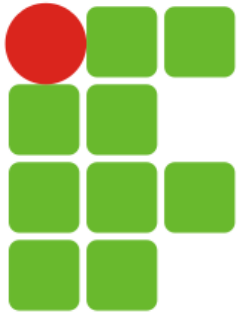


Estrutura de Dados 1

Aula 12 – Lista Ligada ou Sequencial Dinâmica

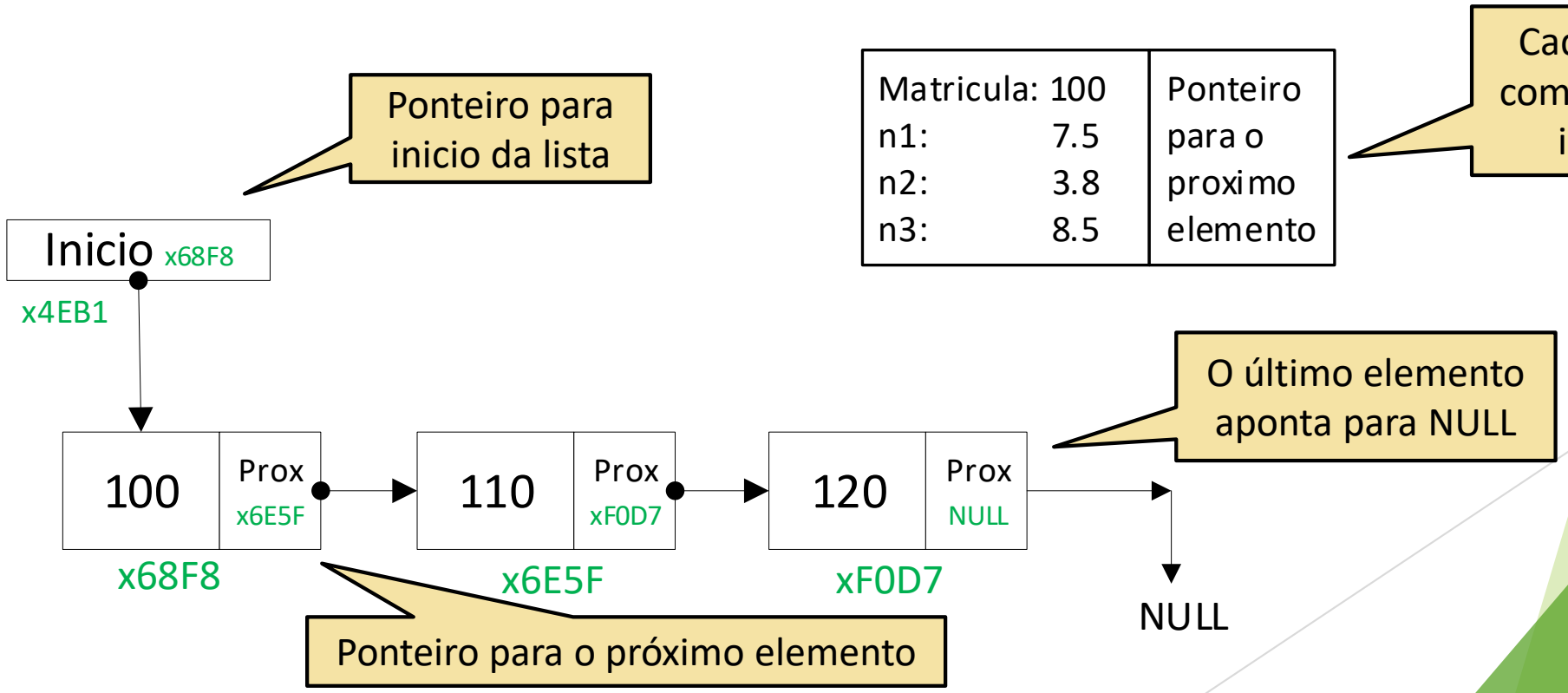
Antonio Angelo de Souza Tartaglia

angelot@ifsp.edu.br

Estrutura de Dados 1

Lista Ligada

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



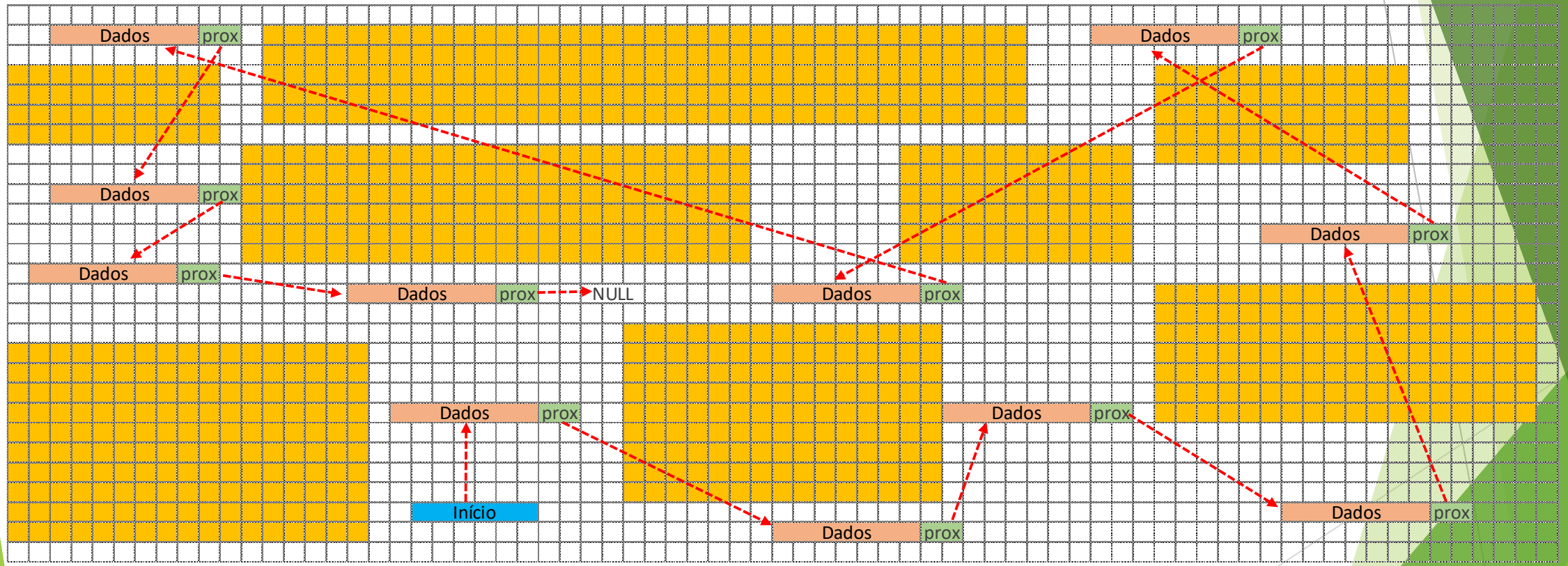
Estrutura de Dados 1

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

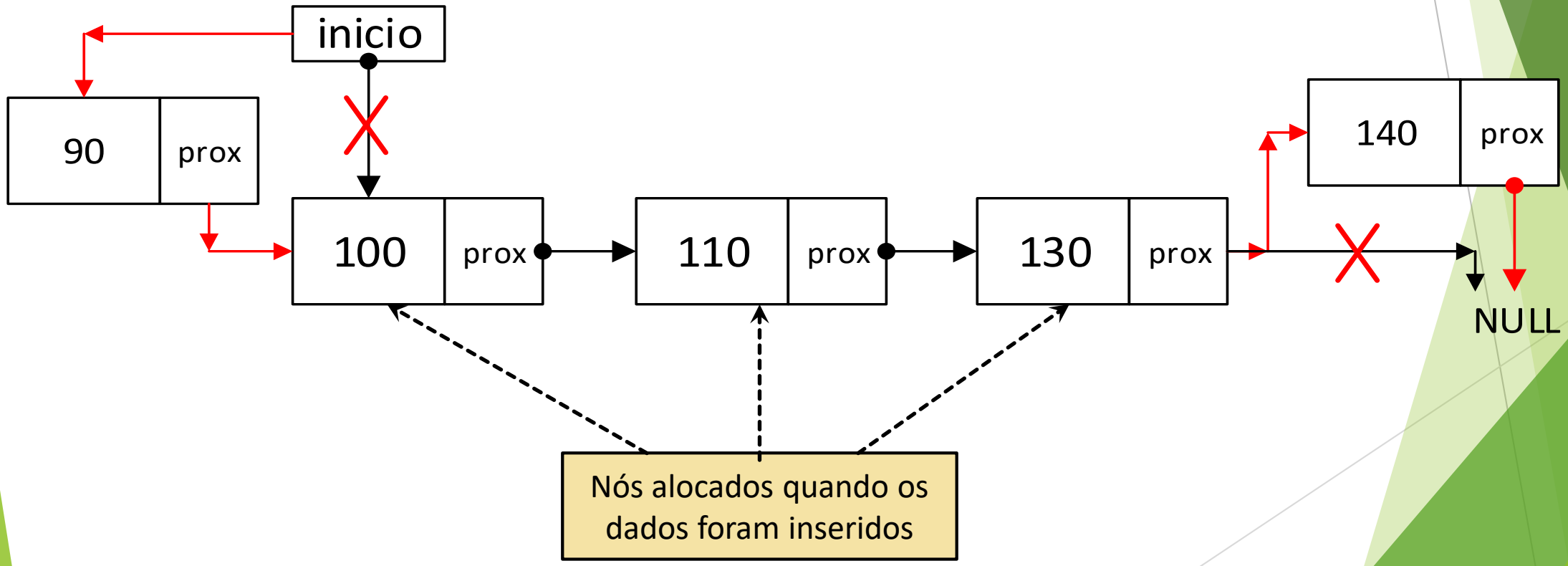
Estrutura de Dados 1

Lista Ligada – Exemplo de utilização de memória



Estrutura de Dados 1

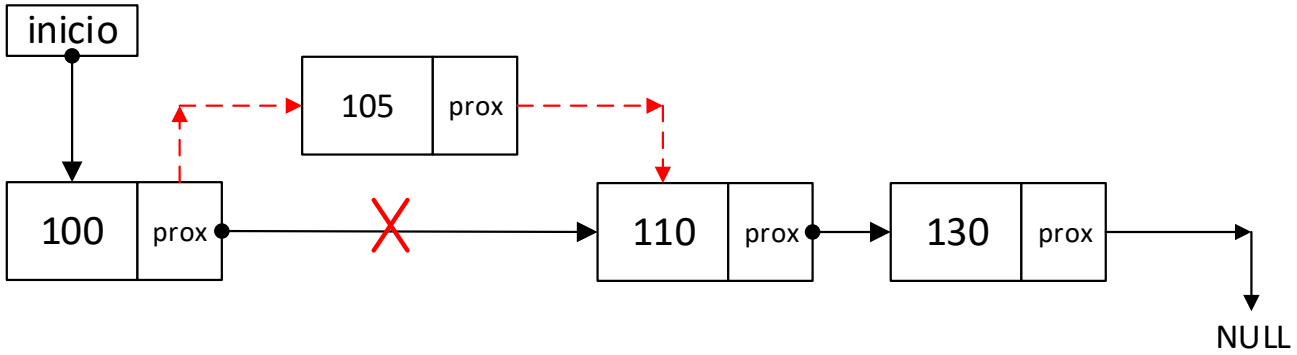
Lista Ligada

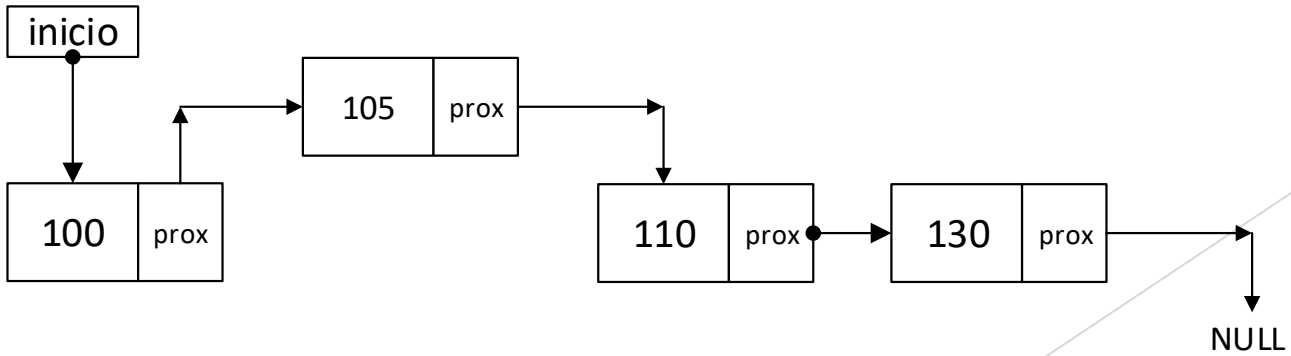


Estrutura de Dados 1

Lista Ligada

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

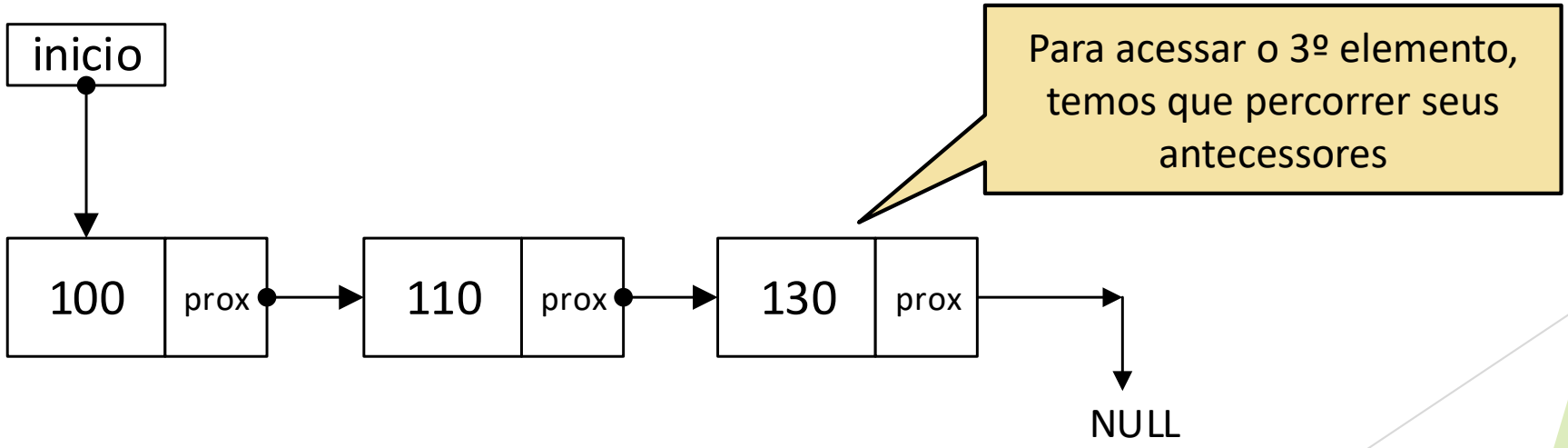




Estrutura de Dados 1

Lista Ligada

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



Estrutura de Dados 1

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.

Estrutura de Dados 1

Lista Ligada - Implementação

- Implementação da Lista Ligada (ou Dinâmica Encadeada):
- Como a lista Estática, também neste tipo de lista implementaremos um TAD composto de 3 arquivos:
  - main.c;
  - 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.

