



Estruturas de Dados 1

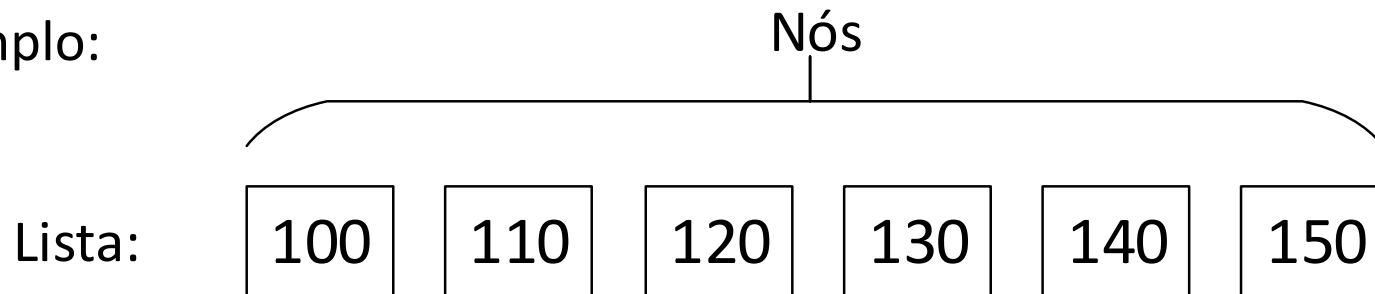
10 – Lista Sequencial Estática

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

Estrutura de Dados 1

Lista Sequencial Estática

- Lista – Sequência de elementos do mesmo tipo. Seus elementos possuem **estrutura interna abstraída**, ou seja, sua **complexidade é arbitrária** e não afeta o funcionamento do seu programa;
- Exemplo:



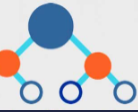
Conjunto de elementos de um mesmo tipo, em uma certa sequencia.

- Para implementar uma lista em seu programa, não é necessário se preocupar como seu TAD é codificado internamente ou como ele armazena os dados. O seu funcionamento será sempre o mesmo.



Estrutura de Dados 1

Lista Sequencial Estática

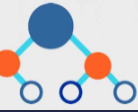


- Uma Lista pode possuir “n” ($n \geq 0$) elementos.
- Se “n = 0”, dizemos que a lista está vazia;
- A quantidade máxima de elementos que uma lista suporta é definido por MAX.
- Existem casos em que:
 - A Lista está vazia ($n = 0$);
 - A Lista tem alguns elementos ($n < \text{MAX}$);
 - A Lista está cheia ($n = \text{MAX}$).

Estrutura de Dados 1

Lista Sequencial Estática

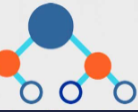
- Aplicações:
 - Cadastro de Funcionários;
 - Itens de um Estoque;
 - Cadastro de Clientes;
 - Lista de Contatos;
 - Lista de pedidos de um restaurante;
 - Etc.
- Em tudo aquilo que se pode colocar em uma forma de lista, podemos utilizar a estrutura Lista.



