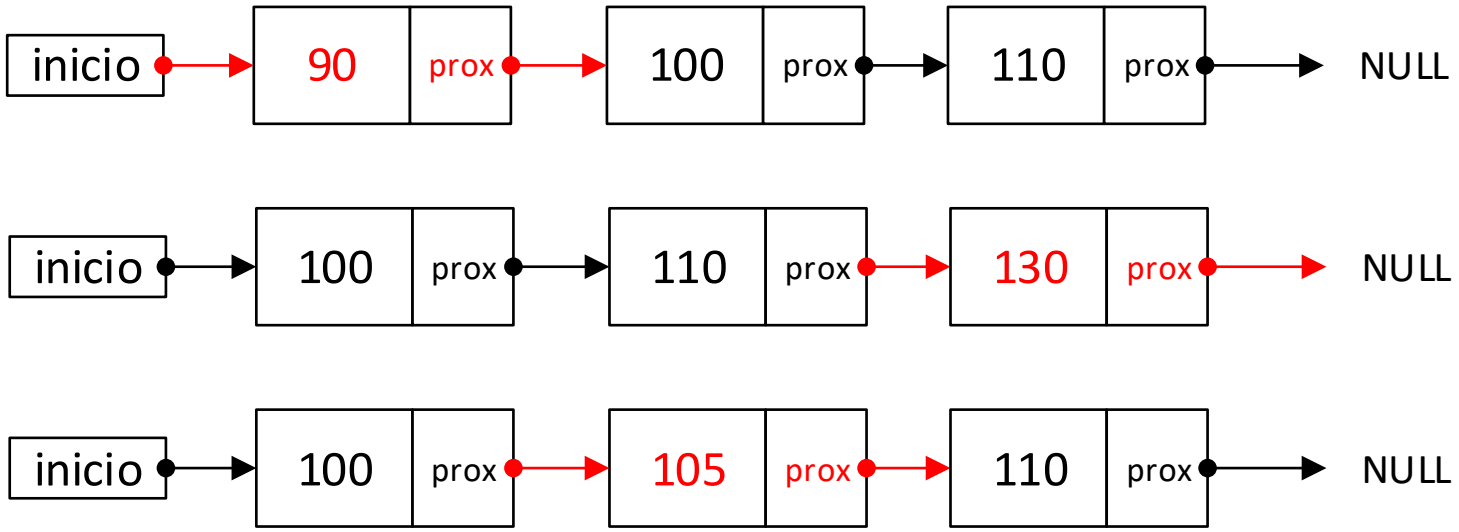
```c
//Arquivo listaLigada.c
int listaVazia(Lista *li){
    if(li == NULL){
        abortaPrograma();
    }
    if(*li == NULL){
        return 1;
    }
    return 0;
}
```

Se a lista não foi alocada, ou seja, li for igual a NULL o programa é então abortado.

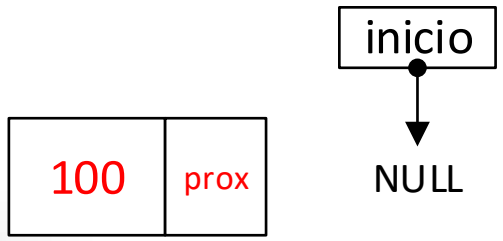
Se no endereço que `*li` aponta, estiver armazenado NULL, ainda não existe nenhum elemento dentro da lista

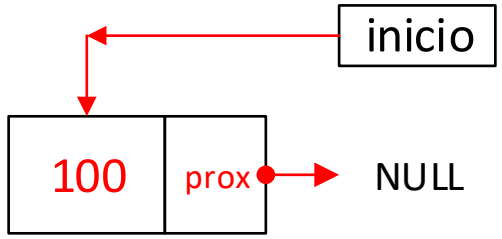
Lista Ligada – Inserção

- 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()`:

x receberá o código de erro como retorno das funções membro do Tipo Abstrato.

matricula será usada na busca por um elemento específico

al1 será usado na inserção no início

```
//Arquivo main()
//código de erro
int x = 0;
// para as buscas
int matricula = 120, posicao = 2;
//para popular a lista e al para consulta
ALUNO al_consulta, al1, al2, al3;

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;
```

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

al_consulta - 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

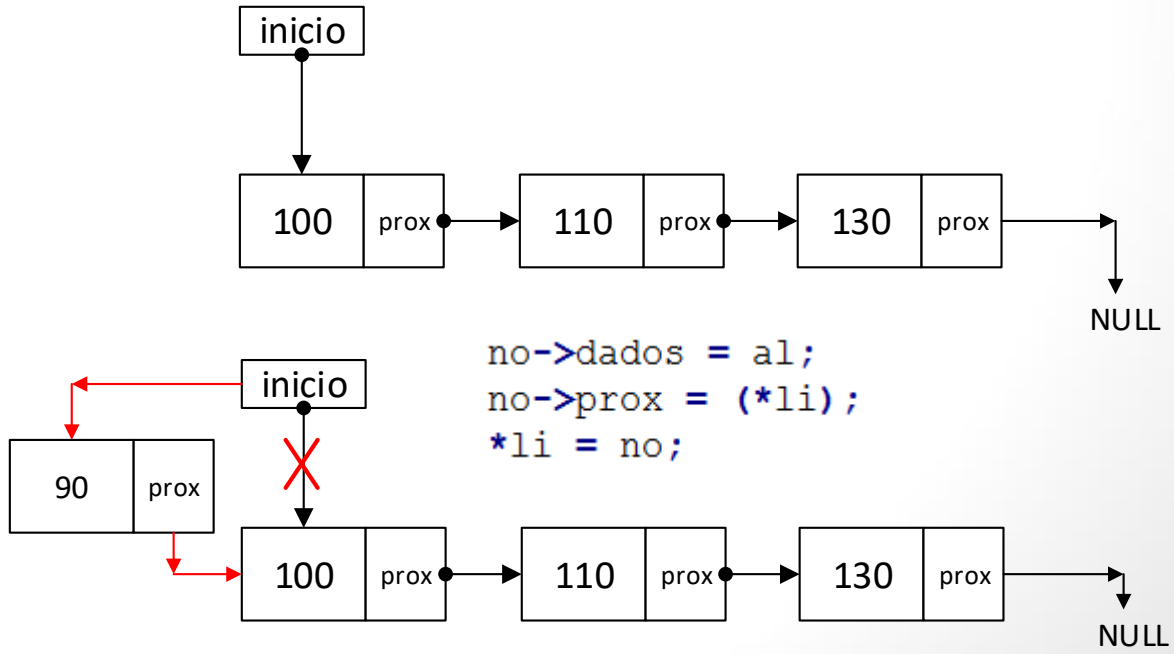
Lista Ligada – Inserção no início da Lista

