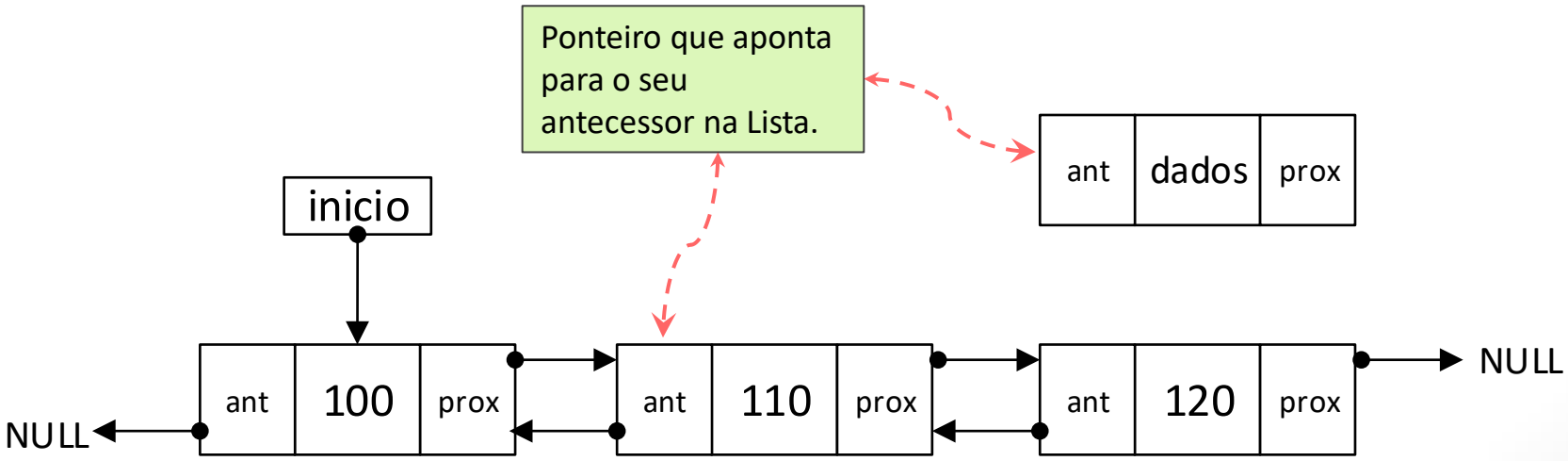
Estruturas de Dados 1

13 – Lista Duplamente Ligada

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

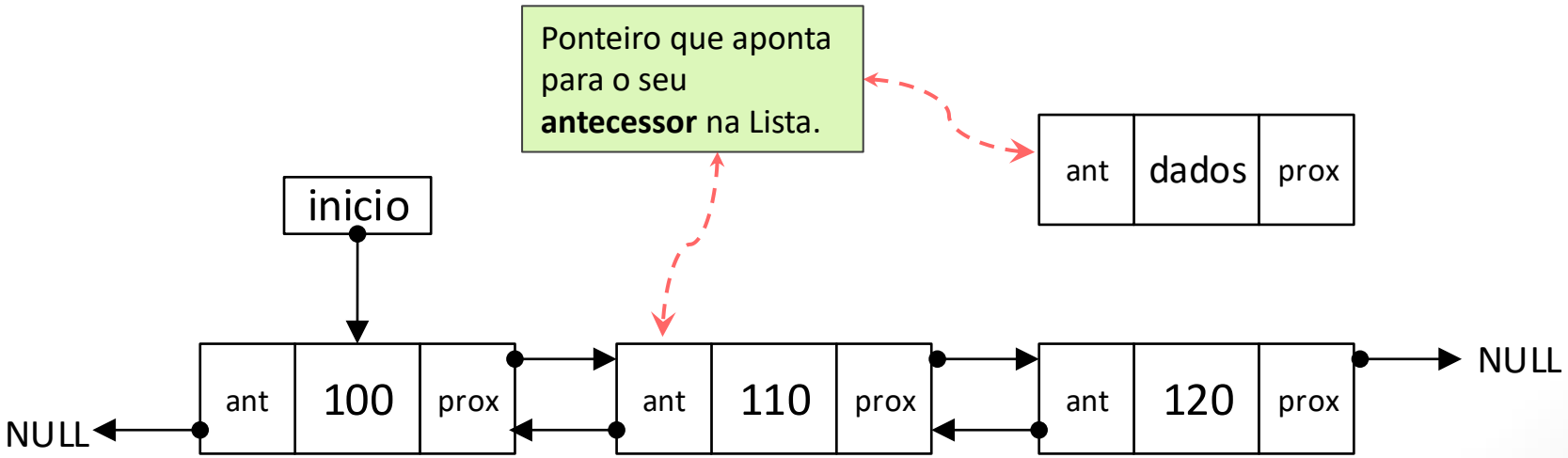
Lista Dinâmica Duplamente ligada

- Tipo de Lista onde cada elemento aponta para o seu sucessor e **antecessor** na Lista;
- Usa um ponteiro especial para o primeiro elemento da lista e uma indicação de final de Lista, **nos dois sentidos**.
- Diferente da Lista Dinâmica, este tipo de Lista possui três campos de informação:



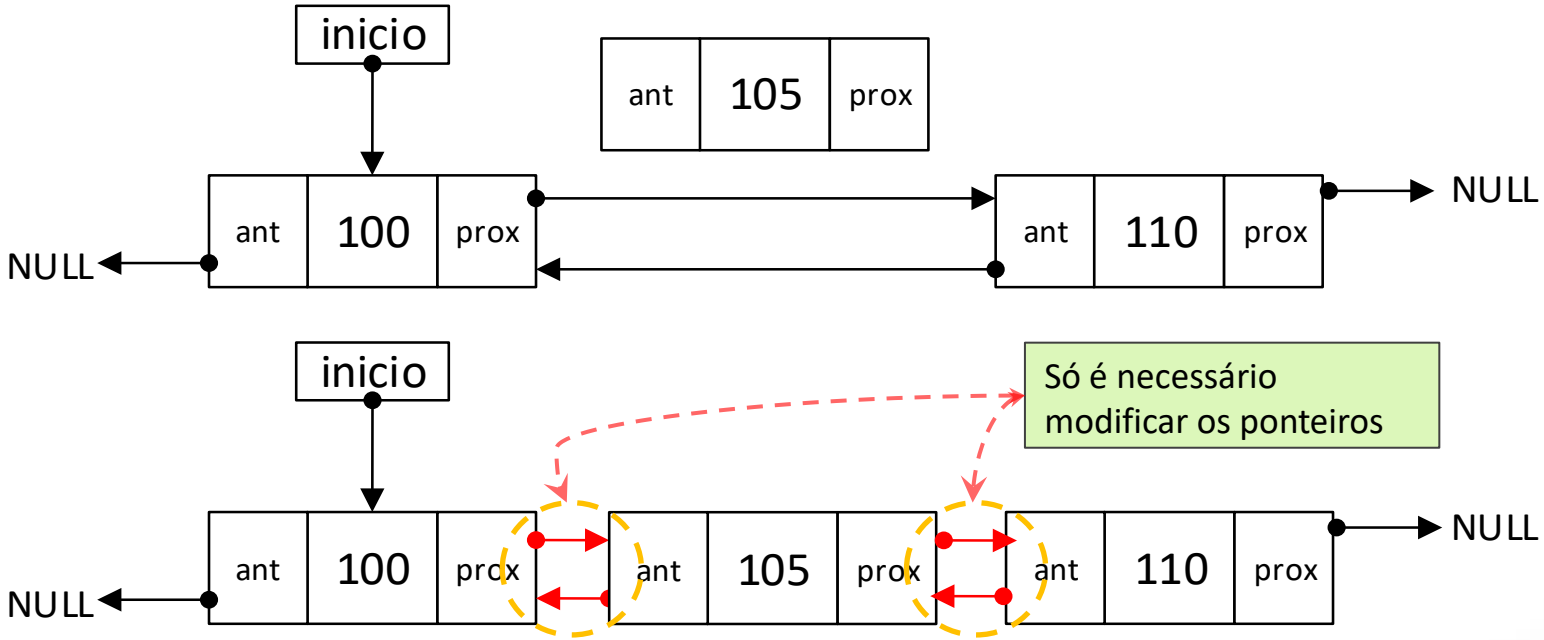
Lista Dinâmica Duplamente ligada

- Cada elemento é tratado como um ponteiro que é alocado dinamicamente, a medida que os dados são inseridos;
- Para armazenar o primeiro elemento, utilizamos um “ponteiro para ponteiro”, que pode armazenar o endereço de um ponteiro;
- Através desse ponteiro especial, fica mais fácil mudar o início da lista;
- O primeiro elemento, em seu campo “**ant**”, e último elemento, em seu campo “**prox**”, apontam para **NULL.**



