



Estruturas de Dados 1

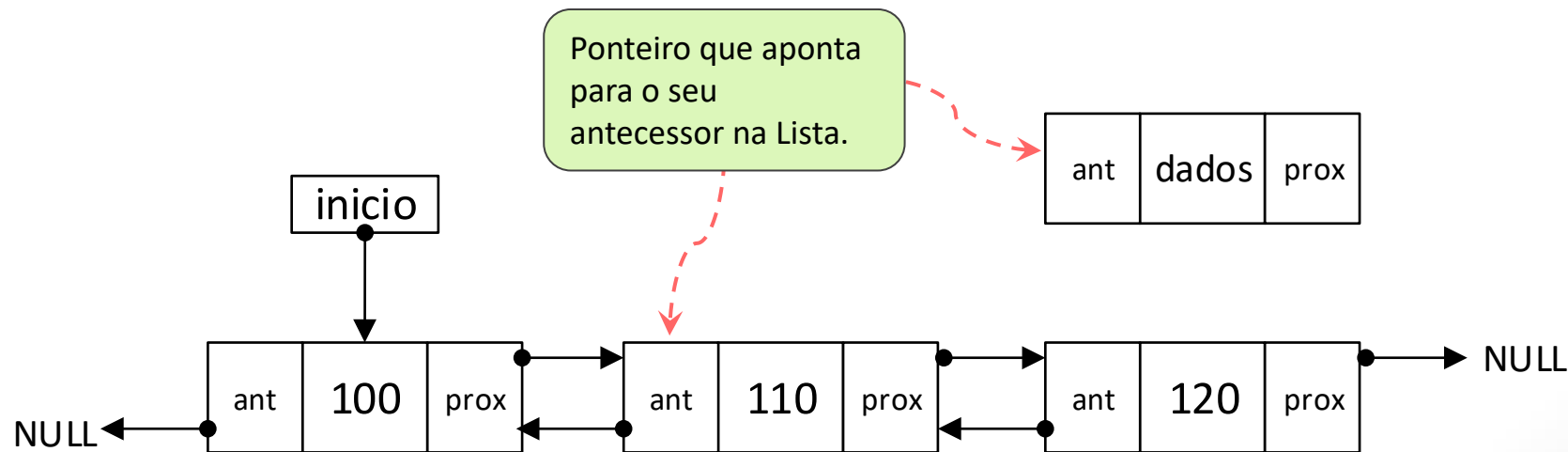
15 – Lista Duplamente Ligada

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

Estrutura de Dados 1

Lista Dinâmica Duplamente ligada

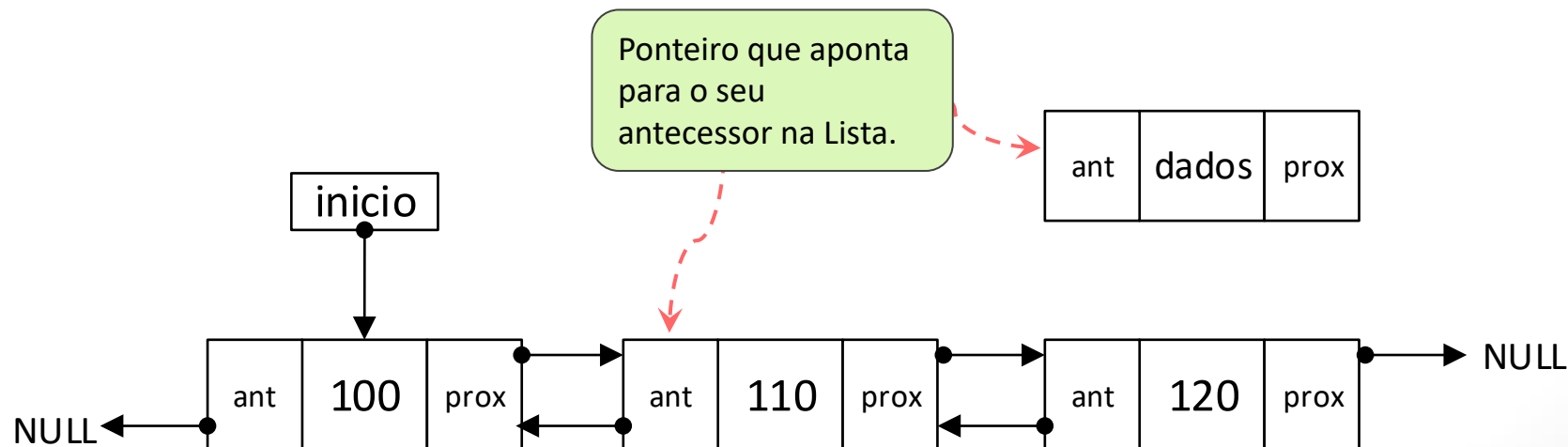
- Tipo de Lista onde cada elemento aponta para o seu sucessor e **antecessor** na Lista;
- Usa um ponteiro especial para o primeiro elemento da lista e uma indicação de final de Lista, **nos dois sentidos**.
- Diferente da Lista Dinâmica, este tipo de Lista possui três campos de informação:



Estrutura de Dados 1

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**.

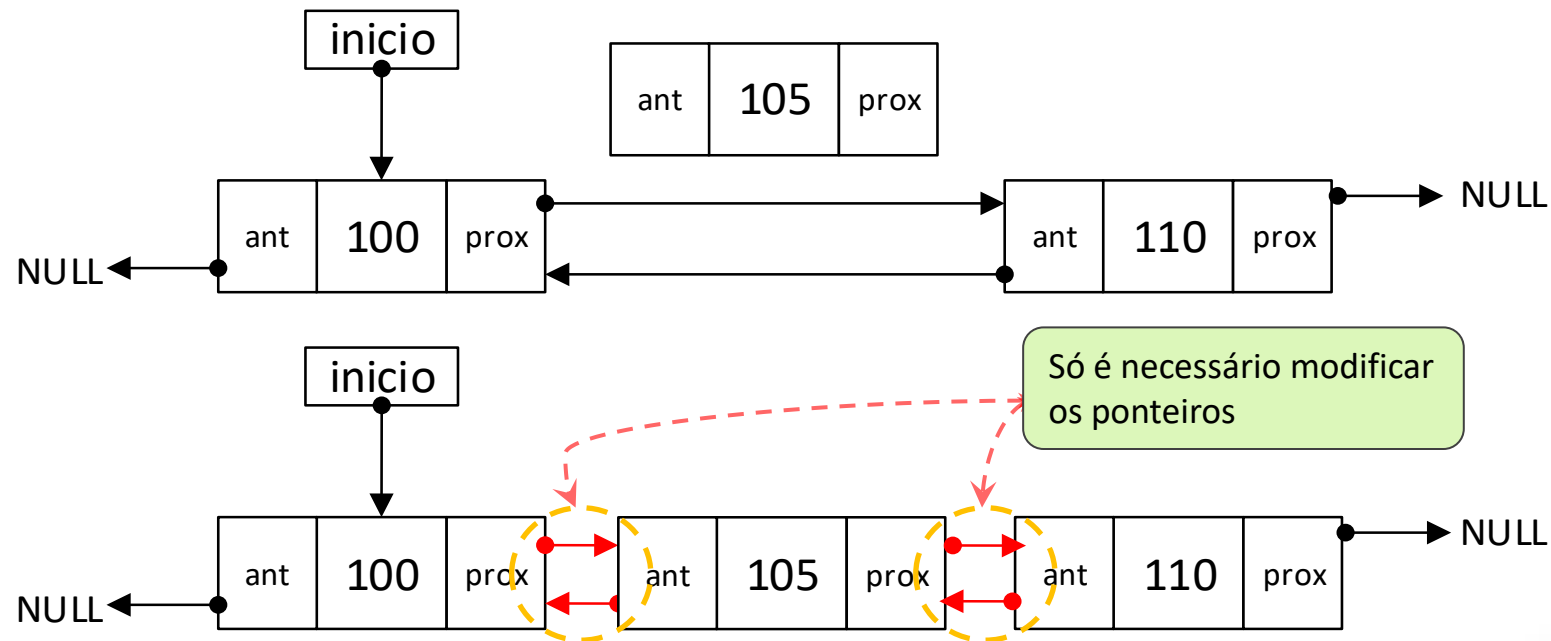


Estrutura de Dados 1

Lista Dinâmica Duplamente ligada

- Vantagens:

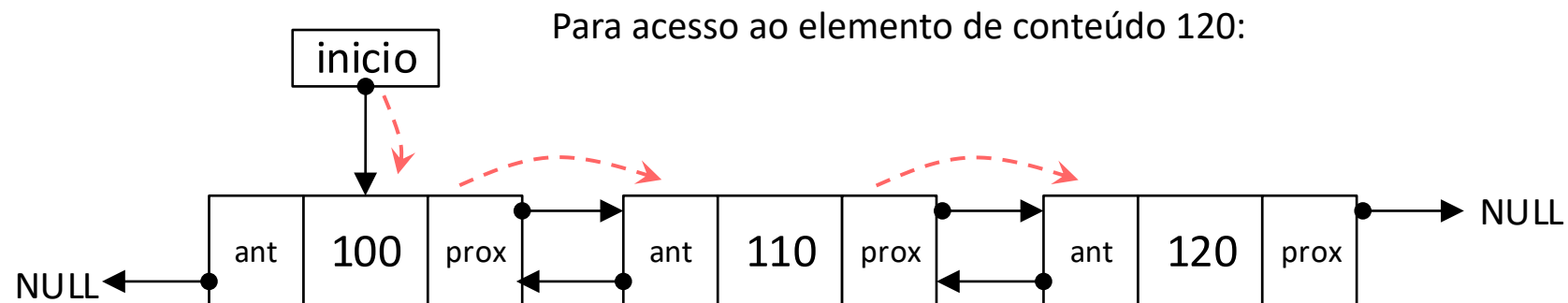
- Melhor utilização dos recursos de memória;
- Não é necessário definir previamente o tamanho da Lista;
- Não é necessário movimentar elementos nas operações de inserção e remoção, como na Lista Estática.



Estrutura de Dados 1

Lista Dinâmica Duplamente ligada

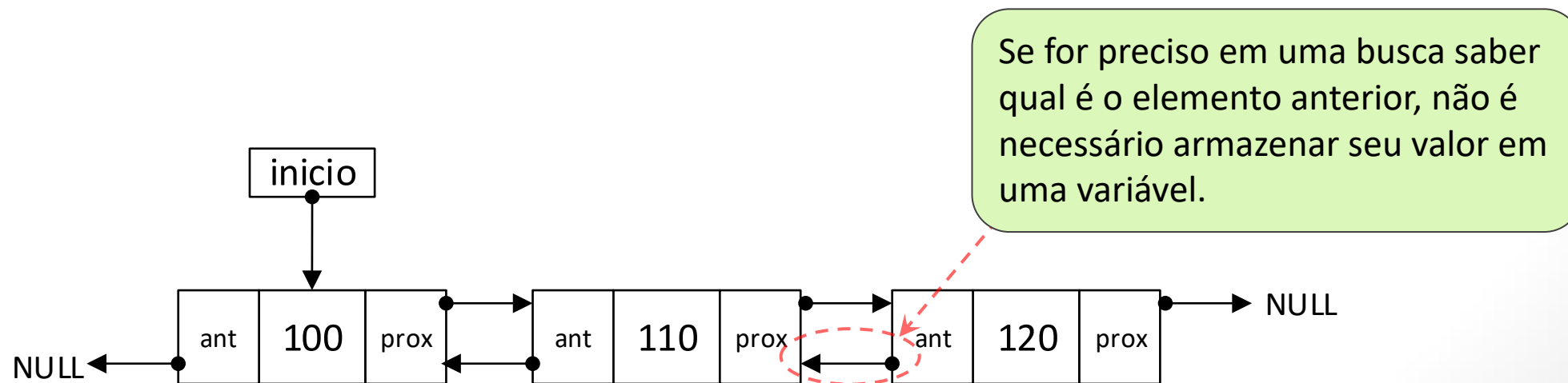
- Desvantagens:
 - Acesso indireto aos elementos;
 - Necessidade de percorrer toda a Lista para acessar um elemento.



Estrutura de Dados 1

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`

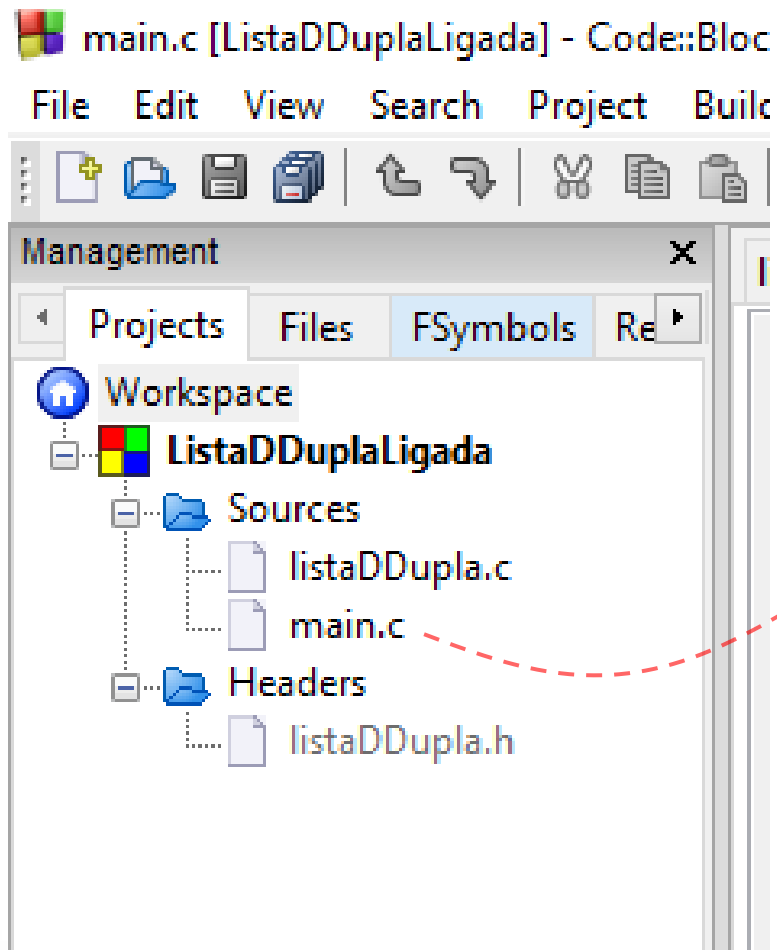
- O tipo de dado Lista;
- Implementações das funções.