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

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

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

Resolve inserção no início da lista e em uma lista vazia



Lista Ligada – Inserção no final da Lista

```c
//Arquivo listaLigada.c
int insereFinal(Lista *li, ALUNO al){
    if(li == NULL){
        abortaPrograma();
    }
    ELEM *no = (ELEM*) malloc(sizeof(ELEM));
    if(no == NULL){
        return 0;
    }
    no->dados = al;
    no->prox = NULL;
    if((*li) == NULL){
        *li = no;
    }else{
        ELEM *aux = *li;
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        aux->prox = no;
    }
    return al.matricula;
}
```

Se lista vazia, insere no início

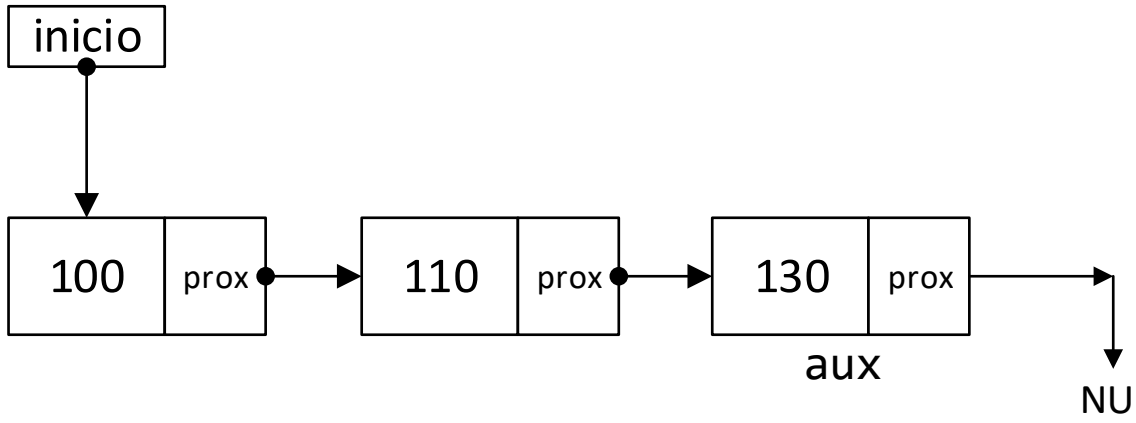
Percorre a lista com um ponteiro auxiliar para preservar a cabeça da lista

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

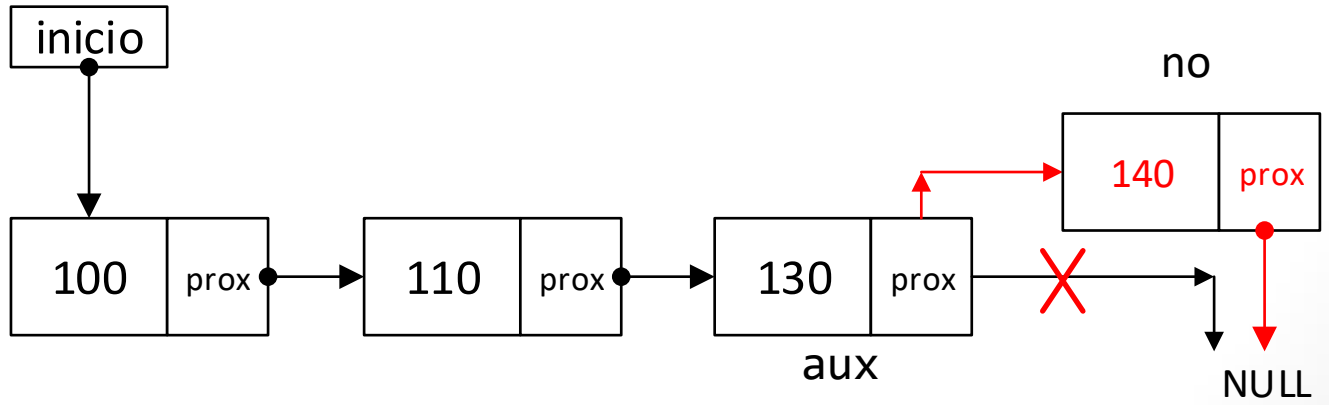
```c
//Arquivo main()
x = insereFinal(li, al2);
if(x){
    printf("\nAluno %d inserido no final com sucesso!", x);
}else{
    printf("\nNao foi possivel inserir no final.");
}
```