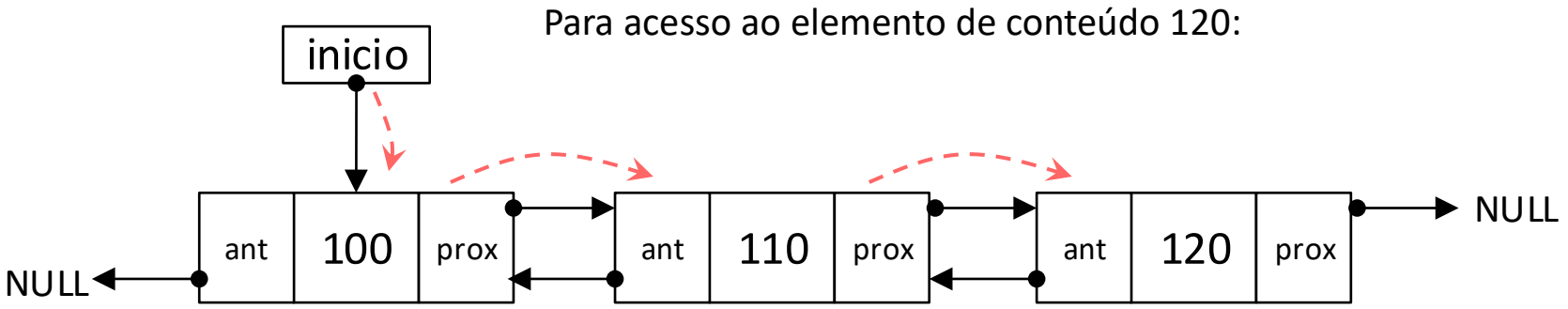
Lista Dinâmica Duplamente ligada

- Vantagens:
  - Melhor utilização dos recursos de memória;
  - Não é necessário definir previamente o tamanho da Lista;
  - Não é necessário movimentar elementos nas operações de inserção e remoção, como na Lista Estática.



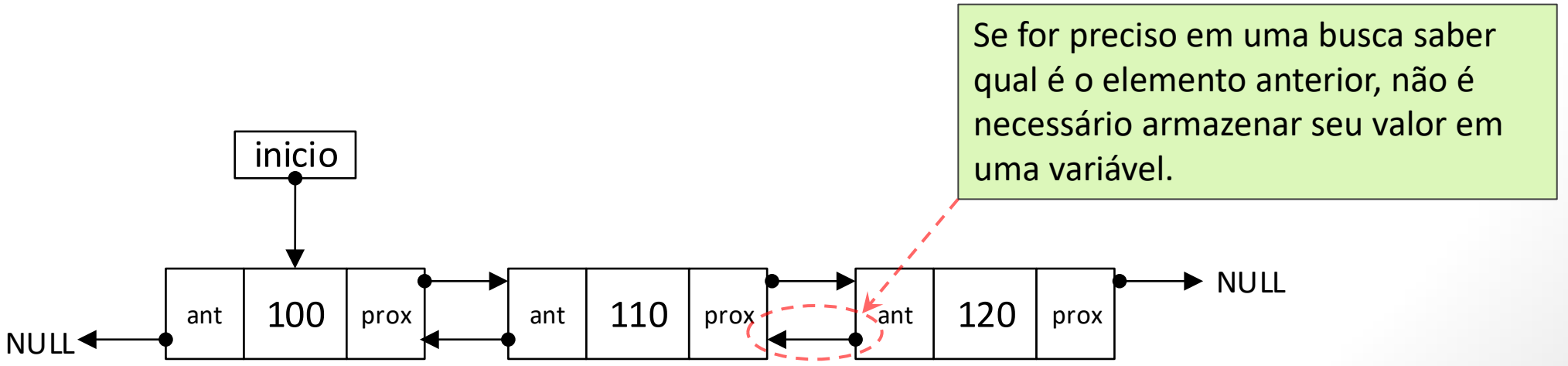
Lista Dinâmica Duplamente ligada

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



Lista Dinâmica Duplamente ligada

- Considerando as vantagens e desvantagens, quando utilizar este tipo de lista?
  - Quando não há a necessidade de garantir um espaço mínimo para a execução do aplicativo;
  - Tamanho máximo da Lista não é definido
  - Inserção e remoção em lista ordenada são as operações mais frequentes;
  - **Quando há a necessidade de acessar informações de um elemento antecessor.**



Lista Dinâmica Duplamente Ligada - Implementação

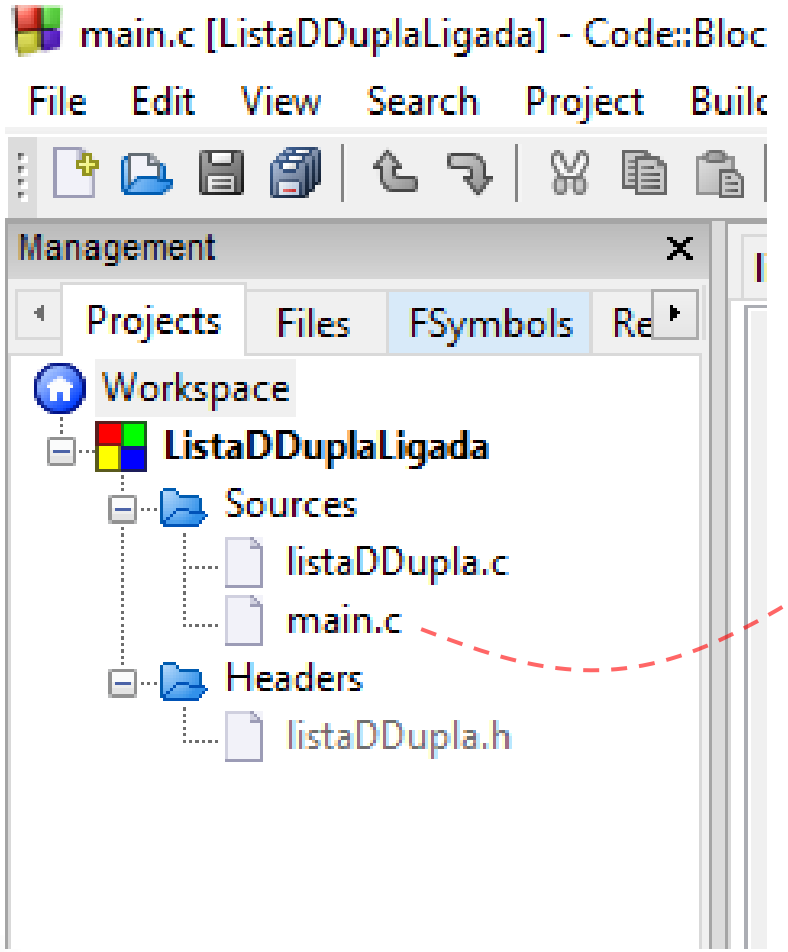
- `listaDDupla.h`:
  - Os protótipos das funções;
  - Tipo de dado armazenado na Lista;
  - O ponteiro Lista.

No arquivo `listaDDupla.h`, é declarado tudo aquilo que será visível ao programador que utilizará esta biblioteca

- `ListaDDupla.c`
  - O tipo de dado Lista;
  - Implementações das funções.

No arquivo `listaDDupla.c`, será definido tudo aquilo que ficará oculto do programador que utilizará esta biblioteca, e serão implementadas as funções definidas no arquivo `listaDDupla.h`

Lista Dinâmica Duplamente Ligada - Implementação




```c
//arquivo main.c
#include <stdio.h>
#include <stdlib.h>
#include "listaDDupla.h"

int main(){
    int x, matricula = 110, pos = 3;
    ALUNO al, al1, al2, al3;
    al1.matricula = 100;
    al1.n1 = 8.3;
    al1.n2 = 8.4;
    al1.n3 = 8.5;

    al2.matricula = 110;
    al2.n1 = 7.3;
    al2.n2 = 7.4;
    al2.n3 = 7.5;

    al3.matricula = 120;
    al3.n1 = 6.3;
    al3.n2 = 6.4;
    al3.n3 = 6.5;
```

Lista Dinâmica Duplamente Ligada - Implementação

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

typedef struct elemento *Lista;
```

Por estarem definidos dentro do arquivo "**.c**", os campos dessa estrutura não são visíveis ao usuário da biblioteca no arquivo `main()`, apenas seu outro nome, "`Lista`" definido no arquivo "`.h`", que que o `main()` tem acesso e pode apenas declarar um ponteiro para ele.