Estrutura de Dados 1

Lista Ligada - Implementação

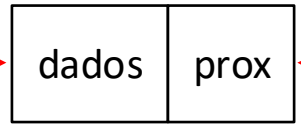
```c
//Arquivo listaLigada.h
typedef struct aluno{
    int matricula;
    float n1,n2,n3;
}ALUNO;

typedef struct elemento* Lista;
```

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

struct elemento{
    ALUNO dados;
    struct elemento *prox;
};
typedef struct elemento ELEM;
```

```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
```



Estrutura de Dados 1

Lista Ligada – Criando a Lista

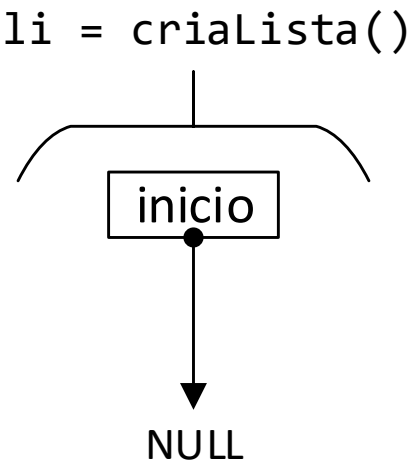
```c
//Arquivo listaLigada.h
Lista *criaLista();
```

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

Guarda o primeiro nó

Se alocação ok, preenche o conteúdo com NULL

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



Estrutura de Dados 1

Lista Ligada – Liberando a Lista

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

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

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

Passa o endereço da inteira que se quer liberar

Lista é válida se for diferente de NULL

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

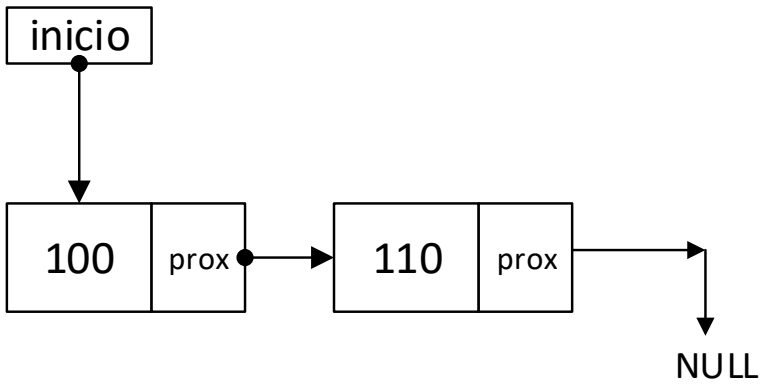
Ao final, libera a cabeça da lista (ponteiro especial), que aponta para o início

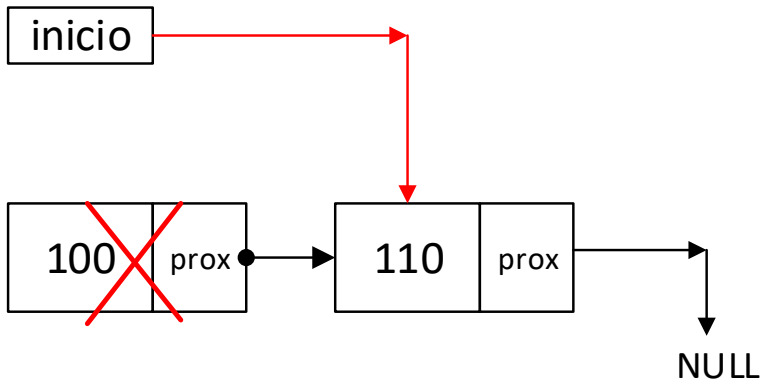
Inicio da lista, aponta para próximo elemento da lista

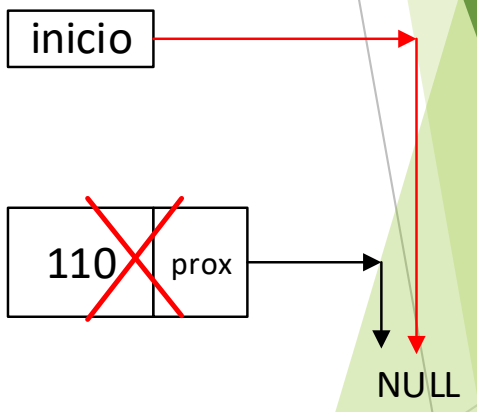
Esta função deve ser a última a ser chamada pelo main().

Estrutura de Dados 1

Lista Ligada – Liberando a Lista








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

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

Estrutura de Dados 1

Lista Ligada - Tamanho

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

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

```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;
}
```

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

Acumulador vai retornar a quantidade de elementos no

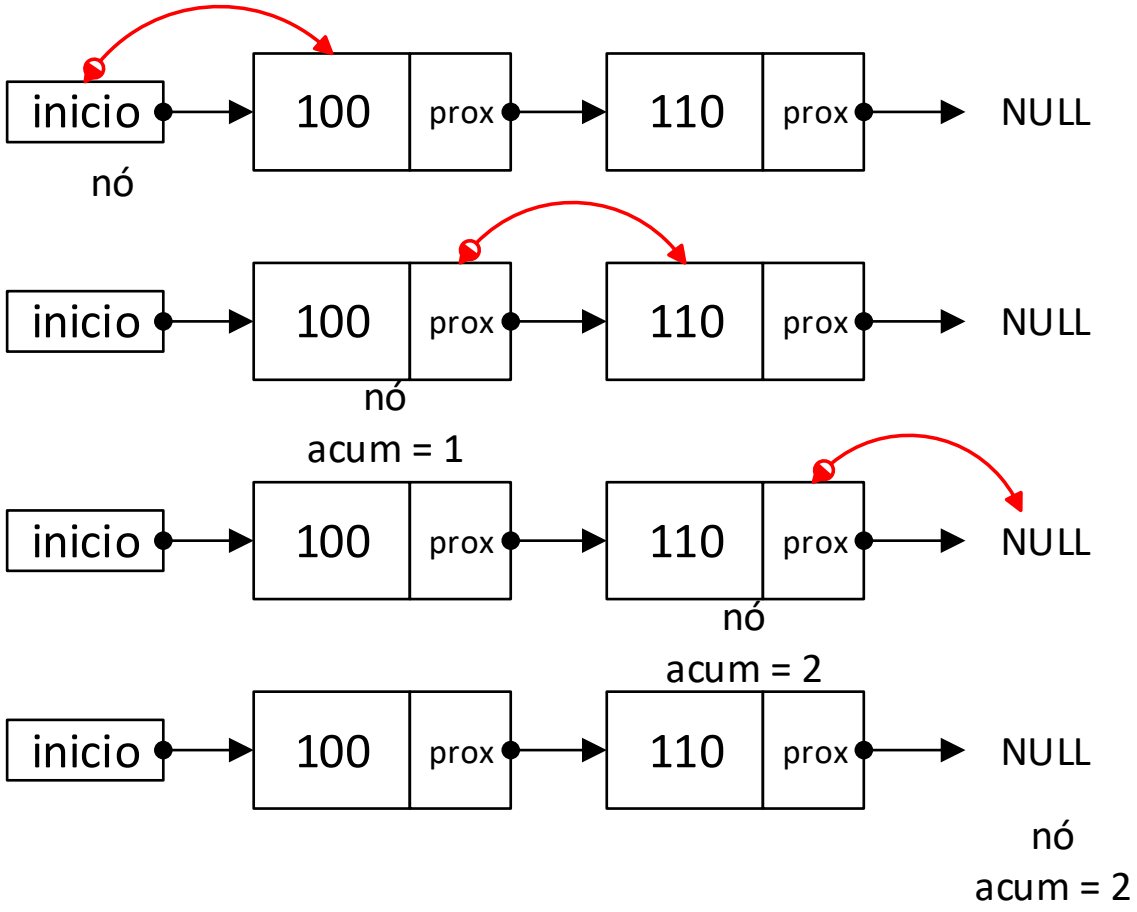
nó auxiliar que recebe 1º elemento da lista

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

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

Estrutura de Dados 1

Lista Ligada – Informações básicas




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

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

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

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

Estrutura de Dados 1

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.

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

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

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

Estrutura de Dados 1

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;
}
```