Estrutura de Dados 1

Lista Sequencial Estática

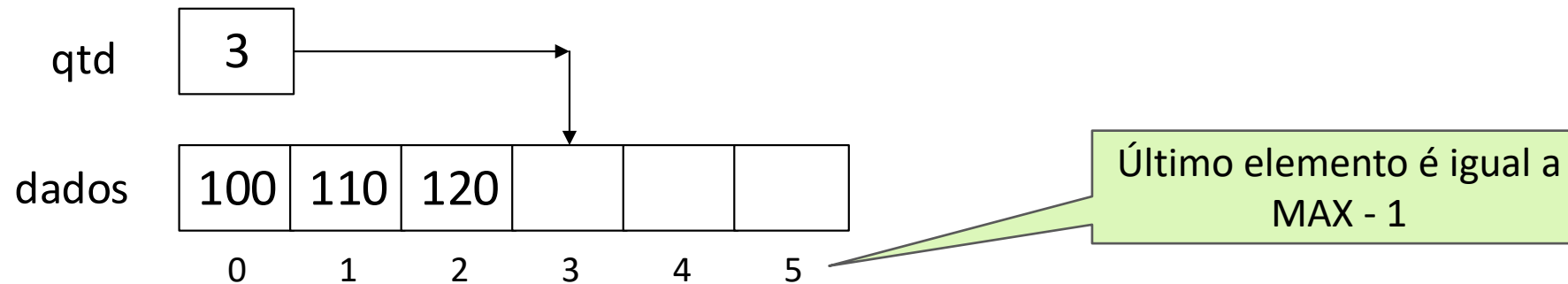
- Em uma Lista podemos realizar as seguintes operações básicas (que são suas funções):
 - Criação da Lista (por ser um TAD);
 - Inserção de um elemento;
 - Exclusão de um Elemento;
 - Acesso a um elemento (recuperar em determinada posição);
 - Destruição da Lista (por ser um TAD);
 - Etc.
- Essas operações dependem do tipo de alocação:
 - Estática;
 - Dinâmica.



Estrutura de Dados 1

Lista Sequencial Estática – Diferenças

- O espaço de memória é alocado no momento da compilação;
- Exige a definição do número máximo de elementos da Lista (MAX);
- Acesso sequencial: os elementos estão de forma consecutiva na memória:



- A lista sequencial estática trabalha com um vetor e um campo “qtd”, que aponta para a próxima posição vaga do vetor da lista. Paralelamente, o campo “qtd” também indica quantos elementos já temos na lista. MAX define a quantidade máxima de elementos na lista.



Estrutura de Dados 1

Lista Sequencial Dinâmica – Diferenças

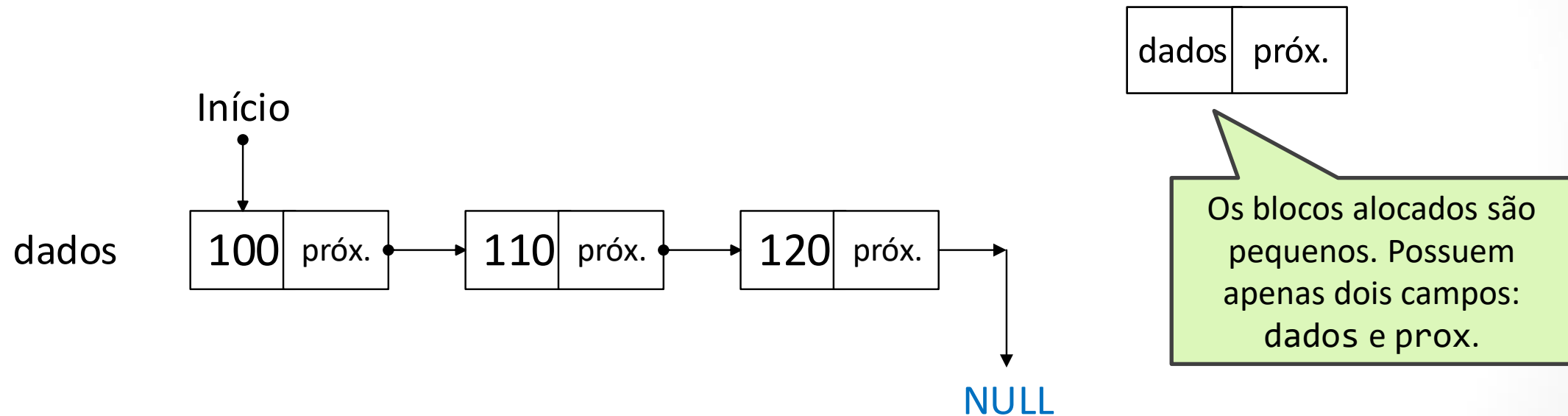
- O espaço de memória é alocado em tempo de execução. Não é necessário definir o tamanho inicial.
- A lista cresce a medida que novos elementos são armazenados e diminui quando estes são removidos.
- Acesso encadeado: cada elemento pode estar em uma área restrita e aleatória da memória.
- Para acessar um determinado elemento, é necessário que se percorra todos os seus antecessores na lista.