```c
//arquivo main.c
Lista *li; //ponteiro para ponteiro
```

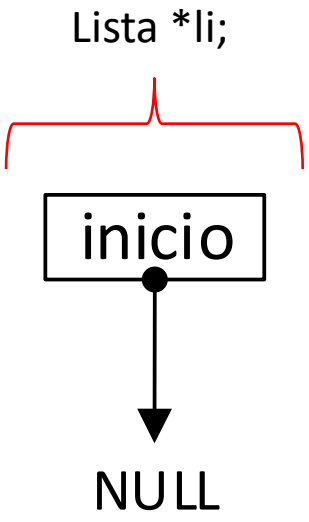
```c
//arquivo listaDDupla.c
#include <stdio.h>
#include <stdlib.h>
#include "listaDDupla.h"

struct elemento{
    struct elemento *ant;
    struct elemento *prox;
    ALUNO dados;
};

typedef struct elemento Elem;
```

Lista Dinâmica Duplamente Ligada - Implementação

```c
//arquivo listaDDupla.h
Lista *cria_lista();
```

```c
//arquivo listaDDupla.c
Lista *cria_lista(){
    Lista *li = (Lista*) malloc(sizeof(Lista));
    if(li != NULL){
        *li = NULL;
    }
    return li;
}
```

```c
//arquivo main.c
li = cria_lista();
```



Lista Dinâmica Duplamente Ligada - Implementação

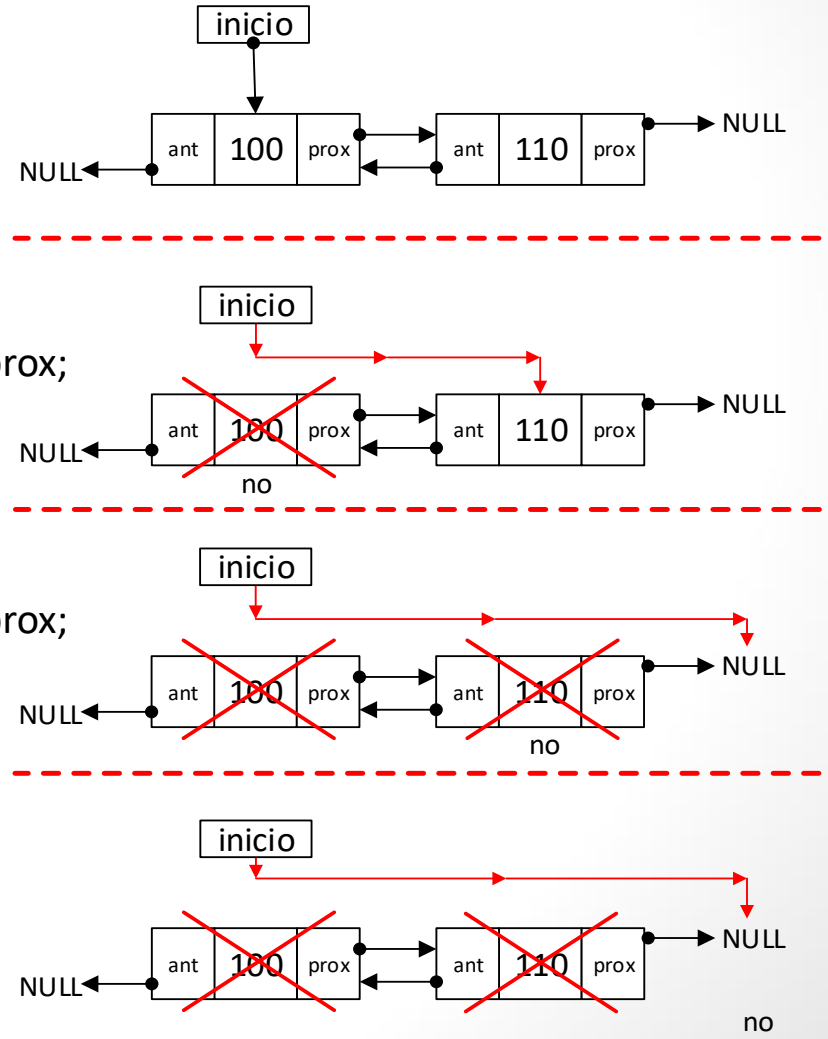
```
//arquivo listaDDupla.h
void destroi_lista(Lista *li);
```

Atenção!!
Esta deverá ser a última função chamada no seu programa principal!

```
//arquivo listaDDupla.c
void destroi_lista(Lista *li){
    if(li != NULL){
        Elem *no;
        while((*li) != NULL){
            no = *li;
            *li = (*li)->prox;
            free(no);
        }
        free(li);
    }
}
```

Não é necessário trabalhar com o ponteiro "ant", somente o "prox", igual a lista ligada.

```
//arquivo main.c
destroi_lista(li);
```



Lista Dinâmica Duplamente Ligada – Informações Básicas

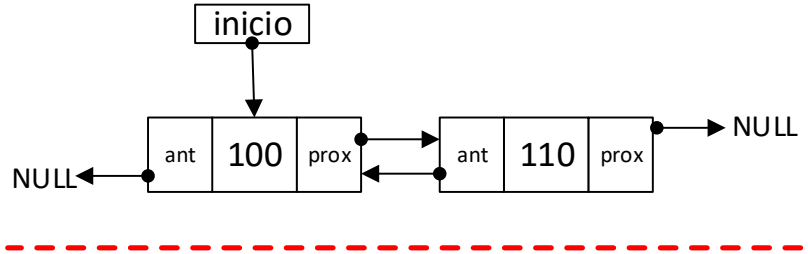
- Tamanho;
- Cheia;
- Vazia.

```c
//arquivo listaDDupla.h
int tamanho_lista(Lista *li);
```

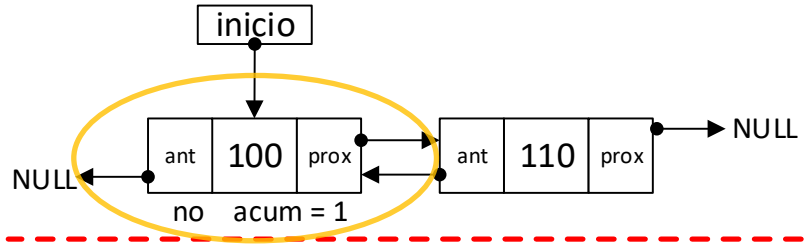
```c
//arquivo main.c
x = tamanho_lista(li);
printf("\nO tamanho da lista e: %d", x);
```

```c
//arquivo listaDDupla.c
int tamanho_lista(Lista *li){
    if(li == NULL){//se lista nula, não temos lista
        return 0;
    }
    int acum = 0;
    Elem *no = *li; //no auxiliar
    while(no != NULL){ //percorre a lista contando
        acum++;        //quantos nós existem
        no = no->prox; //desloca-se para o próximo elemento
    }
    return acum;
}
```

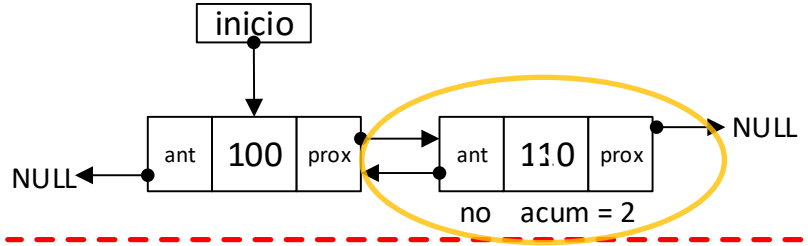
Lista Dinâmica Duplamente Ligada – Informações Básicas



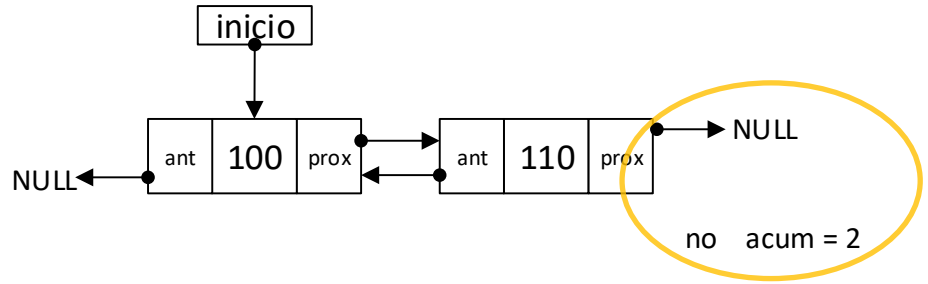

```
acum = 0;
no = *li;
```




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




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




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

Lista Dinâmica Duplamente Ligada – Informações Básicas

- Lista Cheia:

```c
//arquivo listaDDupla.h
int lista_cheia(Lista *li);
```

```c
//arquivo listaDDupla.c
int lista_cheia(Lista *li){
    return 0;
}
```

Uma Lista Dinâmica, somente será considerada cheia quando não houver mais espaço de memória disponível para alocar novos elementos...