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`



Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada - Implementação



```
//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;
```



Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada - Implementação

```
//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 o main() tem acesso, e pode apenas declarar um ponteiro para ele.

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

```
//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;
```



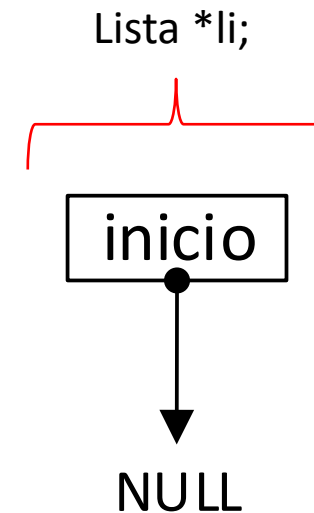
Lista Dinâmica Duplamente Ligada - Implementação



```
//arquivo listaDDupla.h  
LISTA *criaLista();
```

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

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





Lista Dinâmica Duplamente Ligada - Implementação

```
//arquivo listaDDupla.h  
void apagaLista(LISTA *li);
```

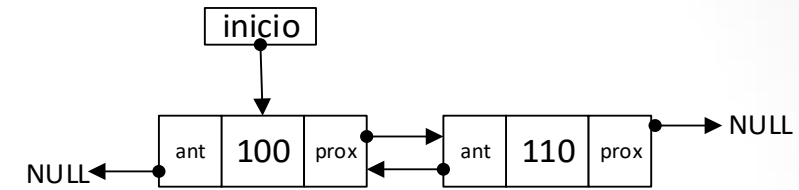
```
//arquivo listaDDupla.c  
void apagaLista(LISTA *li) {  
    if(li != NULL) {  
        ELEM *no;  
        while((*li) != NULL) {  
            no = *li;  
            *li = (*li)->prox;  
            free(no);  
        }  
        free(li);  
    }  
}
```

Atenção!!

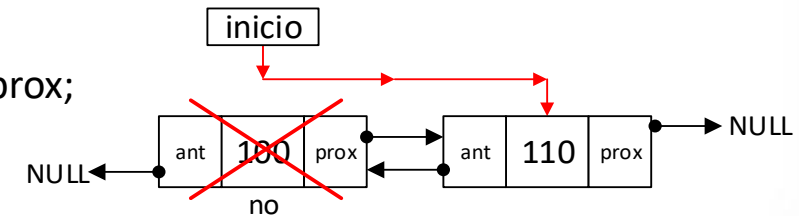
Esta deverá ser a última função chamada no seu programa principal!

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

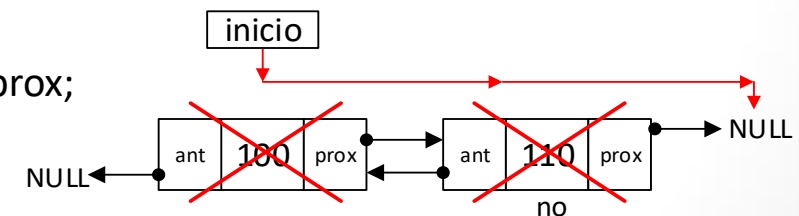
```
//arquivo main.c  
apagaLista(li);
```



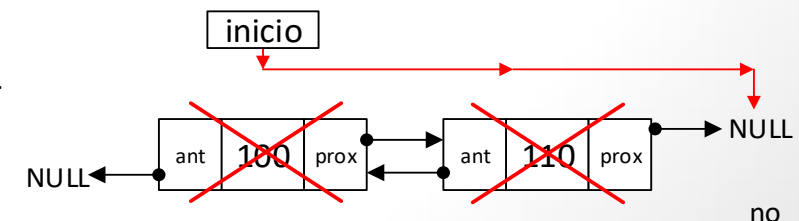
no = *li;
*li = (*li)->prox;
free(no);



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



Fim:
no == NULL





Lista Dinâmica Duplamente Ligada - Implementação

- Este será um modelo básico de uma lista duplamente 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

```
//Arquivo listaDDupla.h  
void abortaPrograma();
```

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



Lista Dinâmica Duplamente Ligada – Informações Básicas

- Tamanho;
- Cheia;
- Vazia.

```
//arquivo listaDDupla.h  
int tamanhoLista(LISTA *li);
```

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

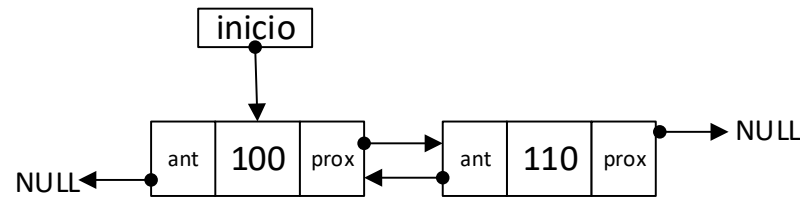
```
//arquivo listaDDupla.c  
int tamanhoLista(LISTA *li){  
    if(li == NULL){  
        abortaPrograma();  
    }  
    int acum = 0;  
    ELEM *no = *li;  
    while(no != NULL){  
        acum++;  
        no = no->prox;  
    }  
    return acum;  
}
```

Se o ponteiro li, contém NULL, significa que a Lista não foi alocada, então o programa é abortado.

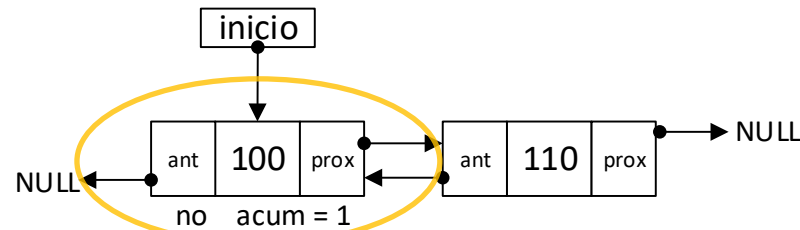
O nó auxiliar recebe o primeiro elemento da lista e a percorre até encontrar seu final (NULL) contando a cada deslocamento, quantos nós existem seguindo para o próximo elemento.

Estrutura de Dados 1

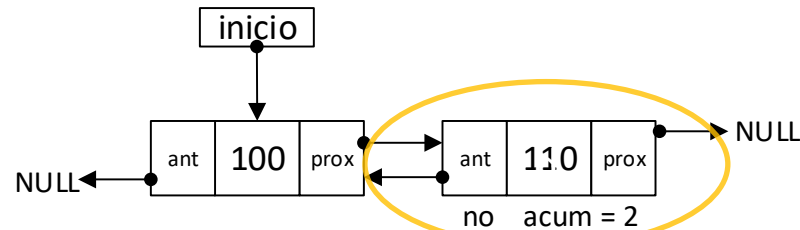
Lista Dinâmica Duplamente Ligada – Informações Básicas



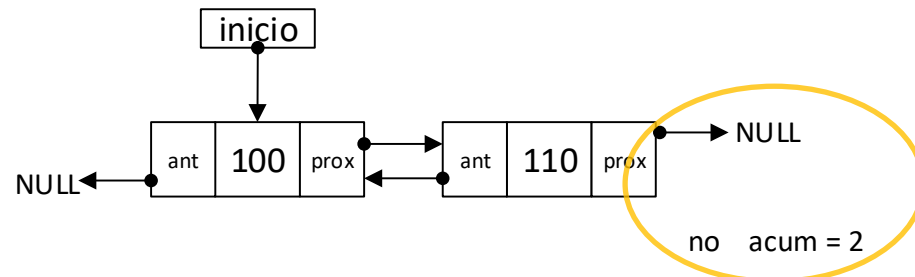
```
acum = 0;  
no = *li;
```



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



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



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





Lista Dinâmica Duplamente Ligada – Informações Básicas

- Lista Cheia:

```
//arquivo listaDDupla.h  
int listaCheia(LISTA *li);
```

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

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

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



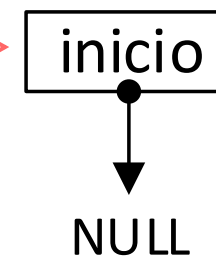
Lista Dinâmica Duplamente Ligada – Informações Básicas

- Lista Vazia:

```
//arquivo listaDDupla.h
int listaVazia(LISTA *li);
```

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

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



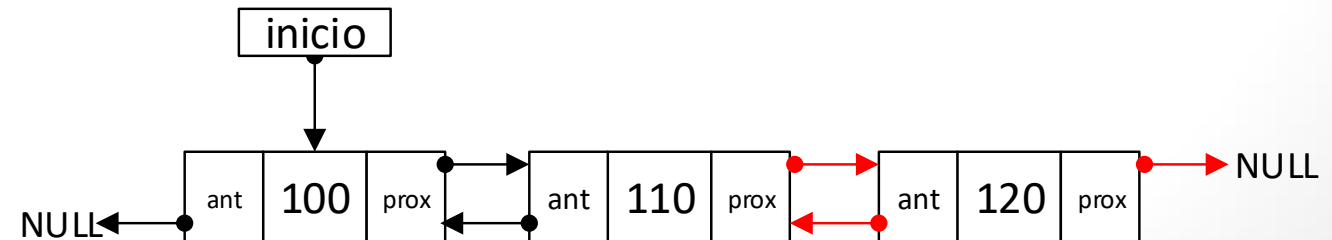
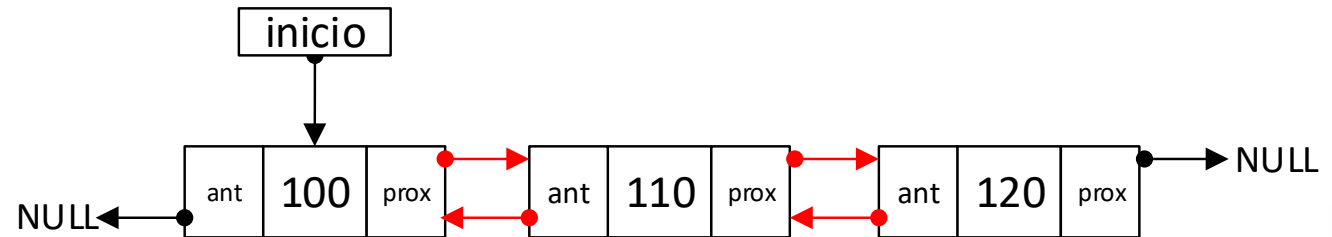
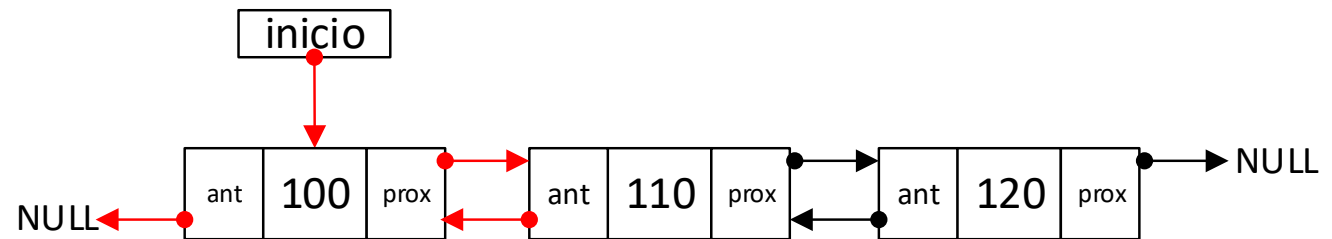
Uma Lista dinâmica é considerada vazia, sempre que o conteúdo de sua “cabeça” ou “**inicio**” apontar para a constante “**NULL**”

Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada – Inserção

- Existem 3 tipos de inserção:

- Início;
- Meio ;
- Fim.



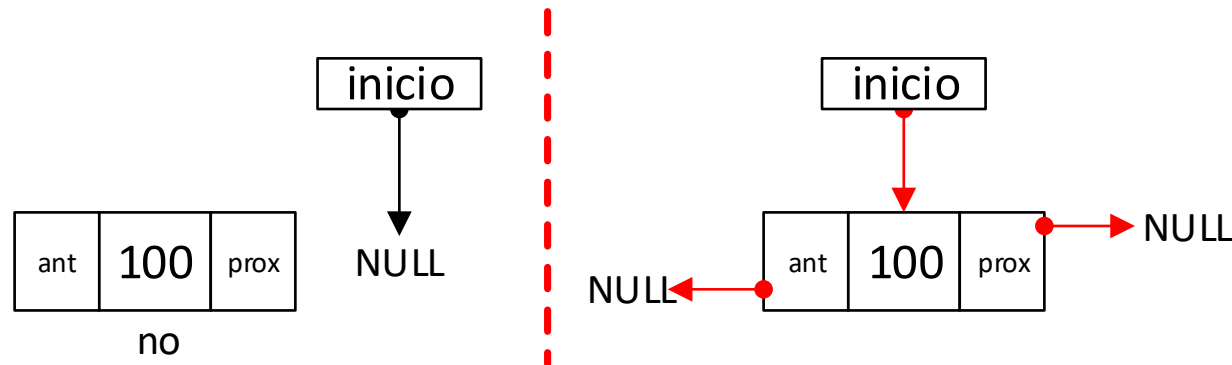
No caso de uma Lista com alocação dinâmica, ela somente será considerada cheia quando não existir mais memória disponível no computador para alocar novos elementos. Quando isso ocorrer a função `malloc()` retornar NULL como resposta.



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 o tratamento para as ligações dos ponteiros acontece de forma diferente, pois agora existe um ponteiro a mais para a ligação com o elemento anterior:

- Anterior;
- Próximo.





Lista Dinâmica Duplamente Ligada – Inserção no início

```
//arquivo listaDDupla.h
int insereInicio(LISTA *li, ALUNO al);