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

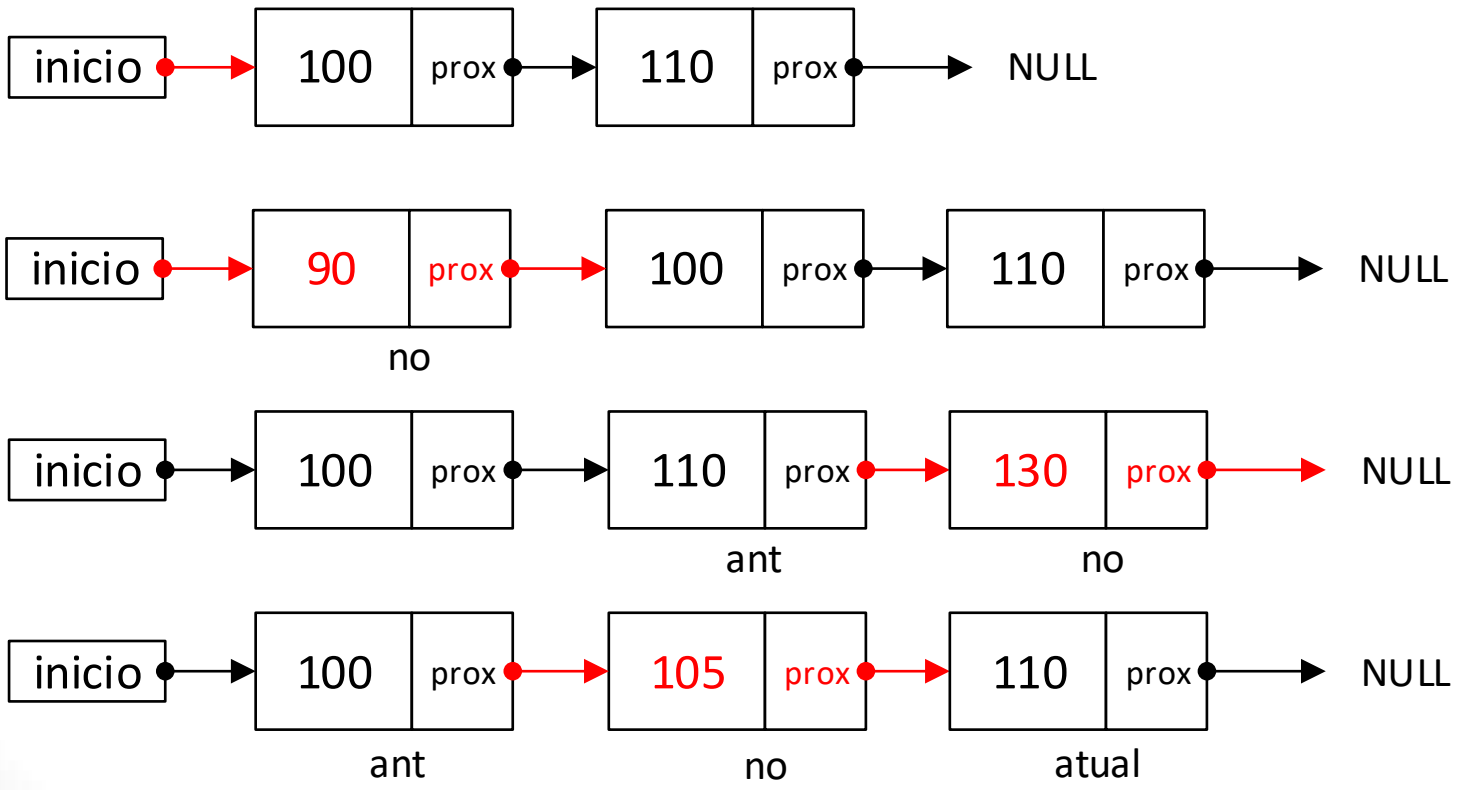
Lista Ligada – Inserção ordenada

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

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

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

Lista Ligada – Inserção ordenada



Busca onde inserir

Inserir no início