```c
//arquivo main.c
x = lista_cheia(li);
if(x){
    printf("\nA lista esta cheia!");
}else{
    printf("\nA lista nao esta cheia.");
}
```

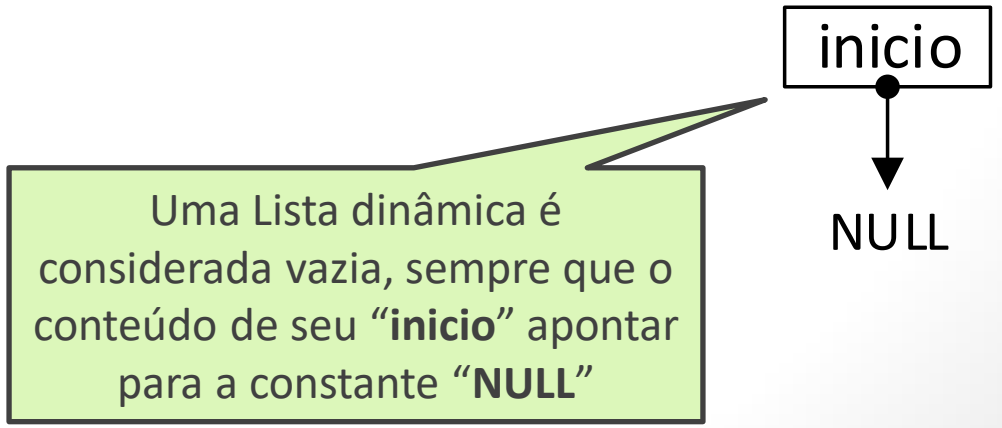
Lista Dinâmica Duplamente Ligada – Informações Básicas

- Lista Vazia:

```
//arquivo listaDDupla.h
int lista_vazia(Lista *li);
```

```
//arquivo main.c
x = lista_vazia(li);
if(x){
    printf("\nA lista esta vazia!");
}else{
    printf("\nA lista nao esta vazia.");
}
```

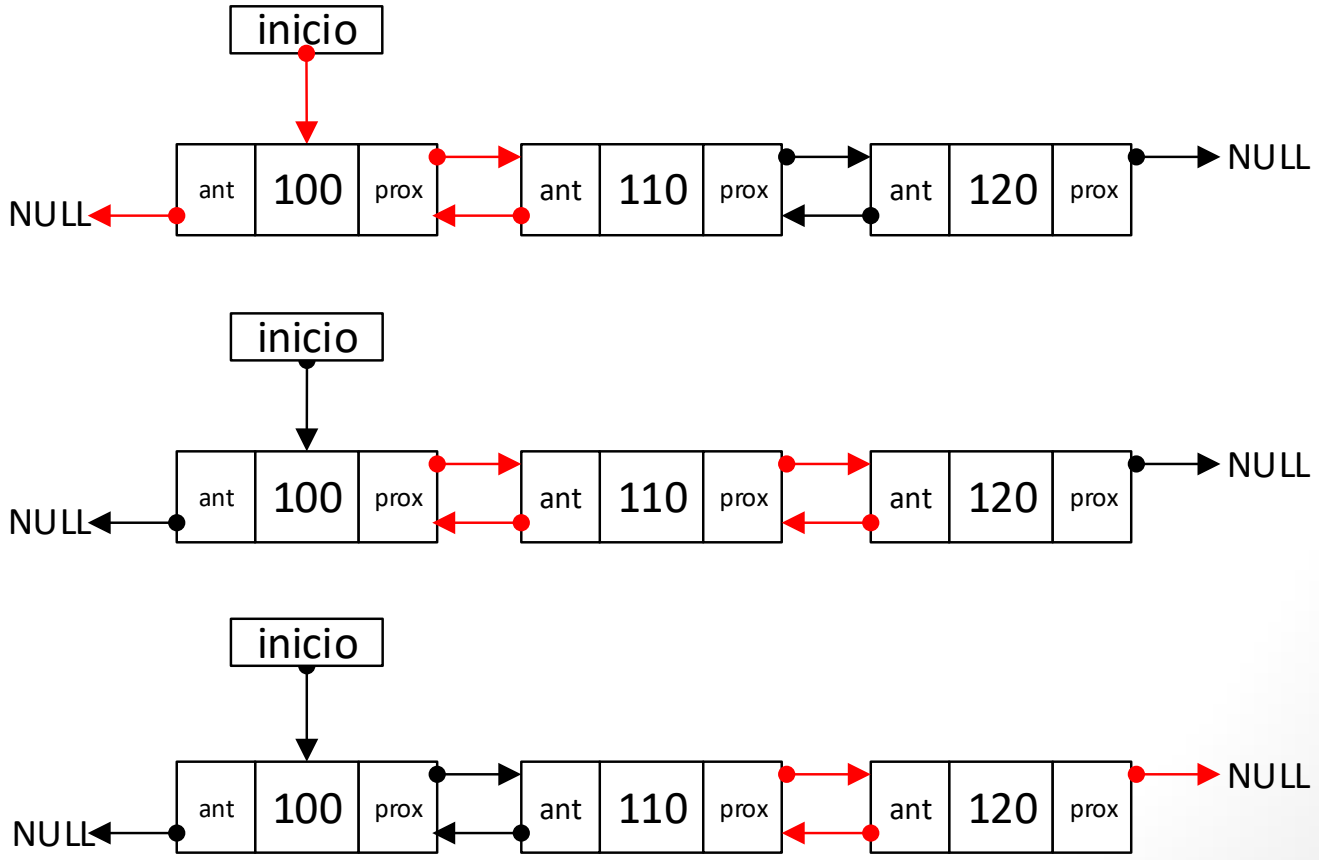
```
//arquivo listaDDupla.c
int lista_vazia(Lista *li){
    if(li == NULL){ //Não existe lista
        return 1;
    }
    if(*li == NULL){ //Lista está vazia
        return 1;
    }
    return 0;
}
```



Lista Dinâmica Duplamente Ligada – Inserção

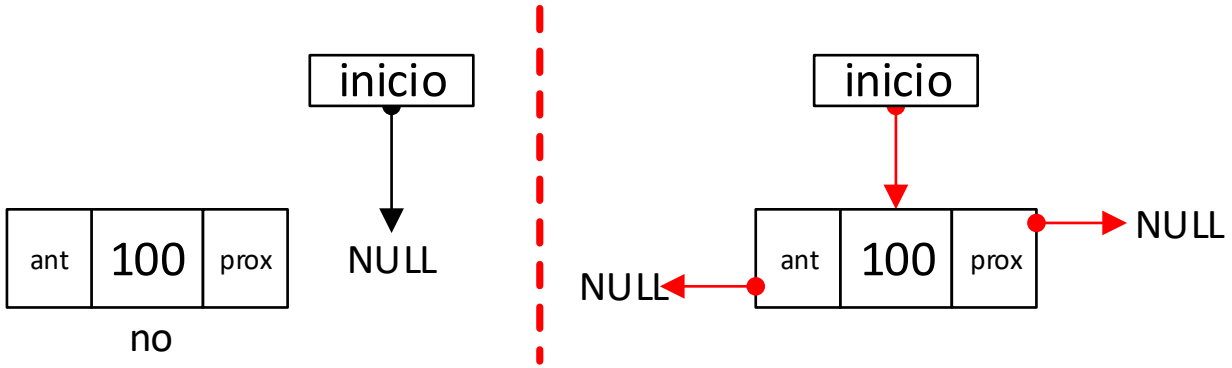
- Existem 3 tipos de inserção:
  - Início;
  - Meio ;
  - Fim.

No caso de uma Lista com alocação dinâmica, ela somente será considerada cheia quando não tivermos mais memória disponível no computador para alocar novos elementos. Isso ocorrerá quando a chamada da função `malloc()` retornar `NULL`.



Lista Dinâmica Duplamente Ligada – Inserção

- Existe o caso onde a inserção é feita em uma lista que está vazia;
- A inserção em Lista Dinâmica Duplamente Ligada é similar a inserção da Lista Dinâmica Simples, porém deve-se levar em consideração que este tipo de Lista conta com dois ponteiros para a devida atualização:
  - Anterior;
  - Próximo.



Lista Dinâmica Duplamente Ligada – Inserção no início

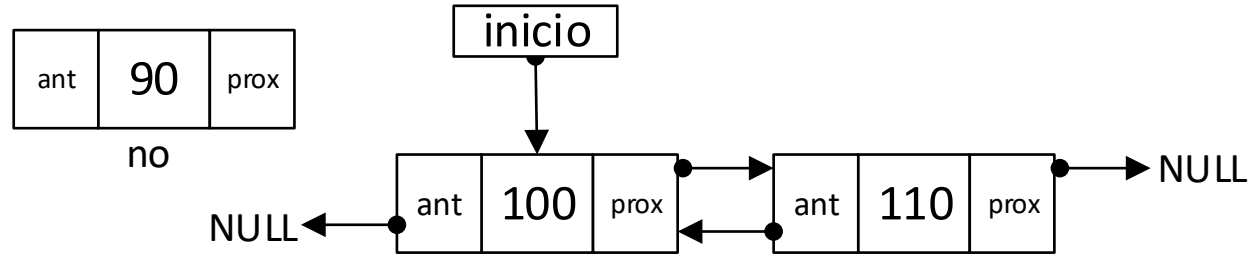
```c
//arquivo listaDDupla.h
int insere_lista_inicio(Lista *li, ALUNO al);
```