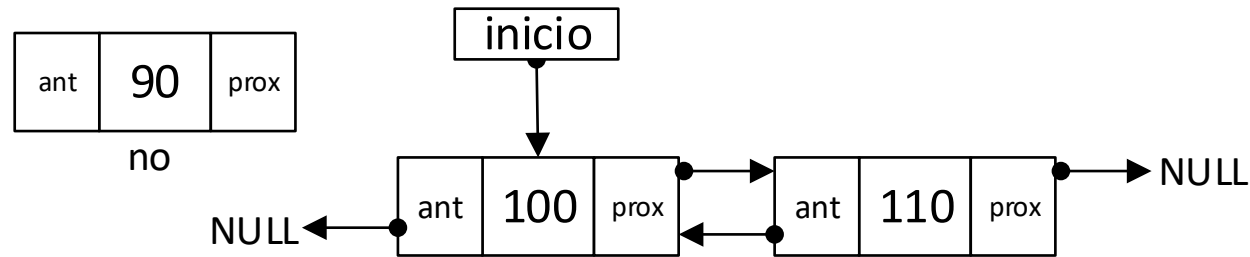
//arquivo listaDDupla.c
int insereInicio(LISTA *li, ALUNO al){
    if(li == NULL){
        abortaPrograma();
    }
    ELEM *no = (ELEM*) malloc(sizeof(ELEM));
    if(no == NULL){
        return 0;
    }
    no->dados = al;
    no->prox = *li;
    no->ant = NULL;
    if(*li != NULL){
        (*li)->ant = no;
    }
    *li = no;
    return al.matricula;
}
```

Se a lista não estava vazia, o atual 1º nó (que então passará a ser o segundo), em sua parte "ant", passa a apontar para o novo nó que será inserido na 1ª posição. Por fim a cabeça, ou ponteiro início, recebe o endereço do novo nó inserido

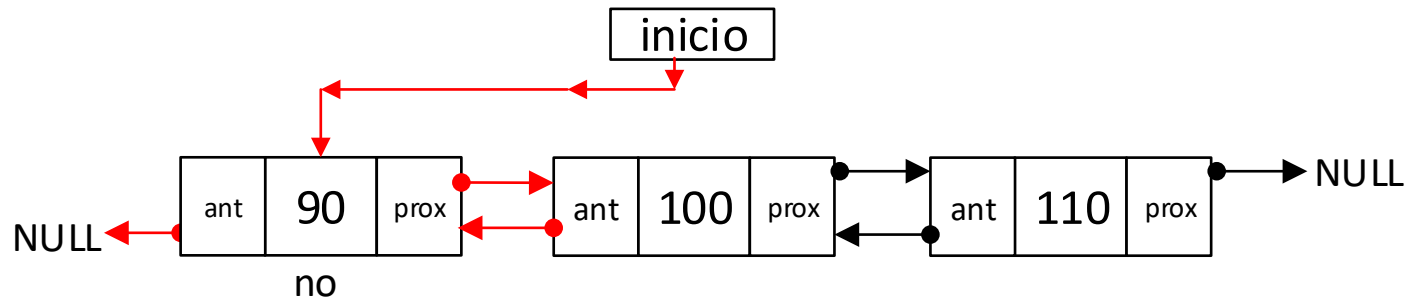
```
//arquivo main.c
x = insereInicio(li, all);
if(x){
    printf("\nAluno %d inserido no inicio com sucesso!", x);
}else{
    printf("\nErro! aluno nao inserido.");
}
```



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

```
//arquivo listaDDupla.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){
        no->ant = NULL;
        *li = no;
    }else{
        ELEM *aux = *li;
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        aux->prox = no;
        no->ant = aux;
    }
    return al.matricula;
}
```

Como será o último nó, já recebe NULL

Se Lista vazia, inserção se dá no início

```
//arquivo listaDDupla.h
int insereFinal(LISTA *li, ALUNO al);
```

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

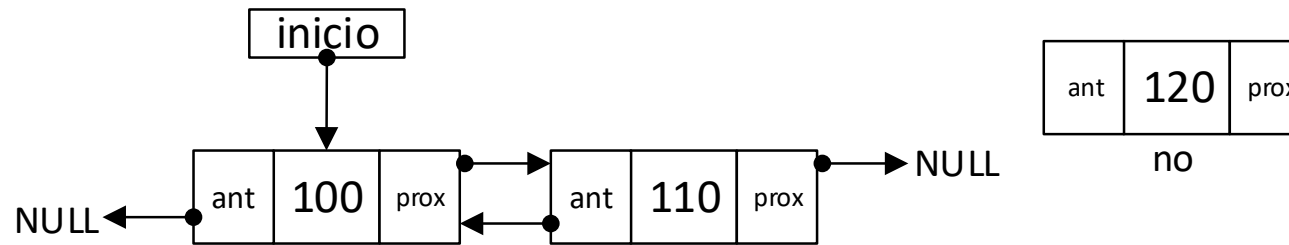
Ponteiro auxiliar para percorrer a Lista. Enquanto o campo prox != NULL, continua se deslocando na Lista.

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

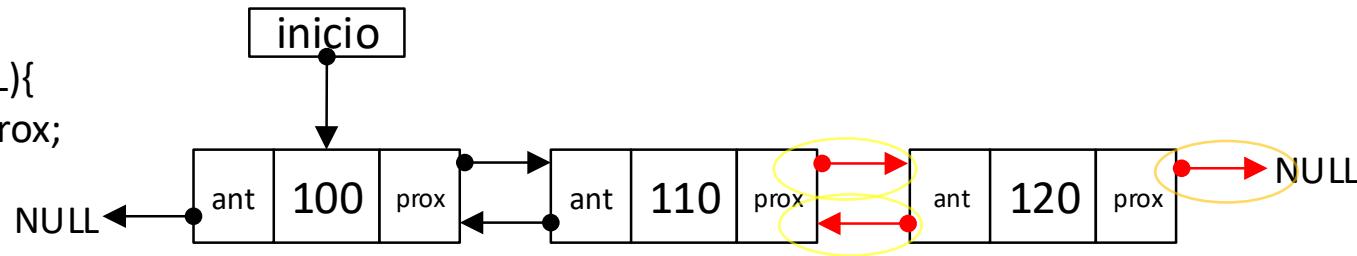
Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada – Inserção no final



Busca onde inserir:

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



Inserir depois de "aux":
no->dados = al;
no->prox = NULL;
aux->prox = no;
no->ant = aux;



Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada – Inserção Ordenada



```
//arquivo listaDDupla.h  
int insereOrdenado(LISTA *li, ALUNO al);
```

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

Lista Dinâmica Duplamente Ligada – Inserção Ordenada