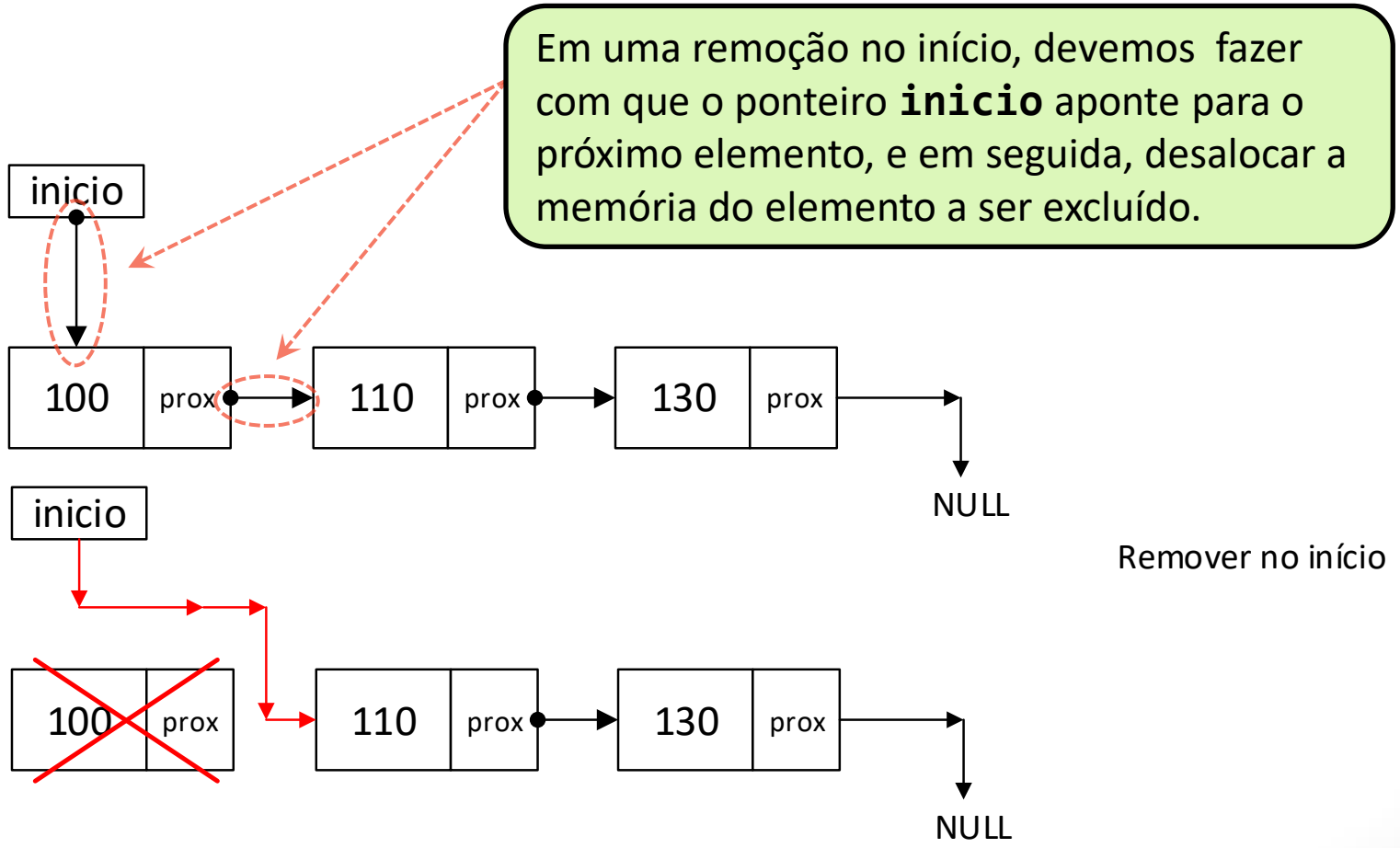
```
no->prox = (*li);
*li = no;
```

Inserir depois de ant

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

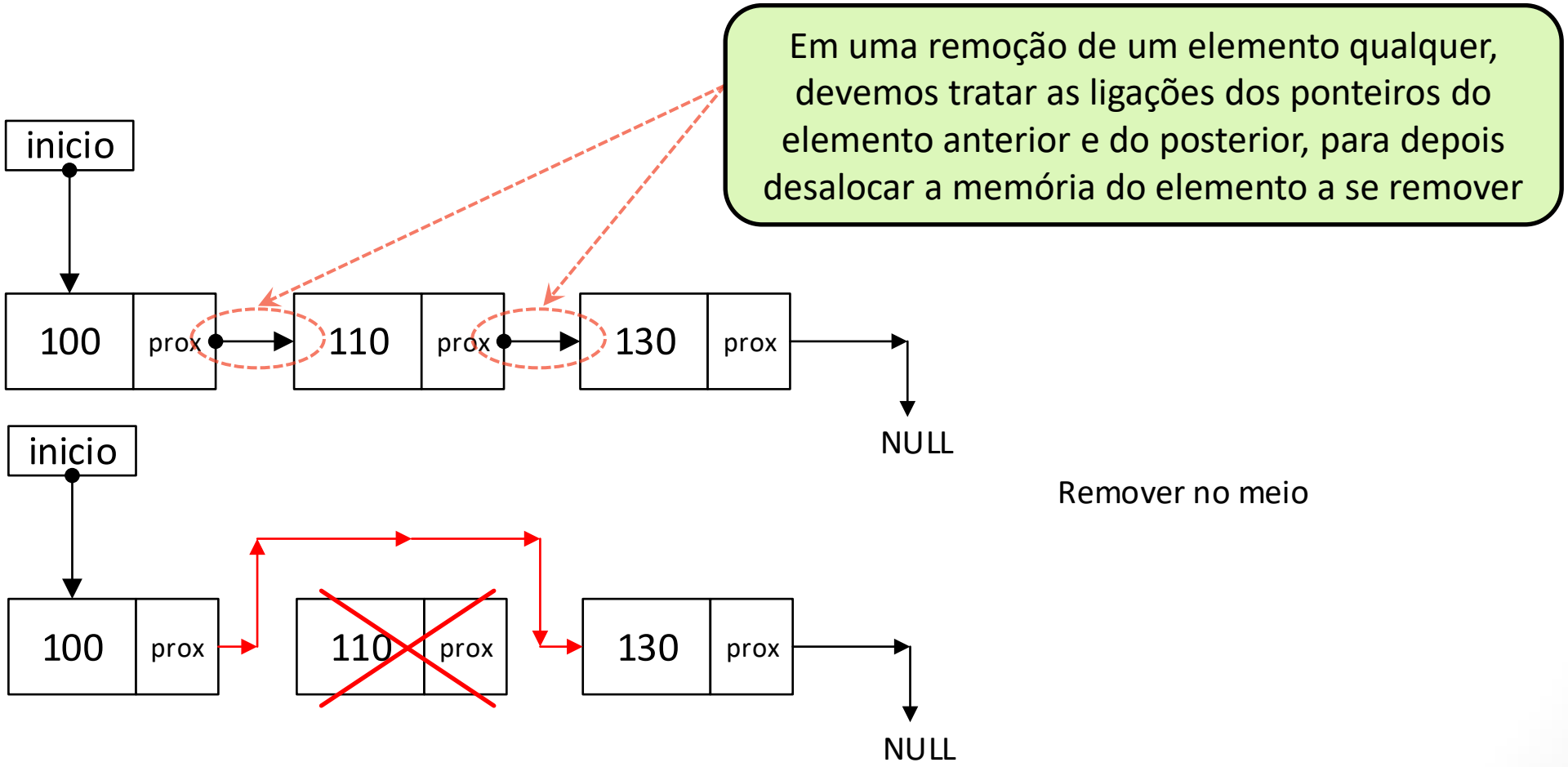
Lista Ligada – Remoção

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



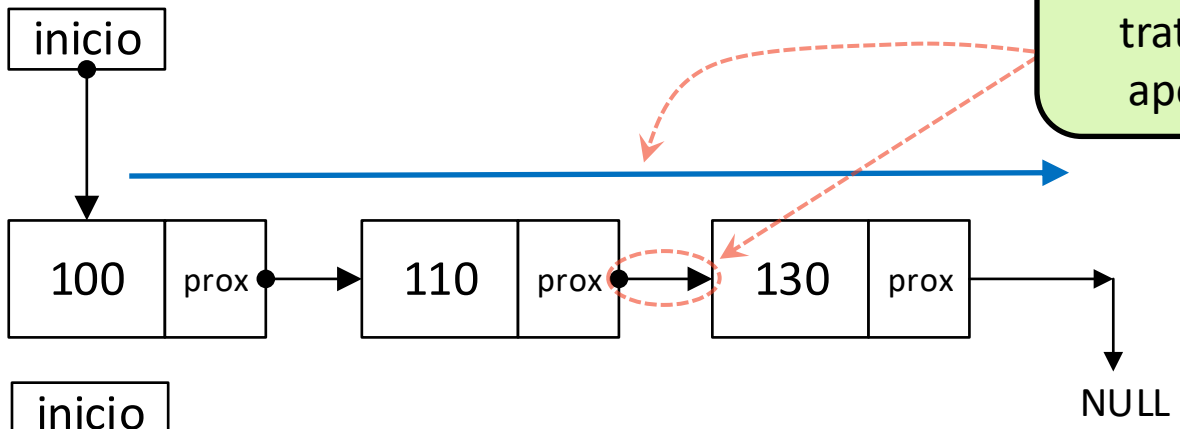
Remover no início

Lista Ligada – Remoção