```c
//arquivo listaDDupla.c
int insere_lista_inicio(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;
    no->ant = NULL;
    if(*li != NULL){
        (*li)->ant = no;
    }
    *li = no;
    return 1;
}
```

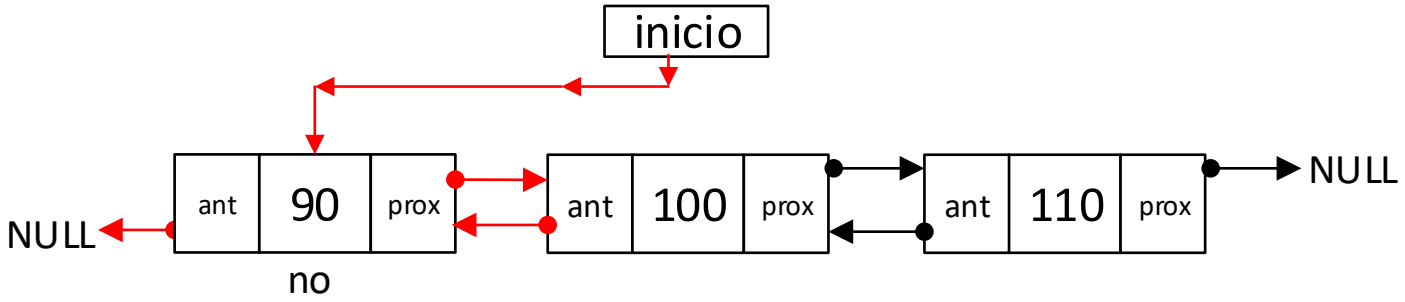
Em lista não vazia, aponta para anterior. Se a lista não era vazia, o antigo 1º nó, em sua parte "ant", passa a apontar para o nó a ser inserido na 1ª posição, e a cabeça recebe o endereço do novo nó inserido

```c
//arquivo main.c
x = insere_lista_inicio(li, al1);
if(x){
    printf("\nAluno inserido no inicio com sucesso!");
}else{
    printf("\nErro! aluno nao inserido.");
}
```

Lista Dinâmica Duplamente Ligada – Inserção no início



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



Se “li” não estava vazia:
(*li)->ant = no;
E por último:
*li = no;

Lista Dinâmica Duplamente Ligada – Inserção no final

```c
//arquivo listaDDupla.c
int insere_lista_final(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){
        no->ant = NULL;
        *li = no;
    }else{
        Elem *aux = *li;
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        aux->prox = no;
        no->ant = aux;
    }
    return 1;
}
```

```c
//arquivo listaDDupla.h
int insere_lista_final(Lista *li, ALUNO al);
```

```c
//arquivo main.c
x = insere_lista_final(li, al3);
if(x){
    printf("\nAluno inserido no final com sucesso!");
}else{
    printf("\nErro! aluno nao inserido.");
}
```

Como será o último nó, já recebe NULL

Se Lista vazia, inserção se dá no início

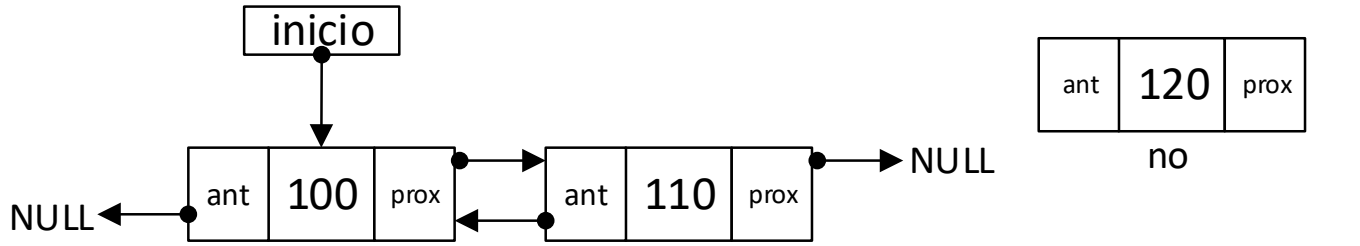
Ponteiro auxiliar para percorrer a Lista

Enquanto campo prox != NULL, percorre a Lista

O atual último elemento passa a apontar para o novo último elemento

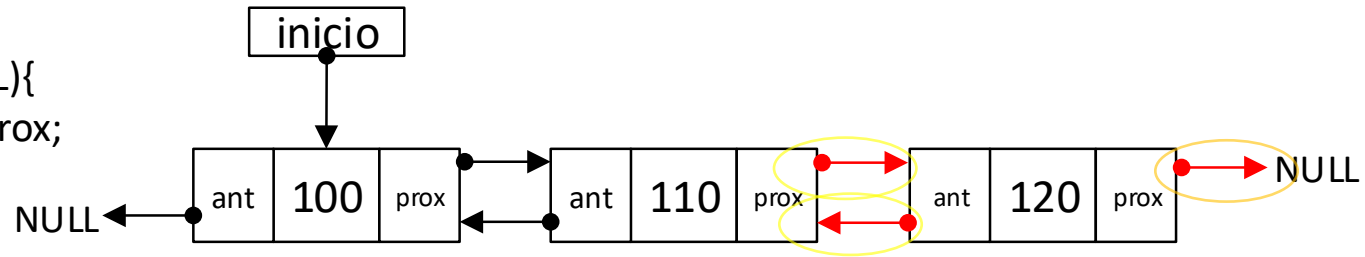
O novo último elemento, em seu campo " ant ", passa a apontar para o antigo último elemento

Lista Dinâmica Duplamente Ligada – Inserção no 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 Dinâmica Duplamente Ligada – Inserção Ordenada

```
//arquivo listaDDupla.h
int insere_lista(Lista *li, ALUNO al);
```

```
//arquivo main.c
x = insere_lista(li, al2);
if(x){
    printf("\nAluno inserido ordenadamente com sucesso!");
}else{
    printf("\nErro! Aluno nao inserido.");
}
```

Lista Dinâmica Duplamente Ligada – Inserção Ordenada