```
//arquivo listaDDupla.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)){
        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;
        }
    }
}
```

Inserir em uma
Lista vazia

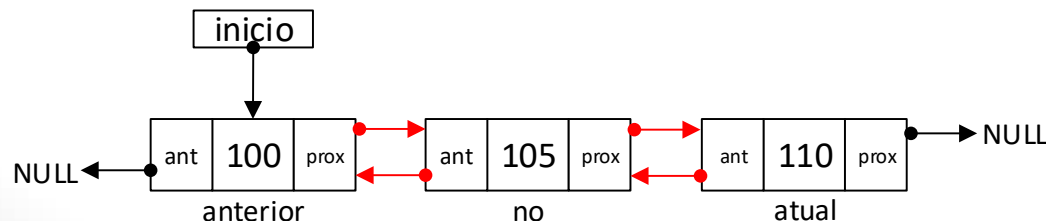
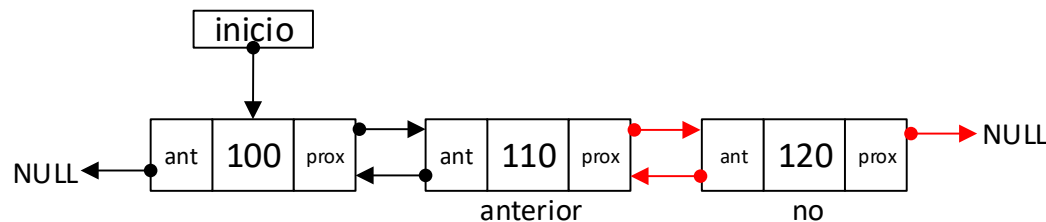
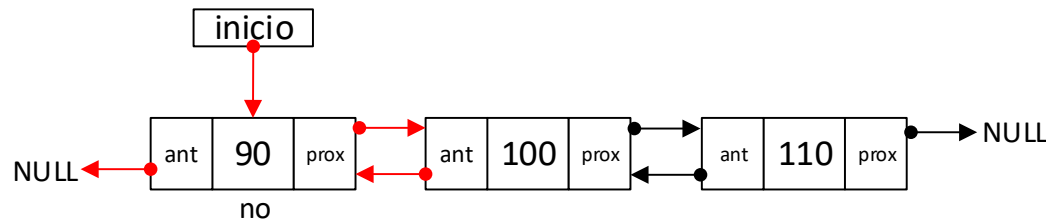
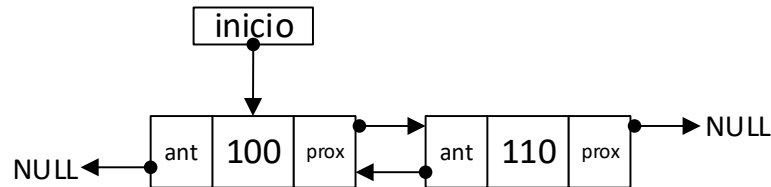
```
if(atual == *li){
    no->ant = NULL;
    (*li)->ant = no;
    no->prox = (*li);
    *li = no;
}
else{
    no->prox = anterior->prox;
    no->ant = anterior;
    anterior->prox = no;
    if(atual != NULL){
        atual->ant = no;
    }
}
return al.matricula;
}
```

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

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;`

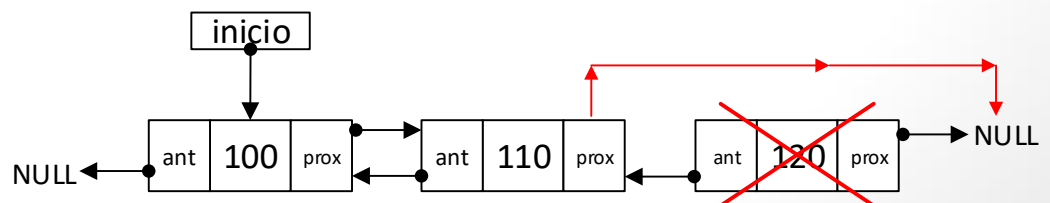
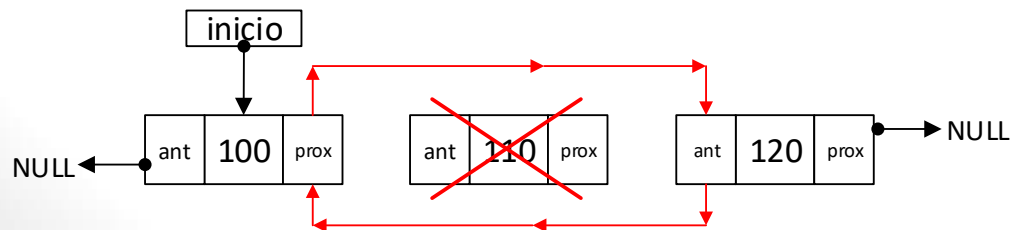
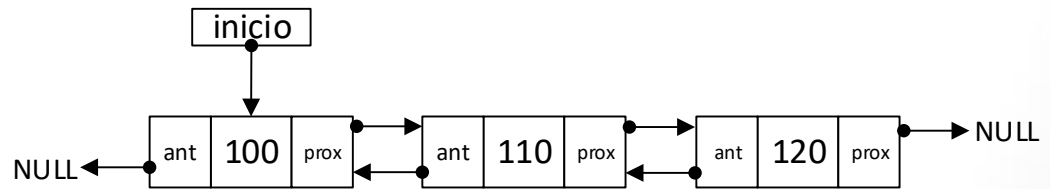
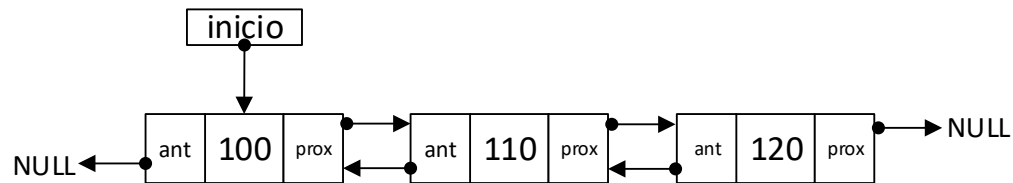
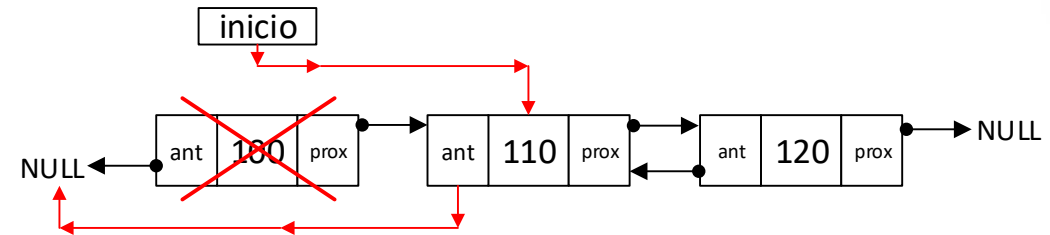
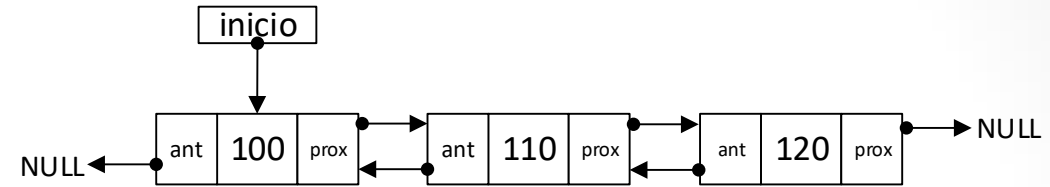
Inserindo quando não é o fim da lista”:

`atual->ant = no;`

Lista Dinâmica Duplamente Ligada – Remoção

- Existem 3 tipos de remoção:

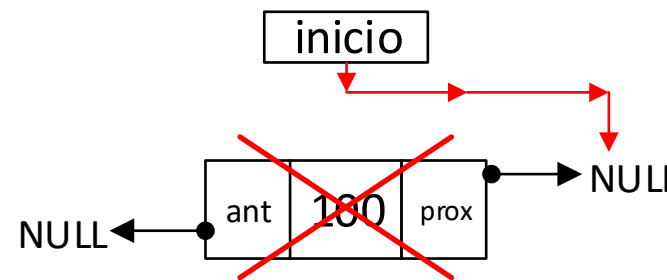
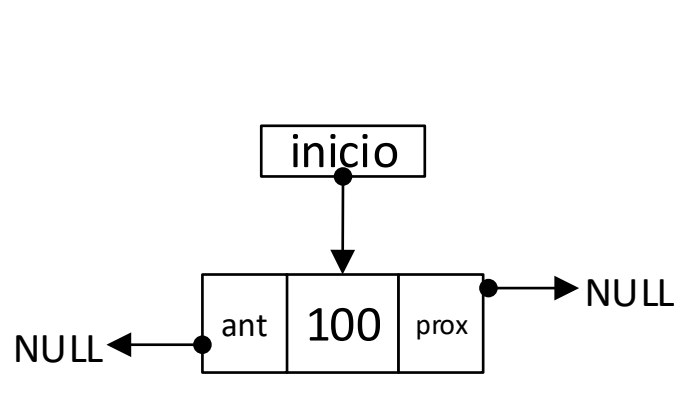
- Início;
- Meio;
- Fim;



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.



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



Lista Dinâmica Duplamente Ligada – Remoção no início

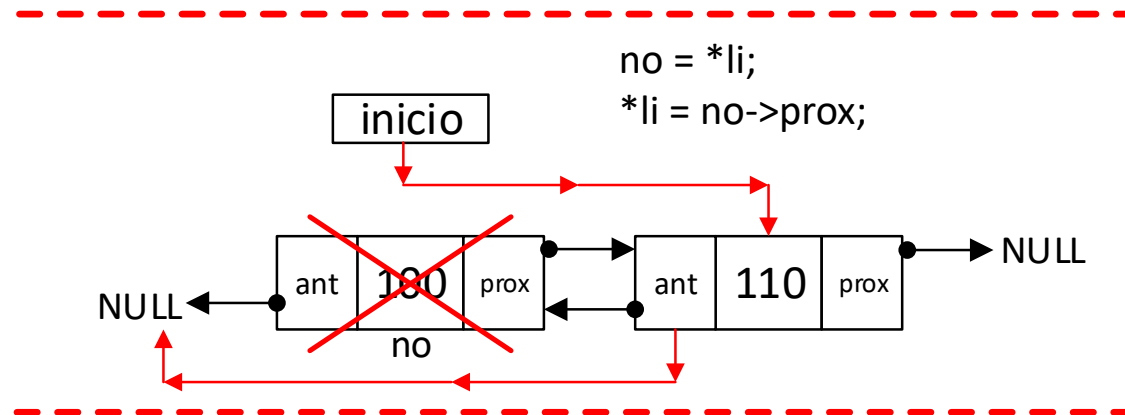
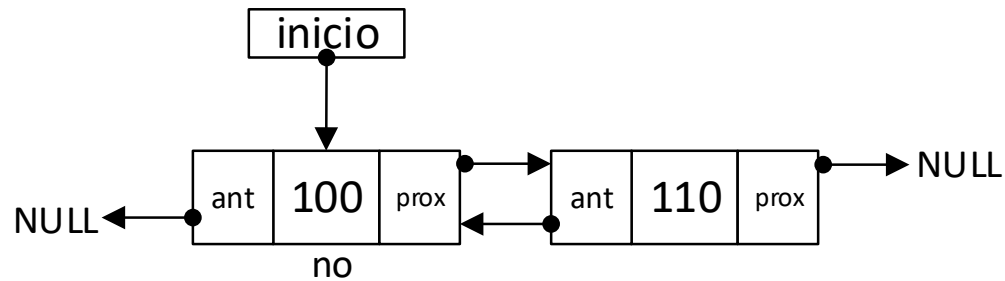
```
//arquivo listaDDupla.h
int removeInicio(LISTA *li);
```