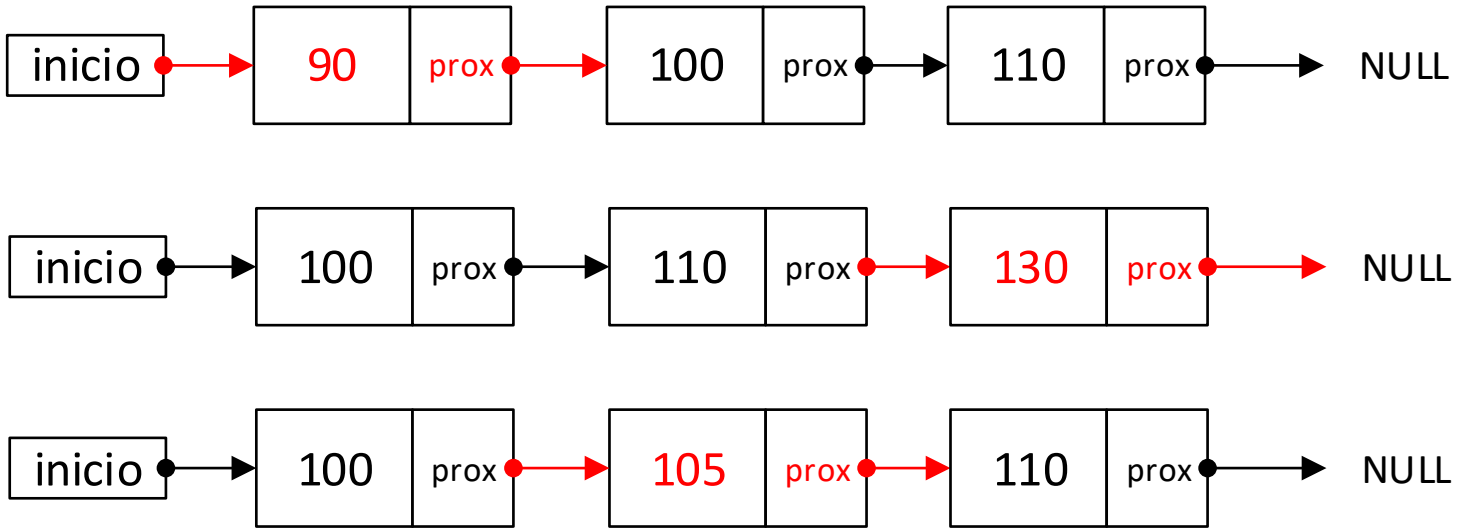
Se a lista for nula, ou seja não existir, informa que ela está vazia

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

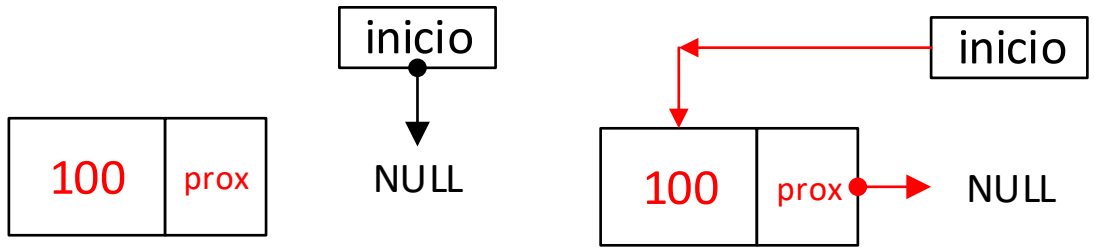
Estrutura de Dados 1

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():

```
int x, mat = 110, posicao = 1;

struct aluno al, al2, al3, dados_aluno;

al.matricula = 100;
al.n1 = 5.3;
al.n2 = 6.9;
al.n3 = 7.4;

al2.matricula = 120;
al2.n1 = 4;
al2.n2 = 2.9;
al2.n3 = 8.4;

al3.matricula = 110;
al3.n1 = 1.3;
al3.n2 = 2.9;
al3.n3 = 3.4;
```

Receberá o código de erro

Será usado na busca por um elemento específico

Será usado na busca por um elemento em uma determinada posição

Estrutura aluno que será utilizada para retorno de informações das buscas na lista

Será usado na inserção no início

Será usado na inserção no final

Será usado na inserção ordenada

Estrutura de Dados 1

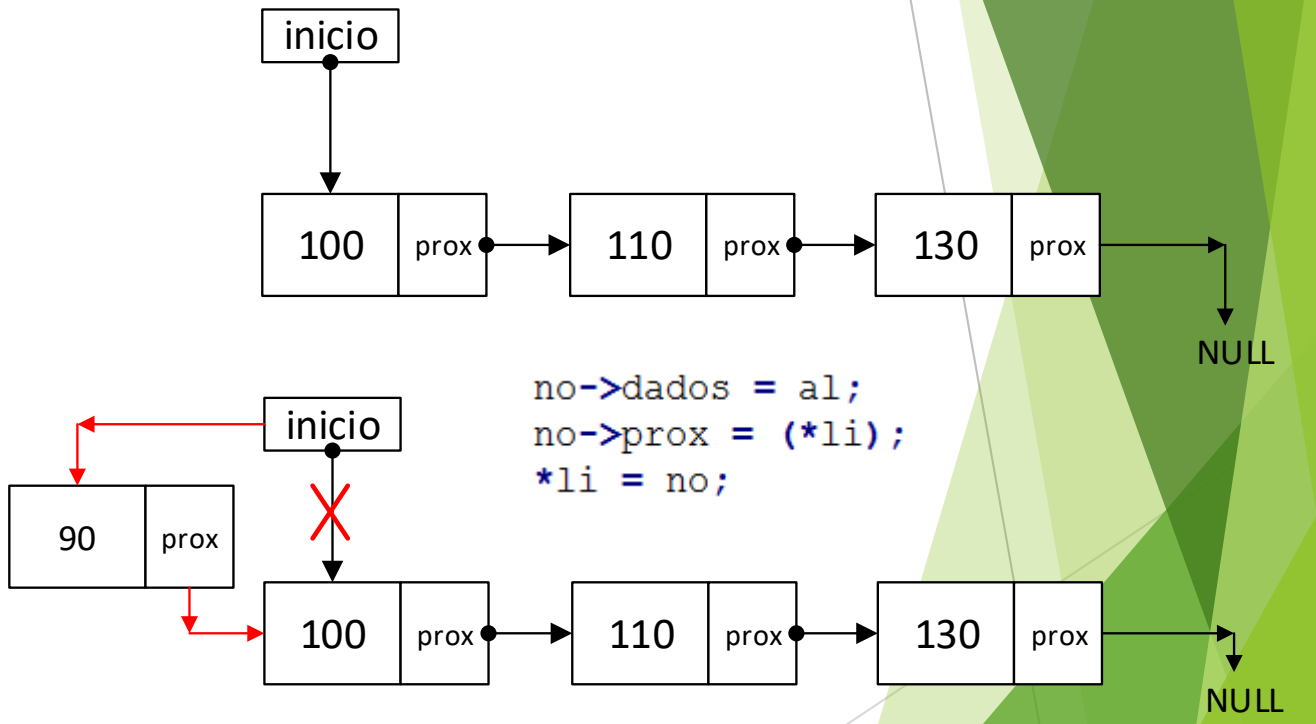
Lista Ligada – Inserção no início da Lista

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

```
//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 lista vazia

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



Estrutura de Dados 1

Lista Ligada – Inserção no final da Lista

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