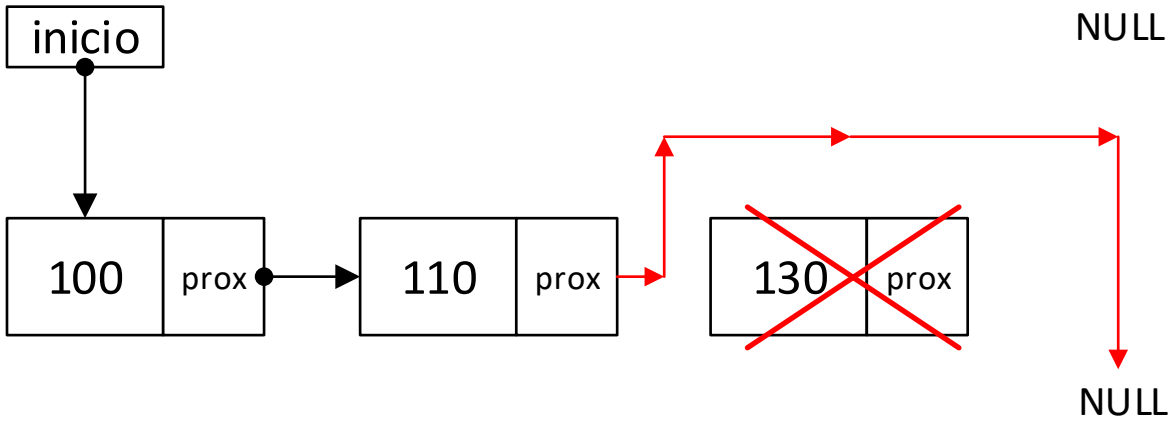


Lista Ligada – Remoção

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

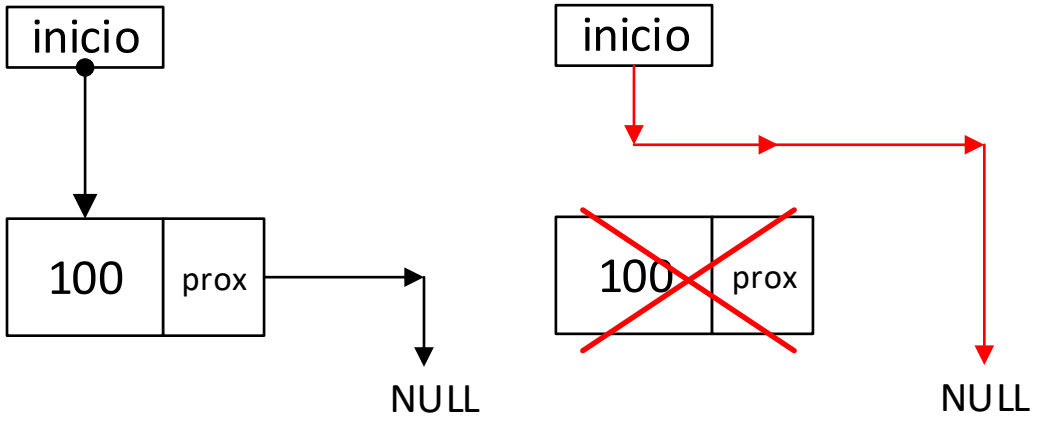


Lista Ligada – Remoção

- 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

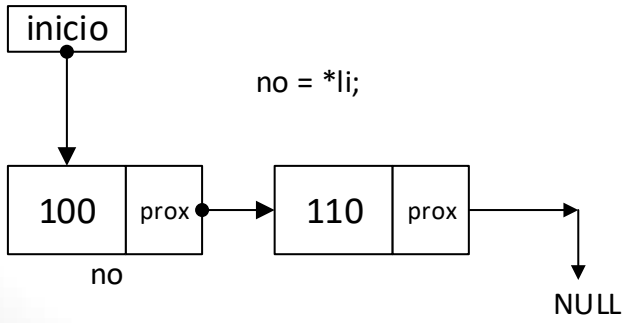
```c
//Arquivo listaLigada.h
int removeInicio(Lista *li);
```