```c
//arquivo listaDDupla.c
int insere_lista(Lista *li, ALUNO al){
    if(li == NULL){
        return 0;
    }
    Elem *no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL){
        return 0;
    }
    no->dados = al;
    if(lista_vazia(li)){
        no->prox = NULL;
        no->ant = NULL;
        *li = no;
        return 1;
    }else{
        Elem *anterior, *atual = *li;
        while(atual != NULL && atual->dados.matricula < al.matricula){
            anterior = atual;
            atual = atual->prox;
        }
        if(atual == *li){
            no->ant = NULL;
            (*li)->ant = no;
            no->prox = (*li);
            *li = no;
        }else{
            no->prox = anterior->prox;
            no->ant = anterior;
            anterior->prox = no;
            if(atual != NULL){
                atual->ant = no;
            }
        }
    }
    return 1;
}
```

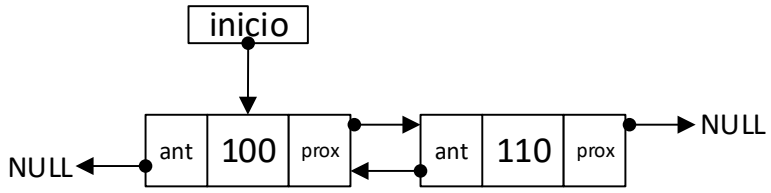
Insere em uma Lista vazia

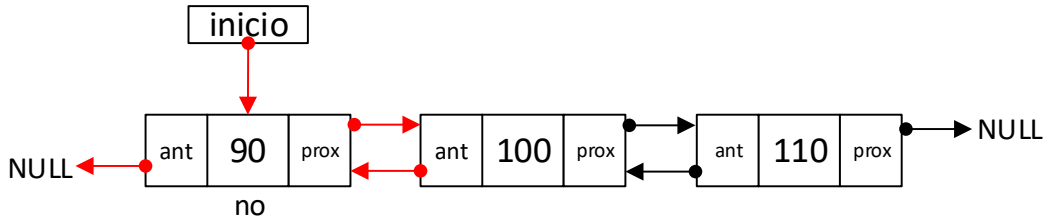
Insere no início, quando matricula é menor que todas. Se entrou aqui, significa que `atual` nunca percorreu a Lista

Esse `else`, trata qualquer inserção após a primeira posição, inclusive na última posição

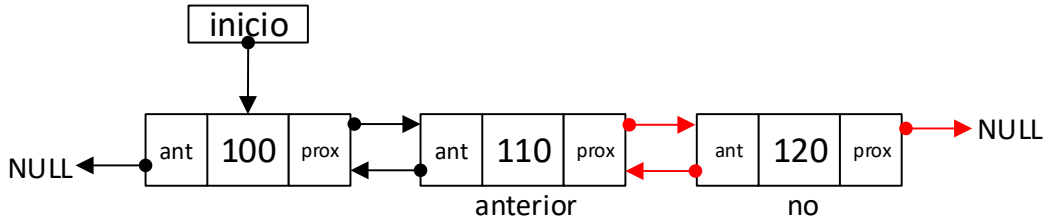
Esse laço `while` percorre a Lista a procura da posição correta de inserção do elemento

Lista Dinâmica Duplamente Ligada – Inserção Ordenada

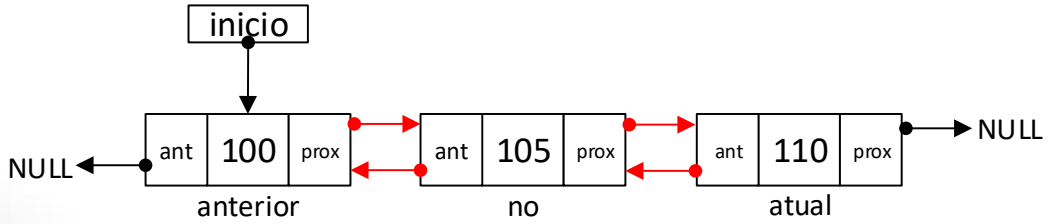




Inserindo no início:
no->ant = NULL;
(*li)->ant = no;
no->prox = (*li);
*li = no;



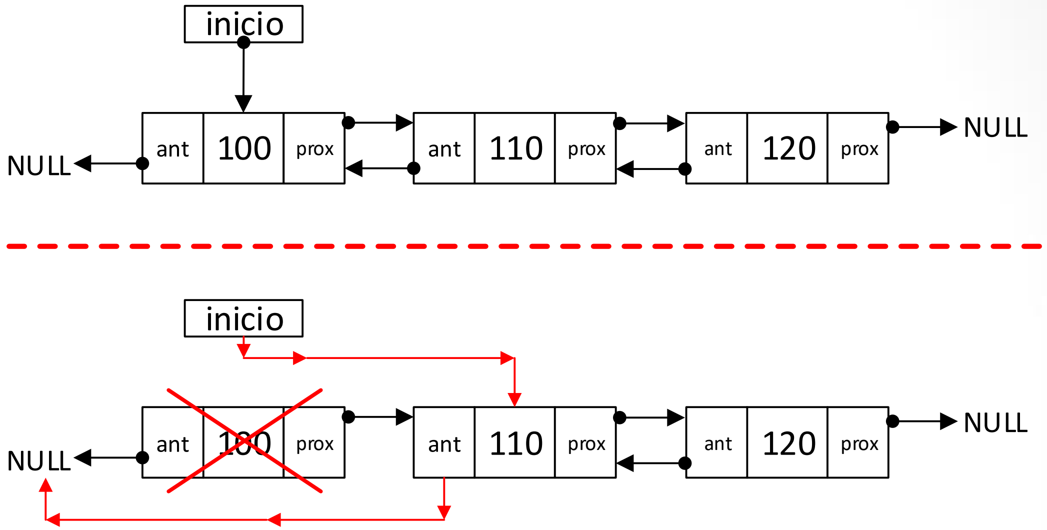
Inserindo depois de “anterior”:
no->prox = anterior->prox;
no->ant = anterior;
anterior->prox = no;

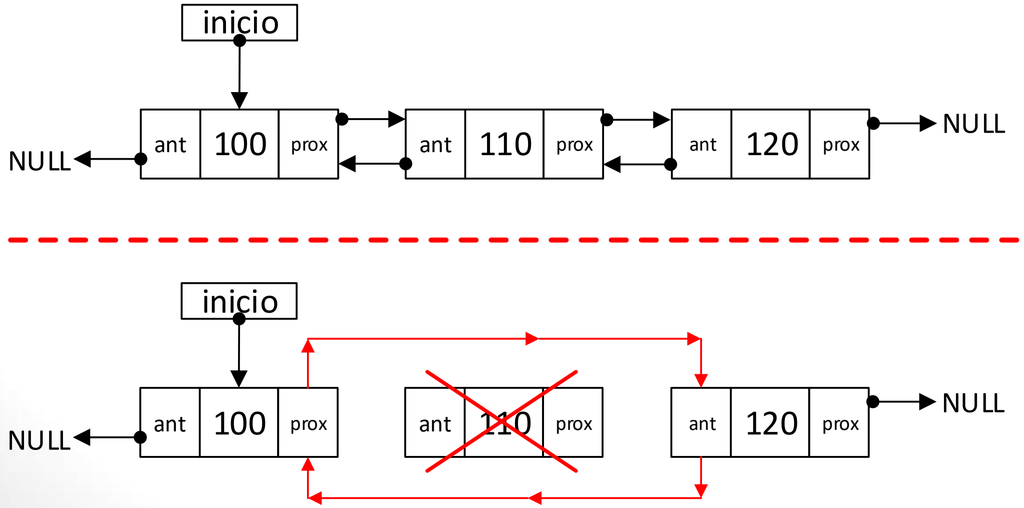


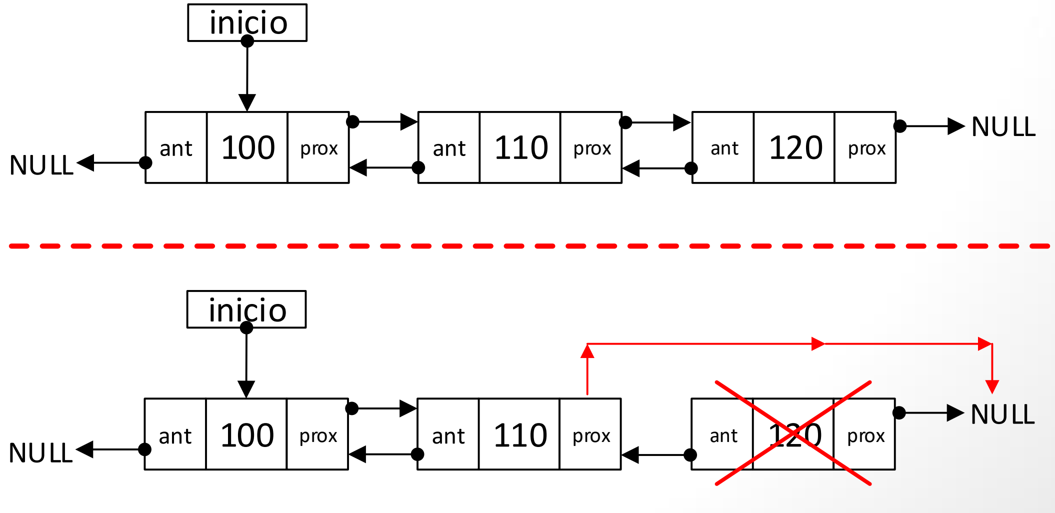
Inserindo quando não é o fim da lista”:
atual->ant = no;

Lista Dinâmica Duplamente Ligada – Remoção

- Existem 3 tipos de remoção:
  - Início;
  - Meio;
  - Fim;



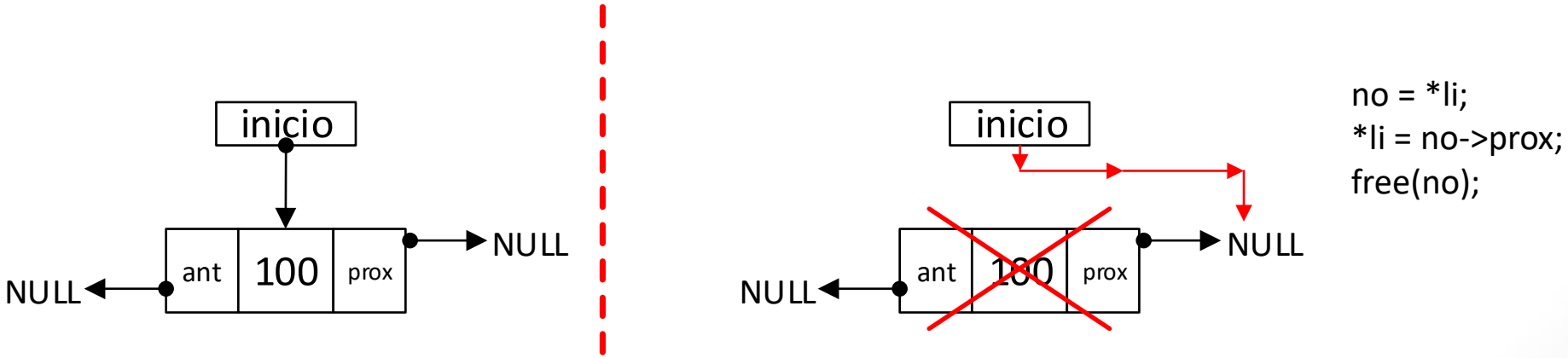




Lista Dinâmica Duplamente Ligada – Remoção

- Os 3 tipos de remoção trabalham juntos. A remoção sempre remove um elemento específico da Lista, o qual pode estar no início, meio ou final da Lista.
- Cuidado:
- Não se pode remover de uma Lista vazia;
- Removendo o último nó, a Lista fica vazia.

Estes itens têm que ser tratados.



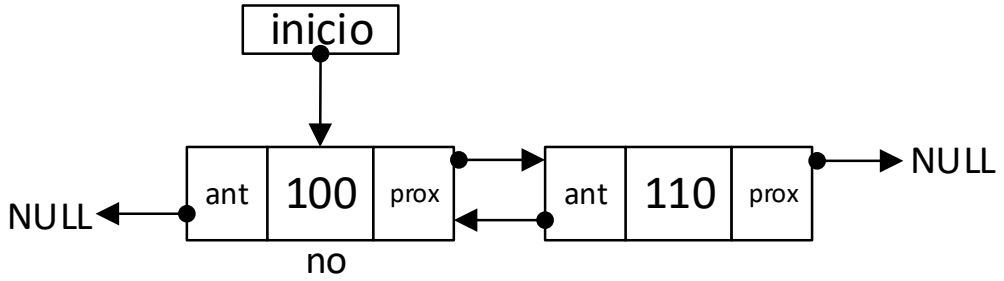
Lista Dinâmica Duplamente Ligada – Remoção no início