```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;
}
```

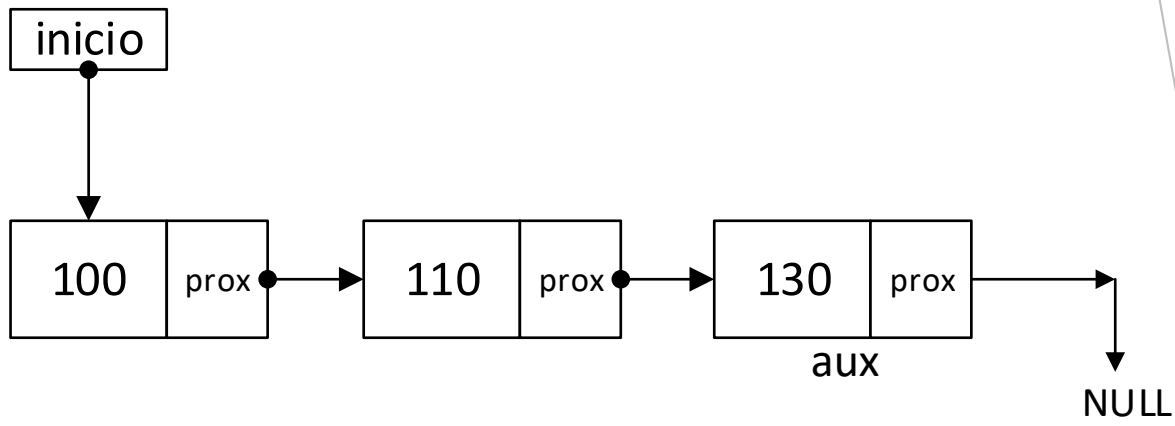
```c
//Arquivo main()
x = insere_final_lista(li, al2);
if(x){
    printf("\nInserido no final com sucesso!");
}else{
    printf("\nNao foi possivel inserir no final.");
}
```

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

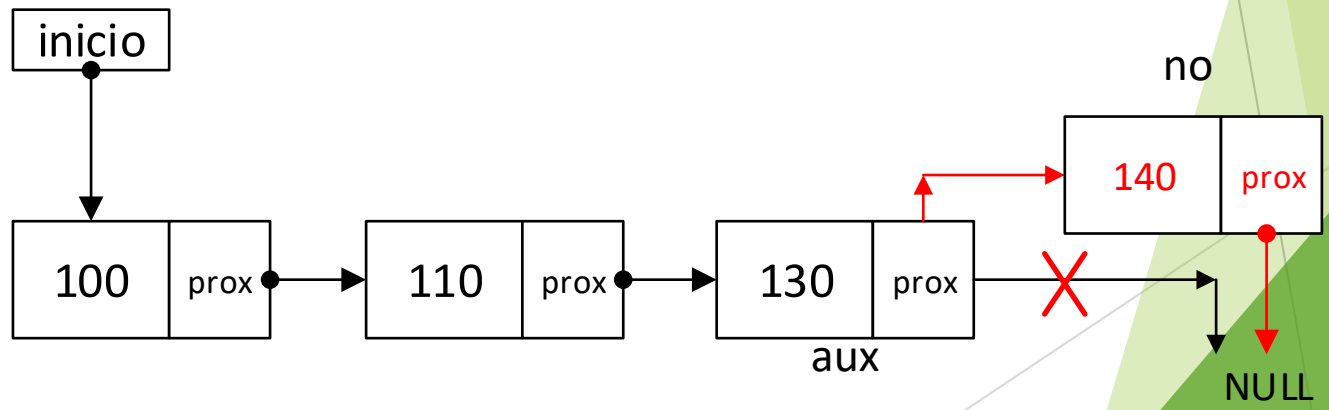
Estrutura de Dados 1

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

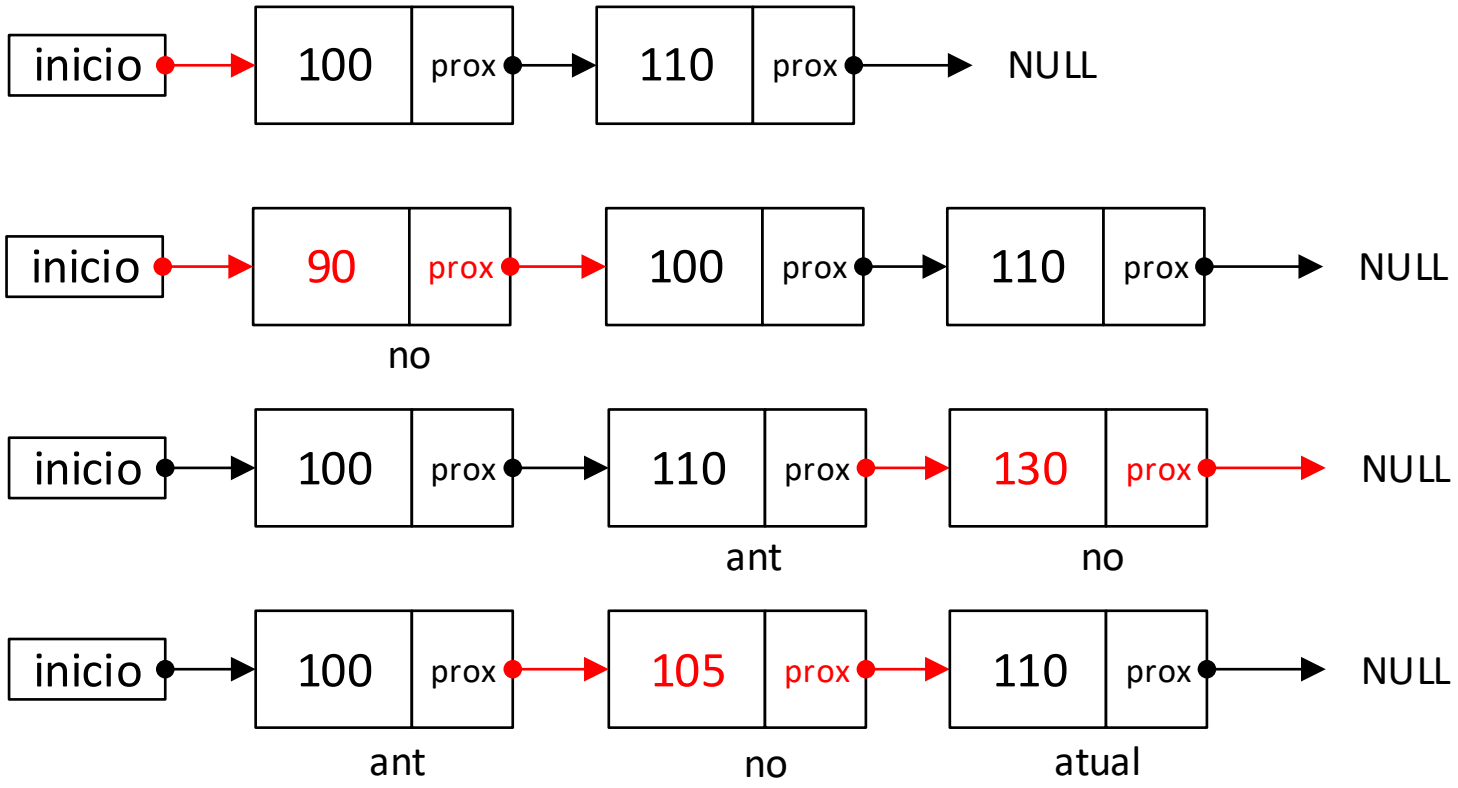
```c
//Arquivo listaLigada.h
int insere_lista_ordenada(Lista *li, ALUNO al);
```