```
//arquivo listaDDupla.c
int removeInicio(LISTA *li){
    int matricula;
    if(li == NULL){
        abortaPrograma();
    }
    if(!listaVazia(li)){
        return 0;
    }
    ELEM *no = *li;
    *li = no->prox;
    if(no->prox != NULL){
        no->prox->ant = NULL;
    }
    matricula = no->dados.matricula;
    free(no);
    return matricula;
}
```

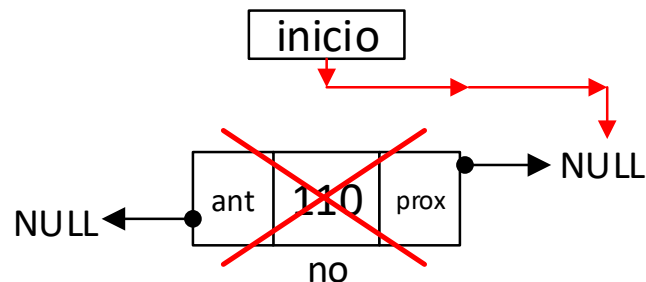
```
//arquivo main.c
x = removeInicio(li);
if(x){
    printf("\nElemento %d removido no inicio com sucesso!", x);
}else{
    printf("\nErro! Elemento nao removido.");
}
```

Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada – Remoção no início



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

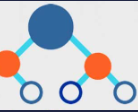


Se “nó” é o único elemento da Lista, a Lista fica vazia.



Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada – Remoção no final



```
//arquivo listaDDupla.h
int removeFinal(LISTA *li);
```

```
//arquivo listaDDupla.c
int removeFinal(LISTA *li){
    int matricula;
    if(li == NULL){
        abortaPrograma();
    }
    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;
    }
    matricula = no->dados.matricula;
    free(no);
    return matricula;
}
```

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

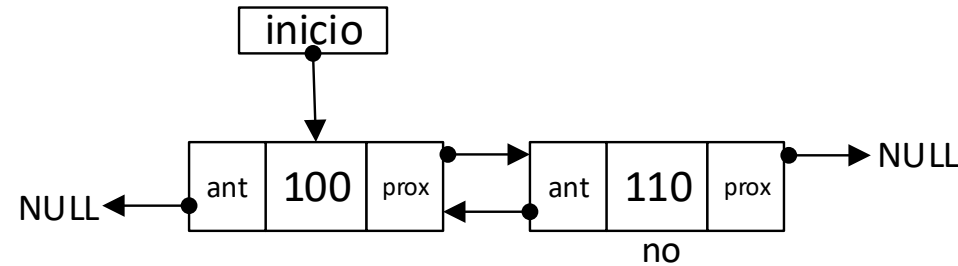
Percorre a Lista até que o campo prox do nó aponte para NULL

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

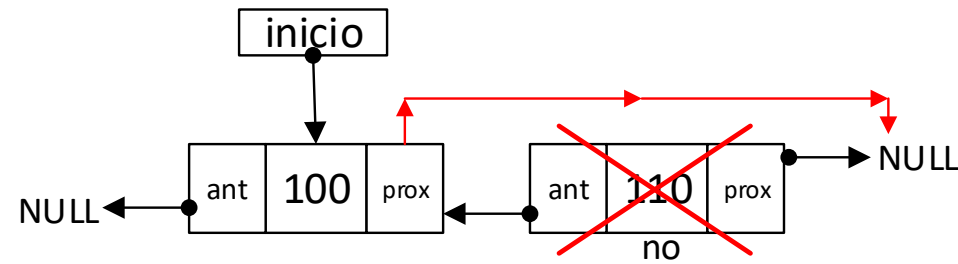
Procura o último elemento da Lista:

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



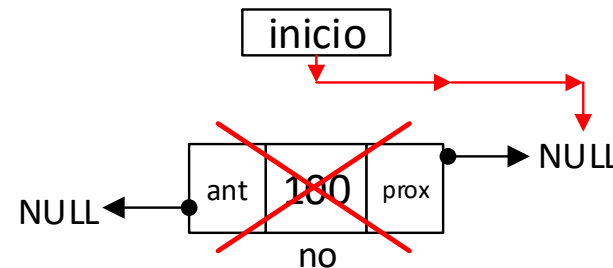
Nó não é o único elemento da Lista:

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



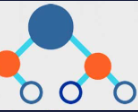
Nó é o único elemento, e a Lista fica vazia:

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



Estrutura de Dados 1

Lista Dinâmica Duplamente Ligada - Remoção de qualquer elemento