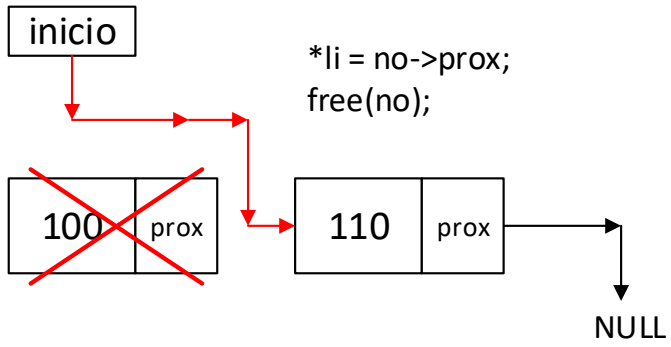
```c
//Arquivo main()
x = removeInicio(li);
if(x){
    printf("\nAluno %d removido do inicio com sucesso!", x);
}else{
    printf("\nNao foi possivel remover do inicio.");
}
```

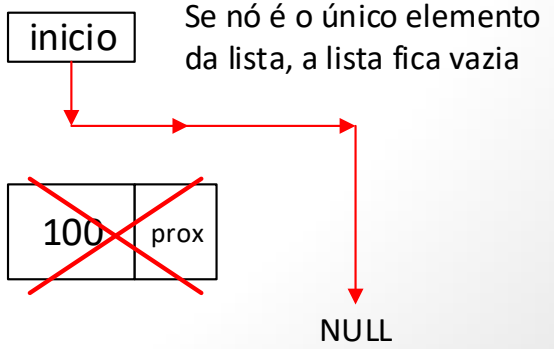
```c
//Arquivo listaLigada.c
int removeInicio(Lista *li){
    int matricula;
    if(li == NULL){
        abortaPrograma();
    }
    if(*li == NULL){
        return 0;
    }
    ELEM *no = *li;
    matricula = no->dados.matricula;
    *li = no->prox;
    free(no);
    return matricula;
}
```

Se lista vazia, não é possível remover

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







Lista Ligada – Remoção no final da Lista

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

Percorre a lista até o final, procurando o último elemento.

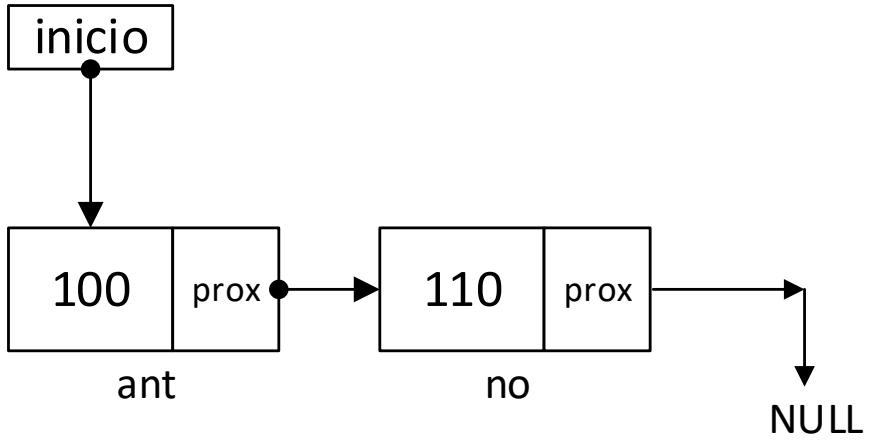
O nó **ant->prox** passa a apontar para onde **no->prox** aponta.

```c
//Arquivo main()
x = removeFinal(li);
if(x){
    printf("\nAluno %d removido do final com sucesso!", x);
}else{
    printf("\nNao foi possivel remover do final.");
}
```