```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;
    }
}
```

Estrutura de Dados 1

Lista Ligada – Inserção ordenada



Busca onde inserir

Inserir no início

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

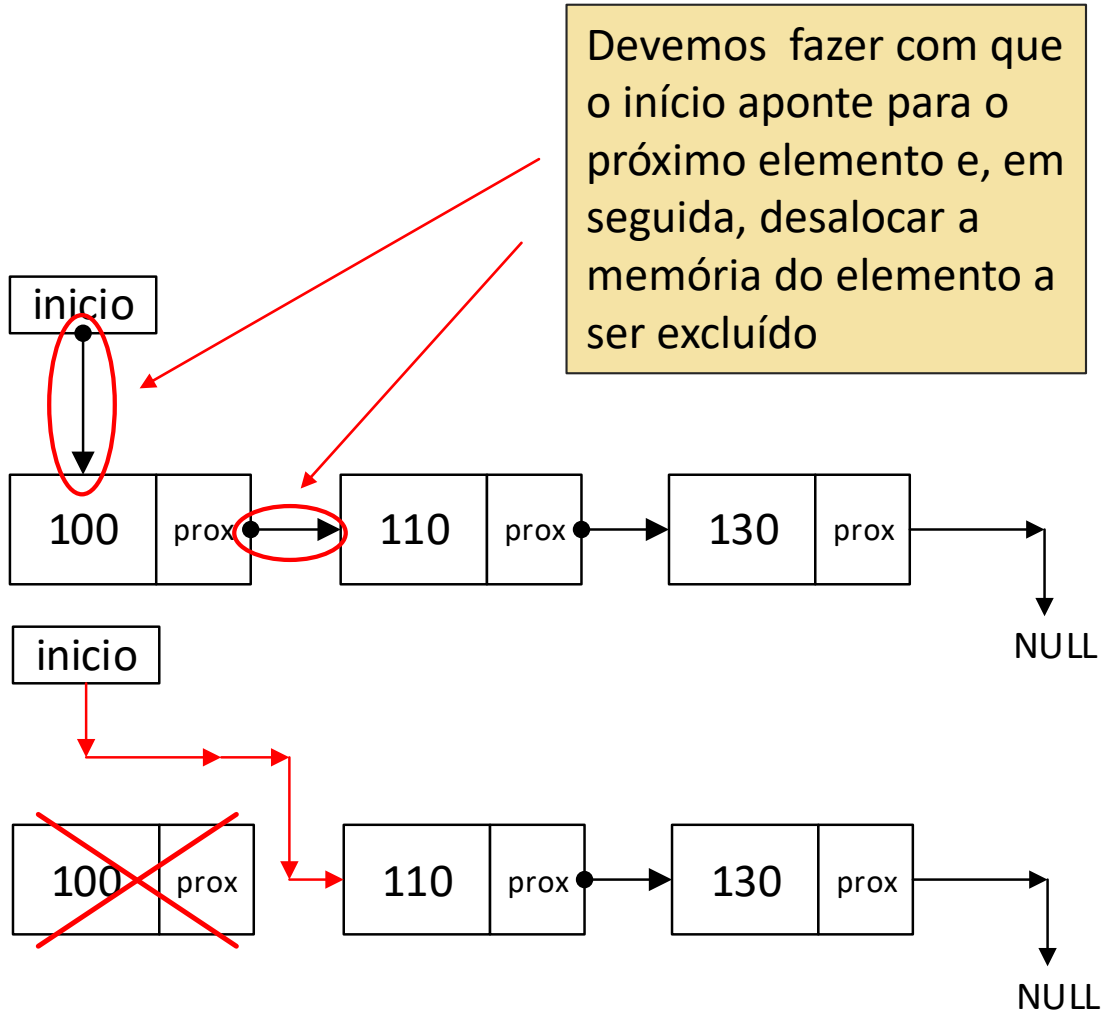
Inserir depois de ant

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

Estrutura de Dados 1

Lista Ligada – Remoção

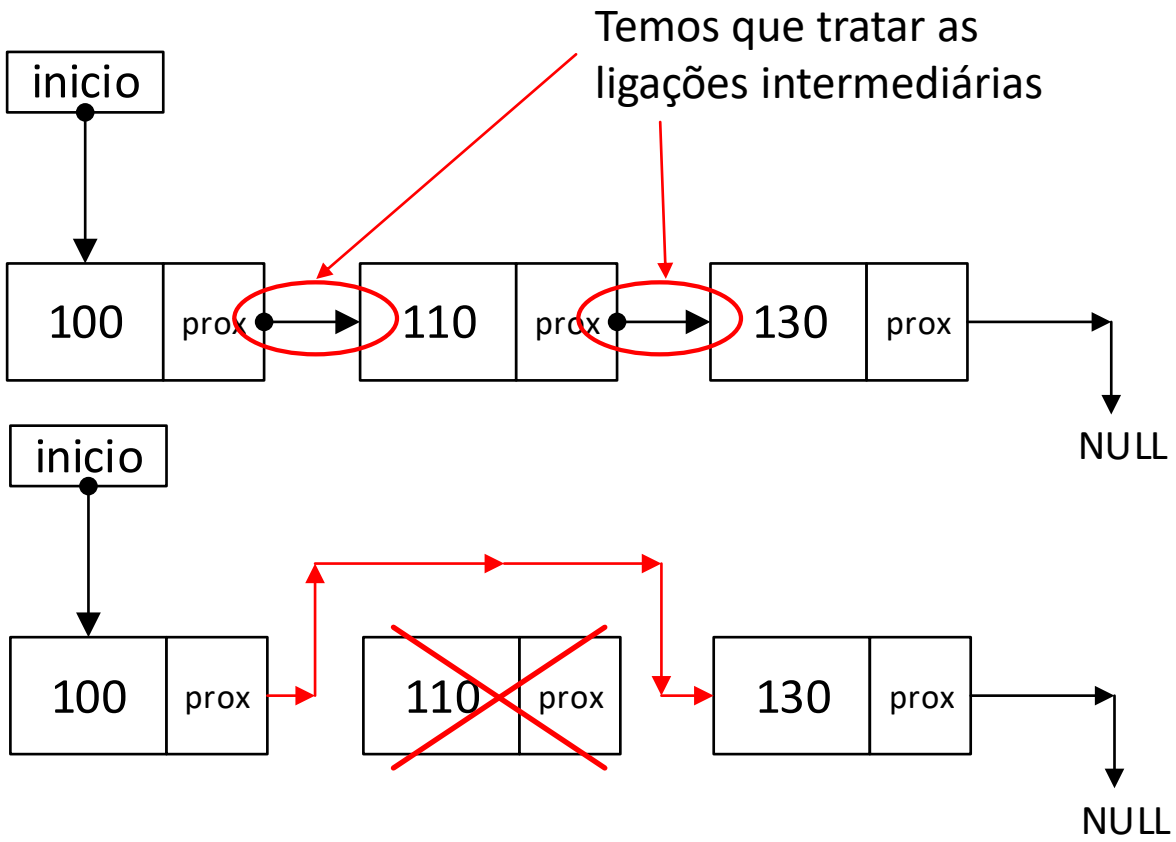
- Existem 3 tipos de remoção:
  - Início
  - Meio
  - Final



Remover no início

Estrutura de Dados 1

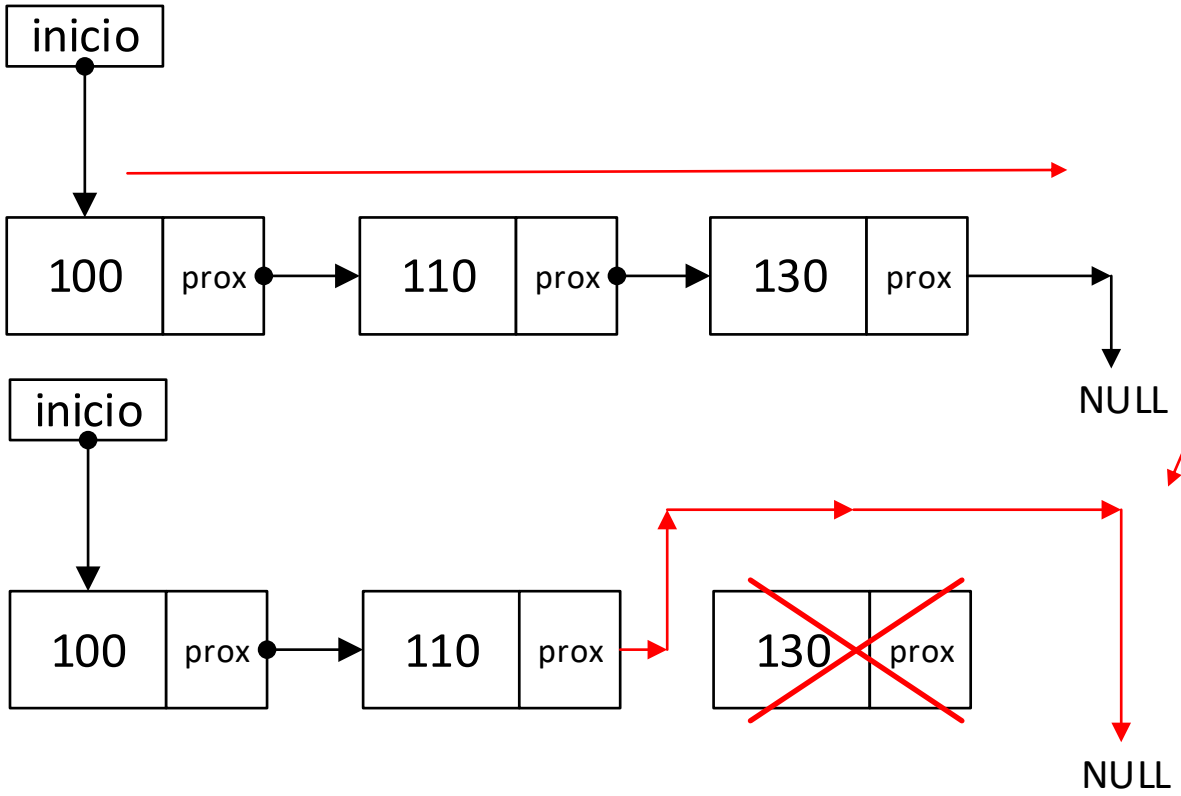
Lista Ligada – Remoção



Remover no meio

Estrutura de Dados 1

Lista Ligada – Remoção



É necessário percorrer toda a lista até chegar ao último elemento, fazer com que o elemento anterior aponte para NULL e então remover o último.

Remover no final

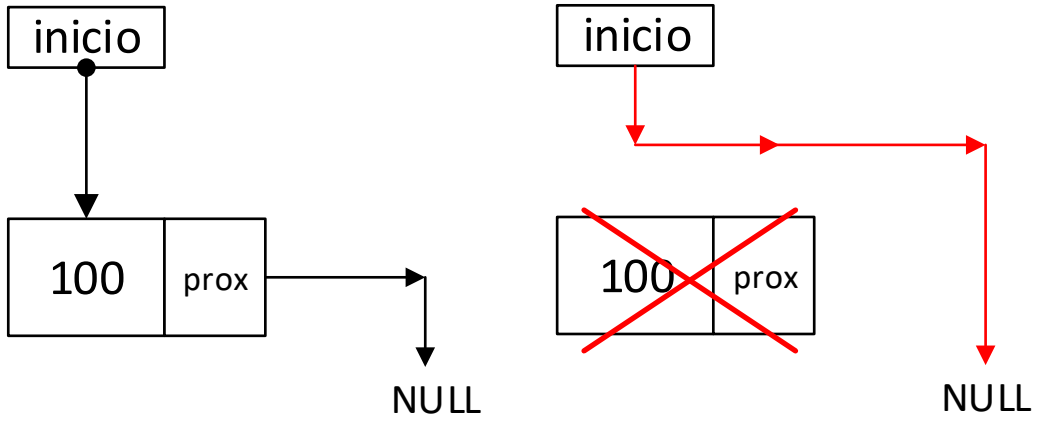
Estrutura de Dados 1

Lista Ligada – Remoção

- Os 3 tipos de remoção trabalham juntos. A remoção sempre remove 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.



Estrutura de Dados 1

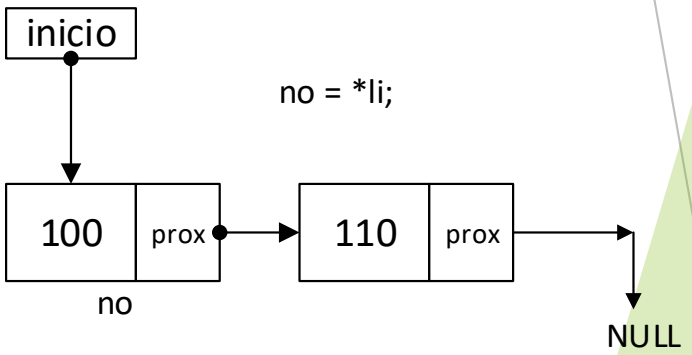