```
//arquivo listaDDupla.h
int removeOrdenado(LISTA *li, int mat);
```

```
//arquivo listaDDupla.c
int removeOrdenado(LISTA *li, int mat){
    int matricula;
    if(li == NULL){
        abortaPrograma();
    }
    ELEM *no = *li;
    while(no != NULL && no->dados.matricula != mat){
        no = no->prox;
    }
    if(no == NULL){ //não encontrado - chegou ao final
        return 0; //e não 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
    }
    matricula = no->dados.matricula;
    free(no);
    return matricula;
}
```

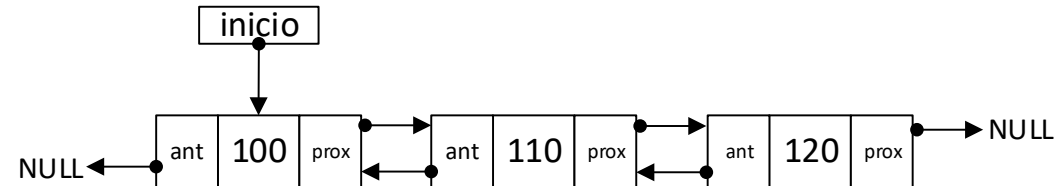
```
//arquivo main.c
x = removeOrdenado(li, matricula);
if(x){
    printf("\nElemento %d removido com sucesso!", x);
}else{
    printf("\nErro! Elemento nao removido.");
}
```


Estrutura de Dados 1

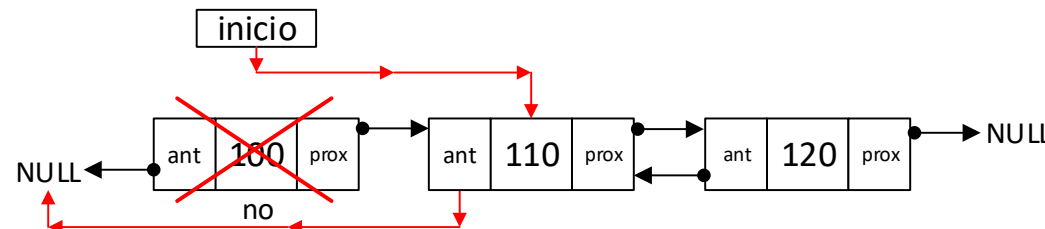


Lista Dinâmica Duplamente Ligada - Remoção de qualquer elemento

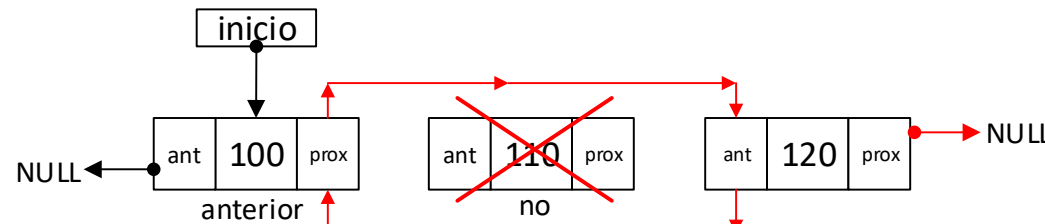
Procura pelo nó a ser removido.



Remover do início:
`*li = no->prox;`



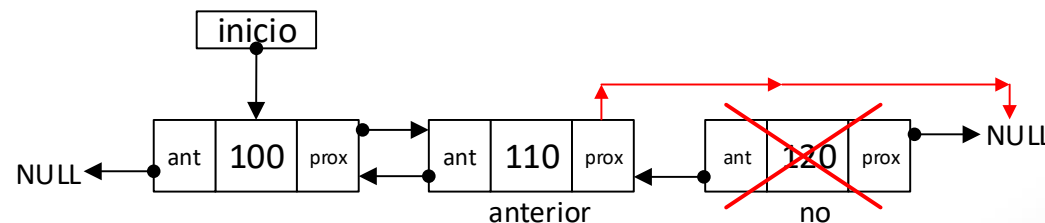
Não está removendo do final:
`no->prox->ant = no->ant;`



Serve para os 2 casos:

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

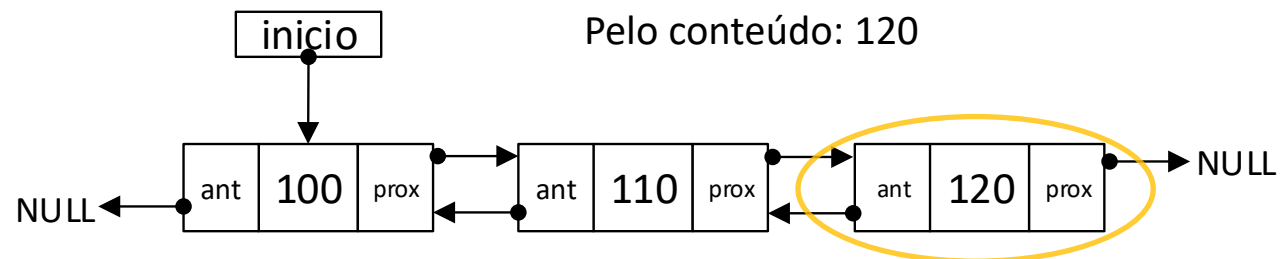
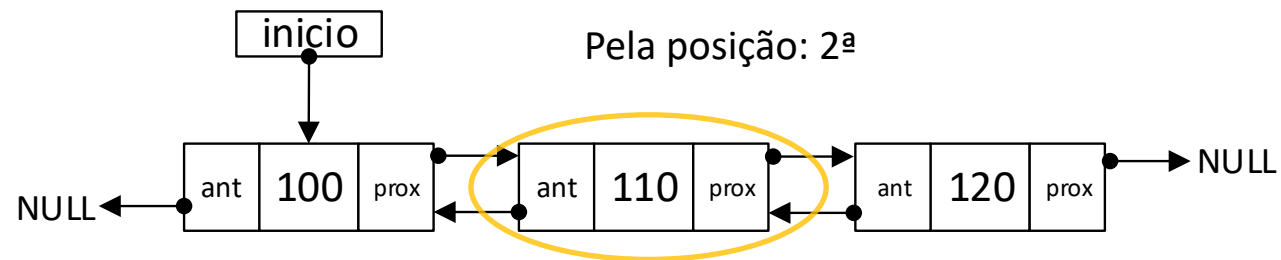
Por último:
`free(no);`



Estrutura de Dados 1

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

```
//arquivo listaDDupla.h
int consultaPosicao(LISTA *li, int pos, ALUNO *al);
```

```
//arquivo listaDDupla.c
int consultaPosicao(LISTA *li, int pos, ALUNO *al){
    if(li == NULL || pos <= 0){
        abortaPrograma();
    }
    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

```
//arquivo main.c
x = consultaPosicao(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

```
//arquivo listaDDupla.h
int consultaMatricula(LISTA *li, int mat, ALUNO *al);
```

```
//arquivo listaDDupla.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 no == NULL,
significa que matrícula
informada não existe

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

```
//arquivo main.c
x = consultaMatricula(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.");
}
```

Estrutura de Dados 1

Atividade

- Entregue todo o projeto compactado na plataforma Moodle.