```c
//arquivo listaDDupla.h
int remove_lista_inicio(Lista *li);
```

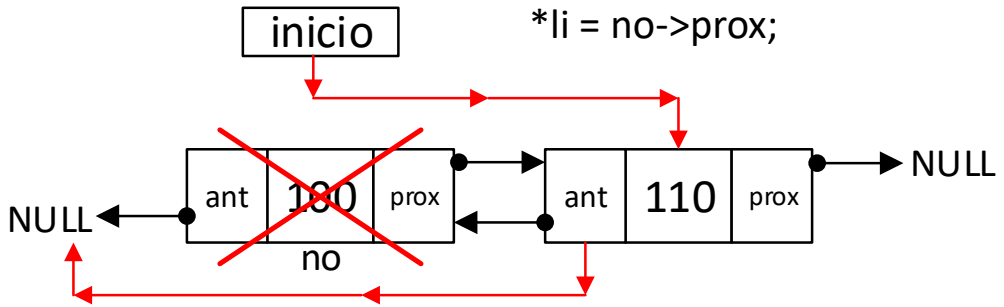
```c
//arquivo main.c
x = remove_lista_inicio(li);
if(x){
    printf("\nElemento removido no inicio com sucesso!");
}else{
    printf("\nErro! Elemento nao removido.");
}
```

```c
//arquivo listaDDupla.c
int remove_lista_inicio(Lista *li){
    if(li == NULL){ //se é válida
        return 0;
    }
    if((*li) == NULL){//se é vazia
        return 0;
    }
    Elem *no = *li;
    *li = no->prox;
    if(no->prox != NULL){
        no->prox->ant = NULL;
    }
    free(no);
    return 1;
}
```

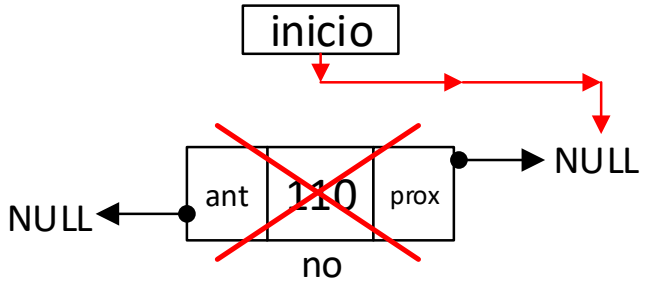
Lista Dinâmica Duplamente Ligada – Remoção no início



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



Se “nó” não é único elemento da Lista:
no->prox->ant = NULL;
E por último:
free(no);



Se “nó” é o único elemento da Lista, a Lista fica vazia.

Lista Dinâmica Duplamente Ligada – Remoção no final

```c
//arquivo listaDDupla.h
int remove_lista_final(Lista *li);
```

```c
//arquivo listaDDupla.c
int remove_lista_final(Lista *li){
    if(li == NULL){
        return 0;
    }
    if((*li) == NULL){
        return 0;
    }
    Elem *no = *li;
    while(no->prox != NULL){
        no = no->prox;
    }
    if(no->ant == NULL){
        *li = no->prox;
    }else{
        no->ant->prox = NULL;
    }
    free(no);
    return 1;
}
```

```c
//arquivo main.c
x = remove_lista_final(li);
if(x){
    printf("\nElemento removido no final com sucesso!");
}else{
    printf("\nErro! Elemento nao removido.");
}
```

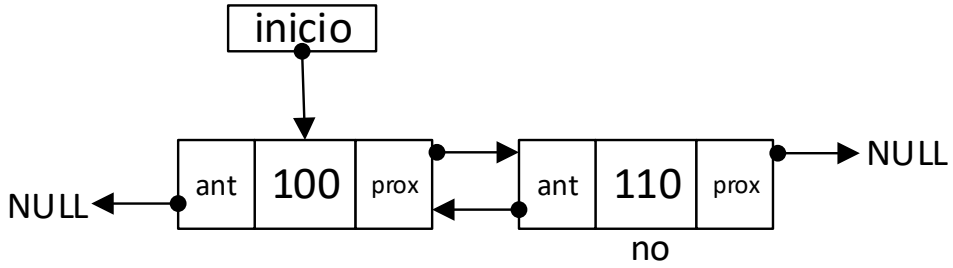
Percorre a Lista até que o campo prox aponte para NULL

Remove o primeiro e único elemento da lista. Para que este “ if ” seja satisfeito, é necessário termos só 1 elemento na Lista

Lista Dinâmica Duplamente Ligada – Remoção no final

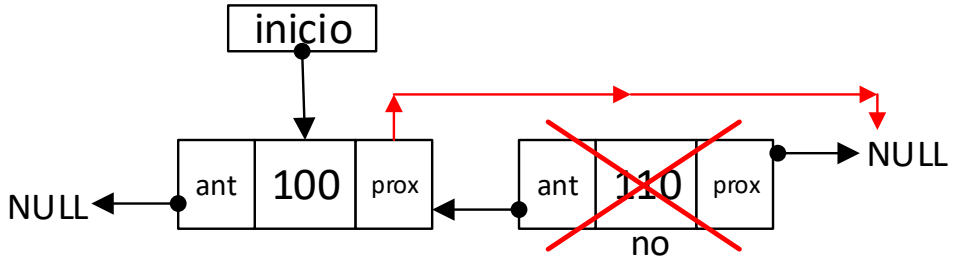
Procura o último elemento da Lista:

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



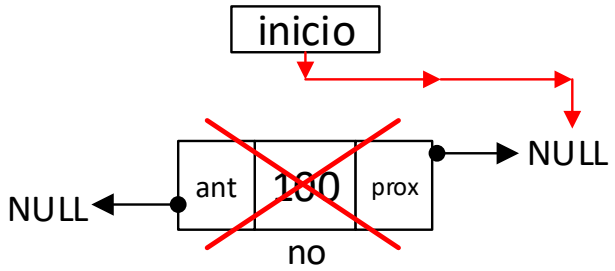
Nó não é o único elemento da Lista:

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



Nó é o único elemento, e a Lista fica vazia:

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



Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada - Remoção de qualquer elemento

```c
//arquivo listaDDupla.h
int remove_lista(Lista *li, int mat);
```

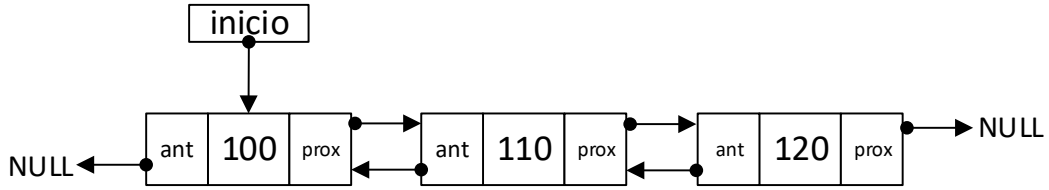
```c
//arquivo main.c
x = remove_lista(li, matricula);
if(x){
    printf("\nElemento removido com sucesso!");
}else{
    printf("\nErro! Elemento nao removido.");
}
```