Lista Ligada – Remoção no início da Lista

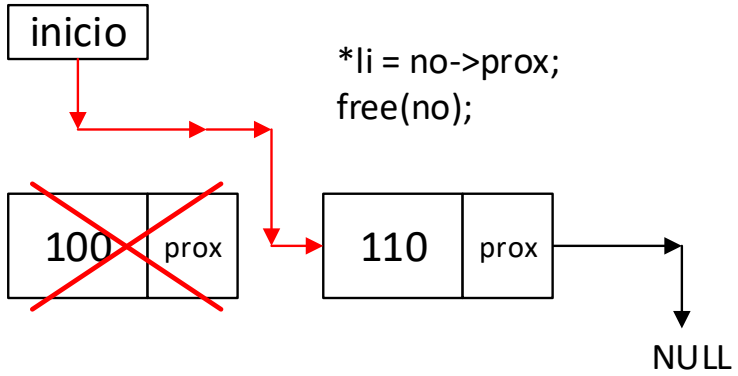
```c
//Arquivo listaLigada.h
int remove_inicio_lista(Lista *li);
```

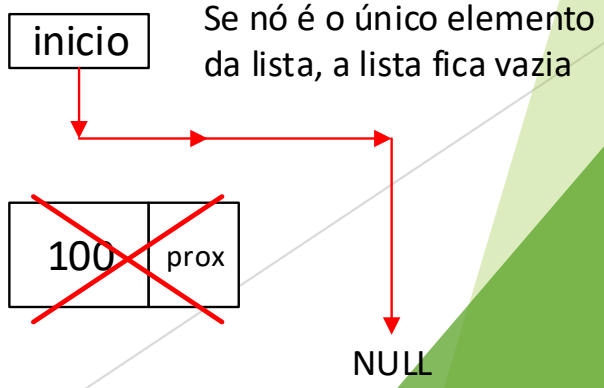
```c
//Arquivo listaLigada.c
int remove_inicio_lista(Lista *li){
    if(li == NULL){ //verifica se a lista existe
        return 0;
    }
    if(*li == NULL){//verifica se a lista está vazia
        return 0;
    }
    ELEM *no = *li;
    *li = no->prox;
    free(no);
    return 1;
}
```

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

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







Estrutura de Dados 1

Lista Ligada – Remoção no final da Lista

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

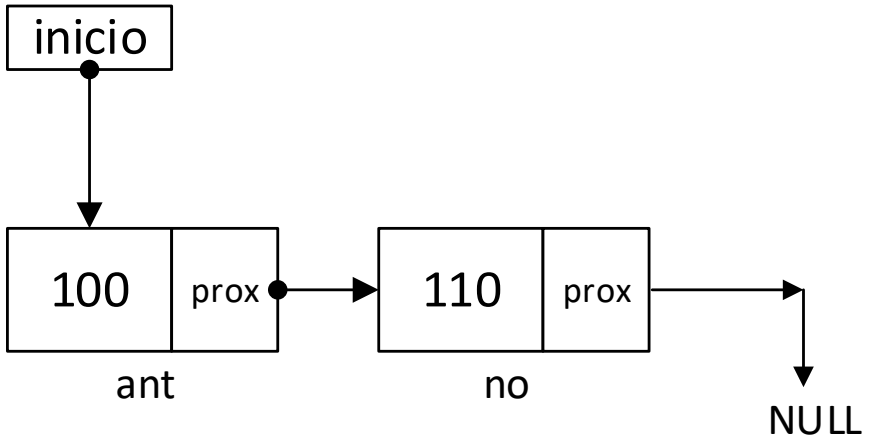
```c
//Arquivo listaLigada.c
int remove_final_lista(Lista *li){
    if(li == NULL){
        return 0;
    }
    if((*li) == NULL){//lista vazia
        return 0;
    }
    ELEM *ant, *no = *li;
    while(no->prox != NULL){
        ant = no;
        no = no->prox;
    }
    if(no == (*li)){//remove o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