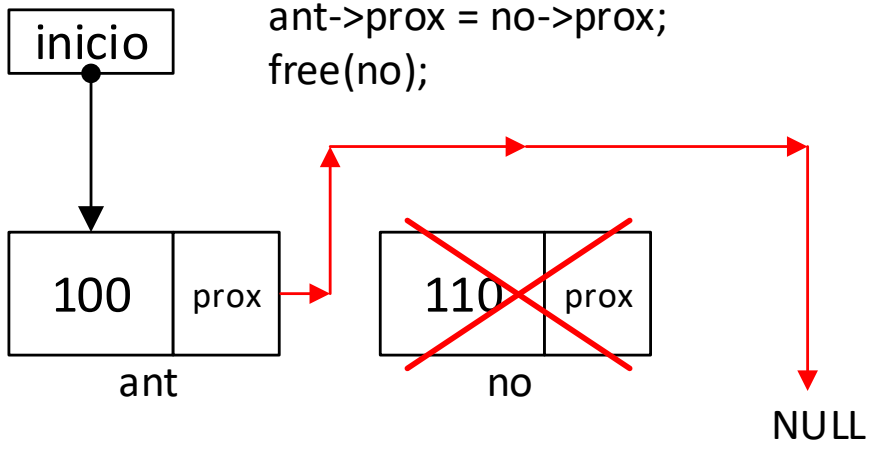
```c
//Arquivo listaLigada.c
int removeFinal(Lista *li){
    int matricula;
    if(li == NULL){
        abortaPrograma();
    }
    if((*li) == NULL){//lista vazia
        return 0;
    }
    ELEM *ant, *no = *li;
    while(no->prox != NULL){
        ant = no;
        no = no->prox;
    }
    if(no ==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

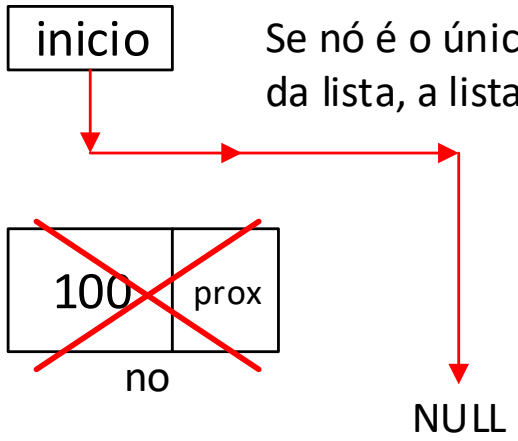
Devolve matricula do aluno removido.

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.h
int removeOrdenado(Lista *li, int mat);
```

```c
//Arquivo main()
x = removeOrdenado(li, matricula);
if(x){
    printf("\nAluno %d removido ordenadamente com sucesso!", x);
}else{
    printf("\nNao foi possivel remover o aluno.");
}
```

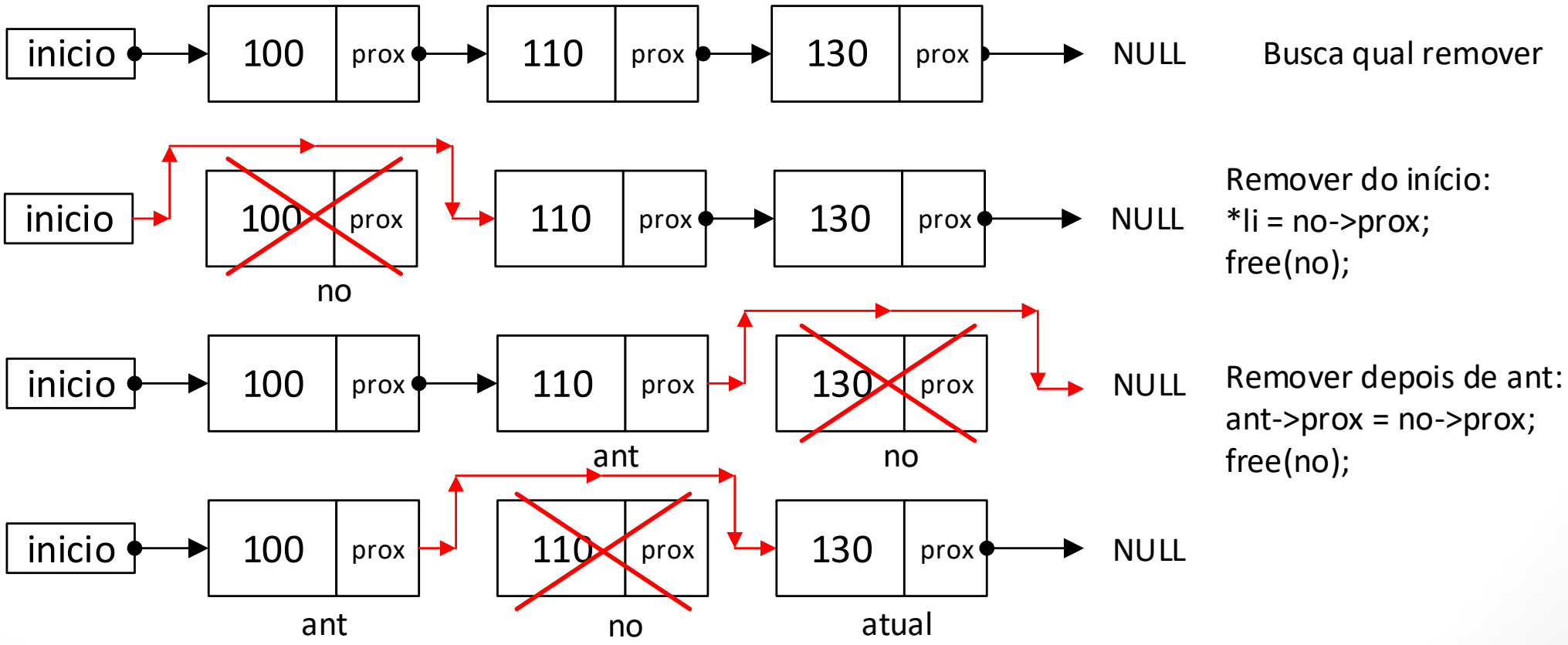
```c
//Arquivo listaLigada.c
int removeOrdenado(Lista *li, int mat){
    int matricula;
    if(li == NULL){
        abortaPrograma();
    }
    ELEM *ant, *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        ant = no;
        no = no->prox;
    }
    if(no == NULL){
        return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

Não é necessário testar se a lista estava vazia para remover, o **`while`** faz esse tratamento, quando testa se no != NULL.

Se o nó está parado na primeira posição, significa que a lista não foi percorrida.
É o caso em que o elemento a ser removido é menor do que todos os outros na lista.

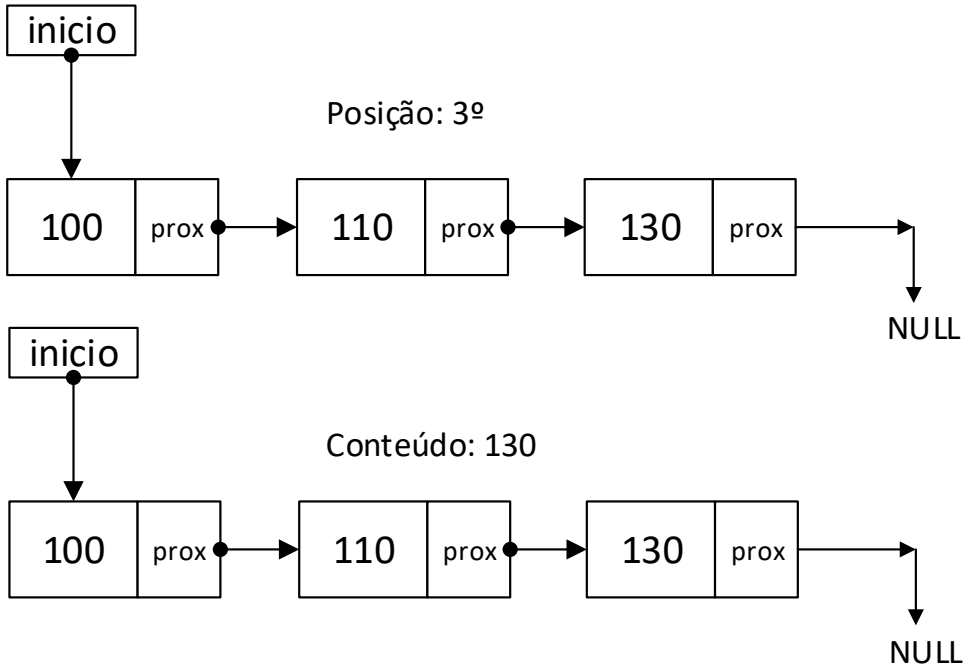
Remove no início, meio e fim.

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 consultaPosicao(Lista *li, int posicao, ALUNO *al);
```

```c
//Arquivo main()
x = consultaPosicao(li, posicao, &al_consulta);
if(x){
    printf("\n\nConsulta na posicao %d:", posicao);
    printf("\nMatricula:  %d", al_consulta.matricula);
    printf("\nNota 1:     %.2f", al_consulta.n1);
    printf("\nNota 2:     %.2f", al_consulta.n2);
    printf("\nNota 3:     %.2f", al_consulta.n3);
}else{
    printf("\nPosicao %d nao existe.", posicao);
}
```

```c
//Arquivo listaLigada.c
int consultaPosicao(Lista *li, int posicao, ALUNO *al){
    if(li == NULL){
        abortaPrograma();
    }
    if(posicao <= 0){
        return;
    }
    ELEM *no = *li;
    int i = 1;
    while (no != NULL && i < posicao){
        no = no->prox;
        i++;
    }
    if(no == NULL){
        return 0;
    }else{
        *al = no->dados;
        return 1;
    }
}
```