```c
//arquivo listaDDupla.c
int remove_lista(Lista *li, int mat){
    if(li == NULL){
        return 0;
    }
    Elem *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        no = no->prox;
    }
    if(no == NULL){//não encontrado - chegou ao final
        return 0;  //e naõ encontrou o elemento
    }
    if(no->ant == NULL){//remover o primeiro
        *li = no->prox;
        //no->prox->ant = NULL;
    }else{
        no->ant->prox = no->prox; //remove no meio
    }
    if(no->prox != NULL){
        no->prox->ant = no->ant; //remove o último
    }
    free(no);
    return 1;
}
```

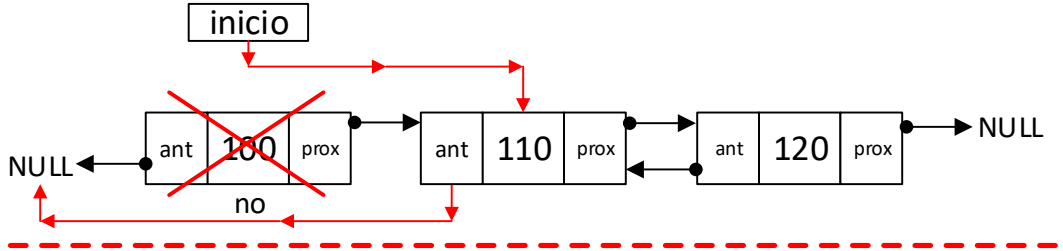
Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada - Remoção de qualquer elemento

Procura pelo nó a ser removido.

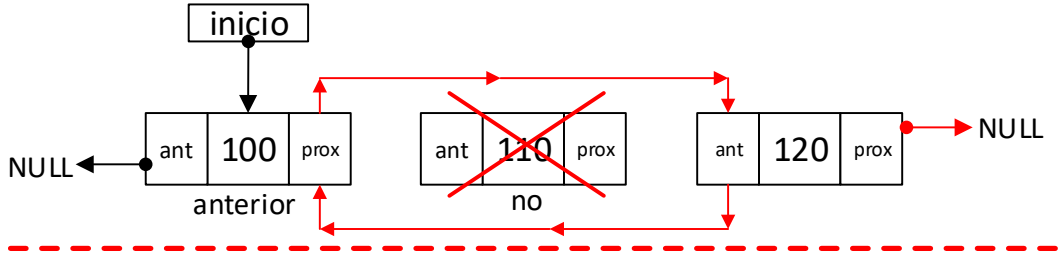


Remover do início:
*li = no->prox;



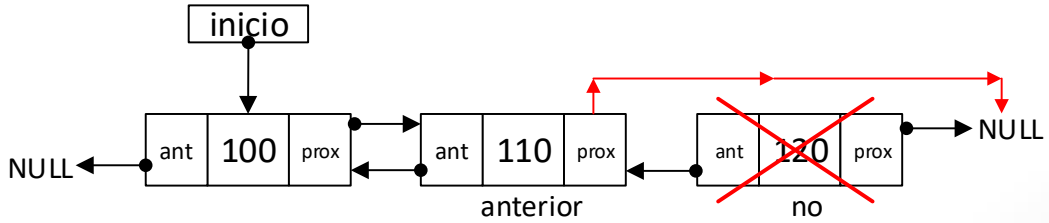
Não está removendo do final:
no->prox->ant = no->ant;

Serve para os 2 casos:



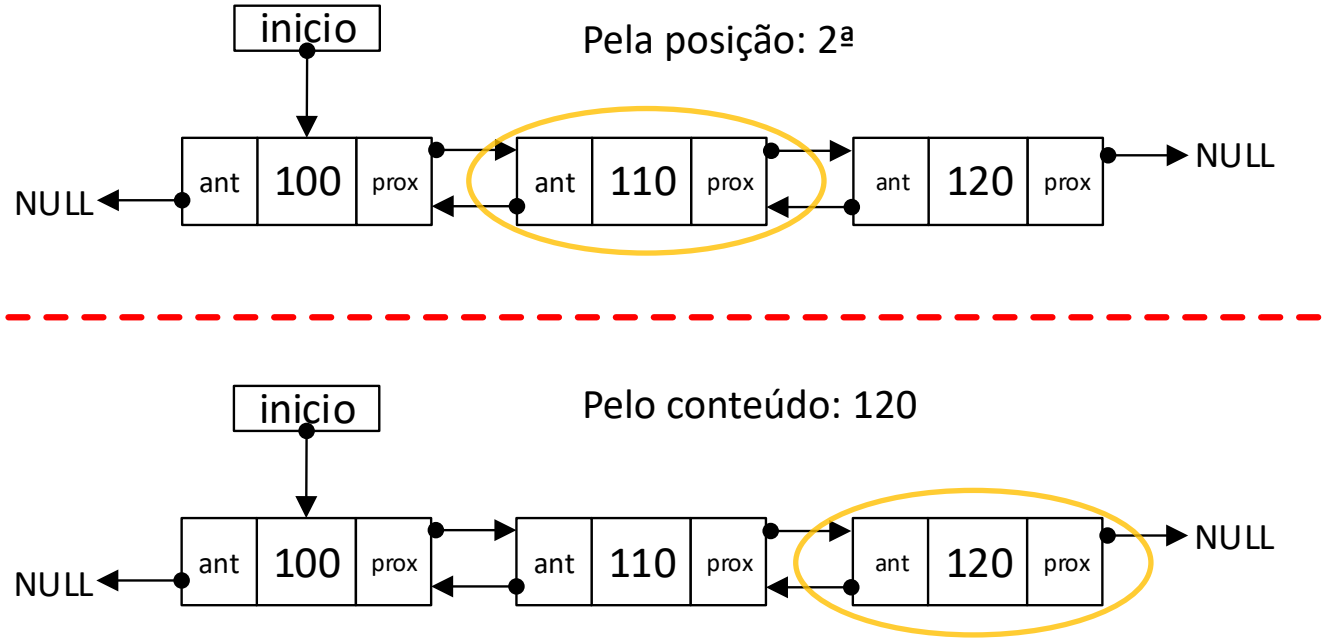
no->ant->prox = no->prox;

Por último:
free(no);



Lista Dinâmica Duplamente Ligada - Consultas

- Existem 2 maneiras de consultar um elemento em uma Lista Duplamente Ligada:
  - Por posição;
  - Pelo conteúdo.
- Ambos dependem de buscas, é necessário percorrer os elementos até encontrar o desejado.



Lista Dinâmica Duplamente Ligada – Consulta por posição

```c
//arquivo listaDDupla.h
int consulta_lista_pos(Lista *li, int pos, ALUNO *al);
```

```c
//arquivo listaDDupla.c
int consulta_lista_pos(Lista *li, int pos, ALUNO *al){
    if(li == NULL || pos <= 0){
        return 0;
    }
    Elem *no = *li;
    int i = 1;
    while(no != NULL && i < pos){
        no = no->prox;
        i++;
    }
    if(no == NULL){
        return 0;
    }else{
        *al = no->dados;
        return 1;
    }
}
```

Se no == NULL, significa que posição informada é maior que nº de elementos

Se no != NULL, o laço while foi interrompido porque elemento foi encontrado

```c
//arquivo main.c
x = consulta_lista_pos(li, pos, &al);
if(x){
    printf("\nConsulta realizada com sucesso:");
    printf("\nMatricula: %d", al.matricula);
    printf("\nNota 1:    %.2f", al.n1);
    printf("\nNota 2:    %.2f", al.n2);
    printf("\nNota 3:    %.2f", al.n3);
}else{
    printf("\nErro, consulta nao realizada.");
}
```

Lista Dinâmica Duplamente Ligada – Consulta por conteúdo

```c
//arquivo listaDDupla.h
int consulta_lista_mat(Lista *li, int mat, ALUNO *al);
```

```c
//arquivo listaDDupla.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;
    }
}
```

Se no == NULL, significa que matrícula informada não existe

Se no != NULL, o laço while foi interrompido porque a matrícula foi encontrada

```c
//arquivo main.c
x = consulta_lista_mat(li, matricula, &al);
if(x){
    printf("\nConsulta realizada com sucesso:");
    printf("\nMatricula: %d", al.matricula);
    printf("\nNota 1:    %.2f", al.n1);
    printf("\nNota 2:    %.2f", al.n2);
    printf("\nNota 3:    %.2f", al.n3);
}else{
    printf("\nErro, consulta nao realizada.");
}
```

Atividade

- Entregue todo o projeto compactado na plataforma Moodle.