Percorre a lista até o final.

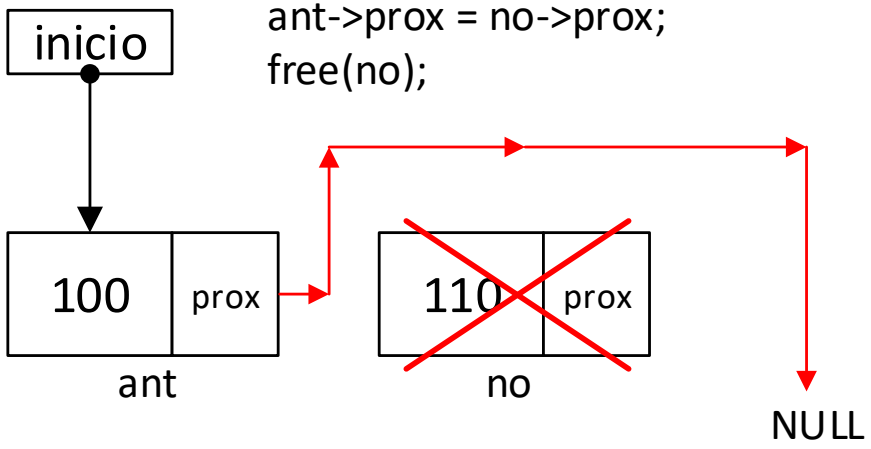
```c
//Arquivo main()
x = remove_final_lista(li);
if(x){
    printf("\nRemovido do final com sucesso!");
}else{
    printf("\nNao foi possivel remover do final.");
}
```