Se nó for igual a NULL, o elemento não foi encontrado (posição não existe), ou lista estava vazia.

Mas se nó é diferente de NULL, significa que nó está apontando para o elemento procurado, e então, é só copiar os dados do elemento.

Percorre toda a lista, incrementando `i` para cada elemento pesquisado

Lista Ligada – Consultas por conteúdo

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

```c
//Arquivo main()
x = consultaMatricula(li, matricula, &al_consulta);
if(x){
    printf("\n\nConsulta aluno matriculado:");
    printf("\nMatricula:  %d", al_consulta.matricula);
    printf("\nNota 1:     %.2f", al_consulta.n1);
    printf("\nNota 2:     %.2f", al_consulta.n2);
    printf("\nNota 3:     %.2f", al_consulta.n3);
}else{
    printf("\nMatricula %d nao encontrada.", matricula);
}
```

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

Se ao final da busca, no for igual a NULL, significa que o elemento buscado não existe na lista, ou a mesma está vazia.

Se no diferente de NULL, significa que o elemento buscado foi encontrado, e então, é só copiar seu conteúdo.

Enquanto no for diferente de NULL, e a matricula na lista for diferente da matricula que procuro...

Estrutura de Dados 1

- Execução do programa Lista-Ligada:

```
"C:\Users\angelot\Documents\Aulas\EDA1\Aulas\Aula 1
O tamanho da lista e: 0
Lista nao esta cheia.
Lista esta vazia.
Aluno 110 inserido no inicio com sucesso!
Lista nao esta vazia.
Aluno 130 inserido no final com sucesso!
Aluno 120 inserido ordenadamente com sucesso!

O tamanho da lista e: 3 alunos

Consulta na posicao 2:
Matricula:  120
Nota 1:     6.60
Nota 2:     2.10
Nota 3:     9.20


Posicao 4 nao existe.

Consulta aluno matriculado:
Matricula:  120
Nota 1:     6.60
Nota 2:     2.10
Nota 3:     9.20


Aluno 120 removido ordenadamente com sucesso!
Matricula 120 nao encontrada.



Consulta na posicao 2:
Matricula:  130
Nota 1:     9.20
Nota 2:     3.50
Nota 3:     8.10


Posicao 3 nao existe.
Aluno 130 removido do final com sucesso!
Aluno 110 removido do inicio com sucesso!



Pressione qualquer tecla para continuar. . .
```

Atividade

- Monte o programa com todas as funções apresentadas pertencente à lista ligada.
- Implemente nesta lista a mesma função `coletadados()`, do exercício lista estática, e descarte as inserções via código (as 3 inserções de teste).
- 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.