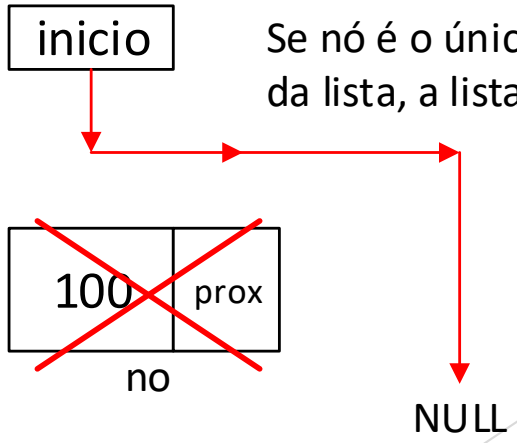
Estrutura de Dados 1

Lista Ligada – Remoção no final da Lista




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





Estrutura de Dados 1

Lista Ligada – Remoção de qualquer elemento

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

//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;
    }
    if(no == NULL){//a lista estava vazia ou percorreu-a toda e não encontrou elemento
        return 0;
    }
    if(no == *li){//remover o primeiro?
        *li = no->prox;
    }else{
        ant->prox = no->prox;//ant vai apontar para elemento seguinte ao nó
    }
    free(no);
    return 1;
}
```

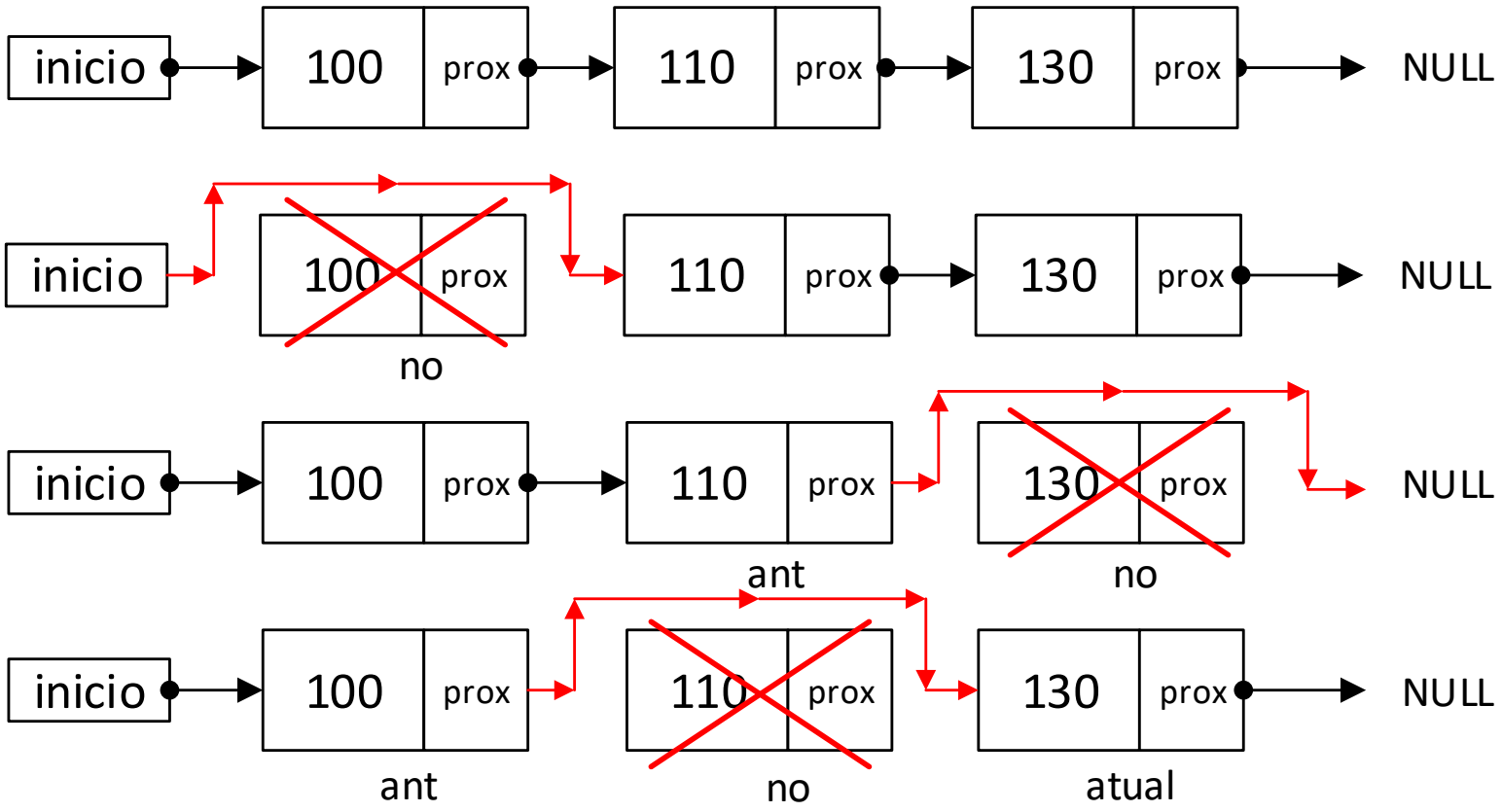
```c
//Arquivo main()
x = remove_lista(li, matricula);
if(x){
    printf("\nRemovido elemento com sucesso!");
}else{
    printf("\nNao foi possivel remover o elemento.");
}
```

Não precisa testar se a lista estava vazia, o while cuida disso.

Remove no início, meio e fim

Estrutura de Dados 1

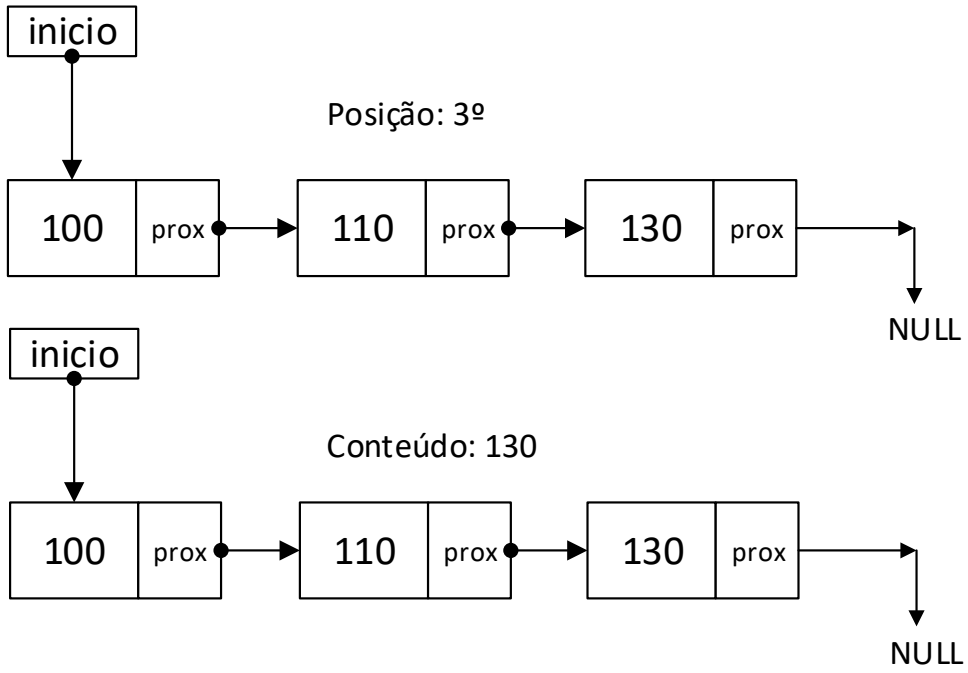
Lista Ligada – Remoção de qualquer elemento



Estrutura de Dados 1

Lista Ligada – Consultas

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



Estrutura de Dados 1

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;
    while (no != NULL && i < posicao){ //percorre a lista
        no = no->prox;
        i++;
    }
    if(no == NULL){ //trata se lista vazia ou não encontrou elemento
        return 0;
    }else{
        *al = no->dados;
        return 1;
    }
}
```

```c
//Arquivo main()
x = consulta_lista_pos(li, posicao, &al);
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);
```

Estrutura de Dados 1

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 matricula, ALUNO *al){
    if(li == NULL){
        return 0;
    }
    ELEM *no = *li;
    while(no != NULL && no->dados.matricula != matricula){
        no = no->prox;
    }
    if(no == NULL){
        return 0;
    }else{
        *al = no->dados;
        return 1;
    }

}
```

```c
//Arquivo main()
x = consulta_lista_mat(li, matricula, &al);
printf("\n\nMatricula encontrada na posicao %d:", posicao);
printf("\n%d", al.matricula);
printf("\n%.2f", al.n1);
printf("\n%.2f", al.n2);
printf("\n%.2f", al.n3);
```

Estrutura de Dados 1

Atividade

- Monte todas as funções da lista ligada e poste-a no Moodle como atividade 1.