Estrutura de Dados 1

Lista Sequencial Dinâmica – Diferenças

- Os nós são formados por dois campos: um contém os dados e o outro um ponteiro que aponta para o próximo nó. Exemplo:



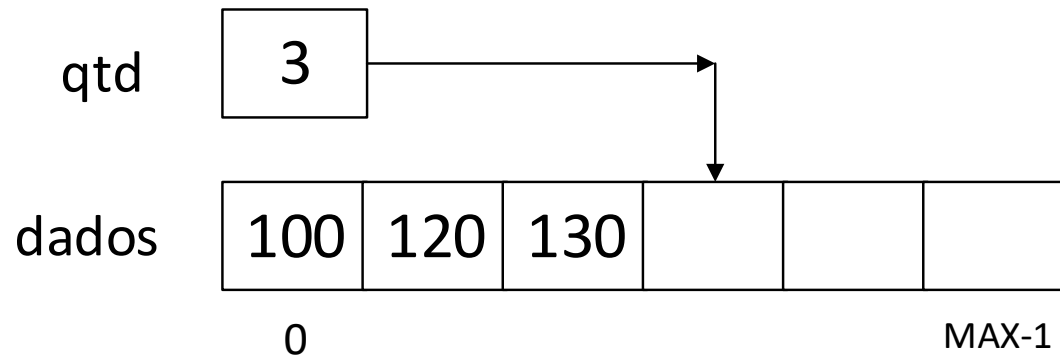
- Não existe um índice para acesso aos elementos em posições determinadas. Por exemplo, para acessarmos o elemento 120 da lista acima, temos que acessar antes, primeiro os elementos 100 e 110.



Estrutura de Dados 1

Lista Sequencial Estática - Implementação

- “Lista Sequencial Estática” ou “Lista Linear Estática” – Tipo de lista onde o sucessor de um elemento ocupa a posição física seguinte do mesmo, para isso é utilizado um vetor (ou *array*).



Estrutura da Lista:

Um *array* ou vetor de tamanho `MAX`, definido antecipadamente.

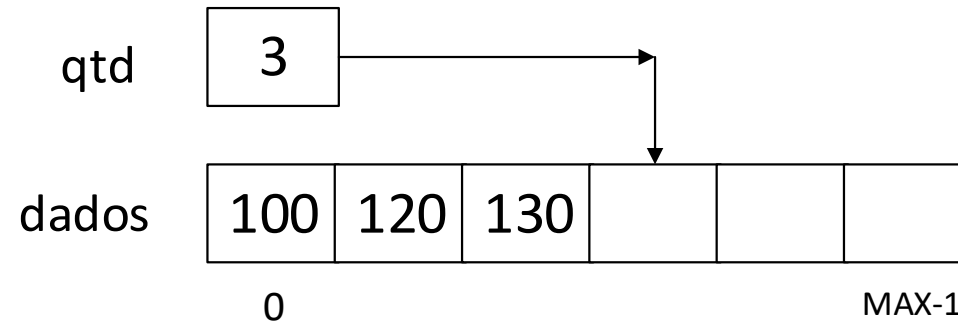
- O campo “`qtd`” indica quantos elementos temos na lista e ao mesmo tempo indica qual a próxima posição vaga.



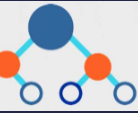
Estrutura de Dados 1

Lista Sequencial Estática - Implementação

- Vantagens do uso de *arrays* :
 - Acesso rápido e direto aos elementos através do índice;
 - Tempo constante para acessar um elemento;
 - Facilidade em modificar informações.



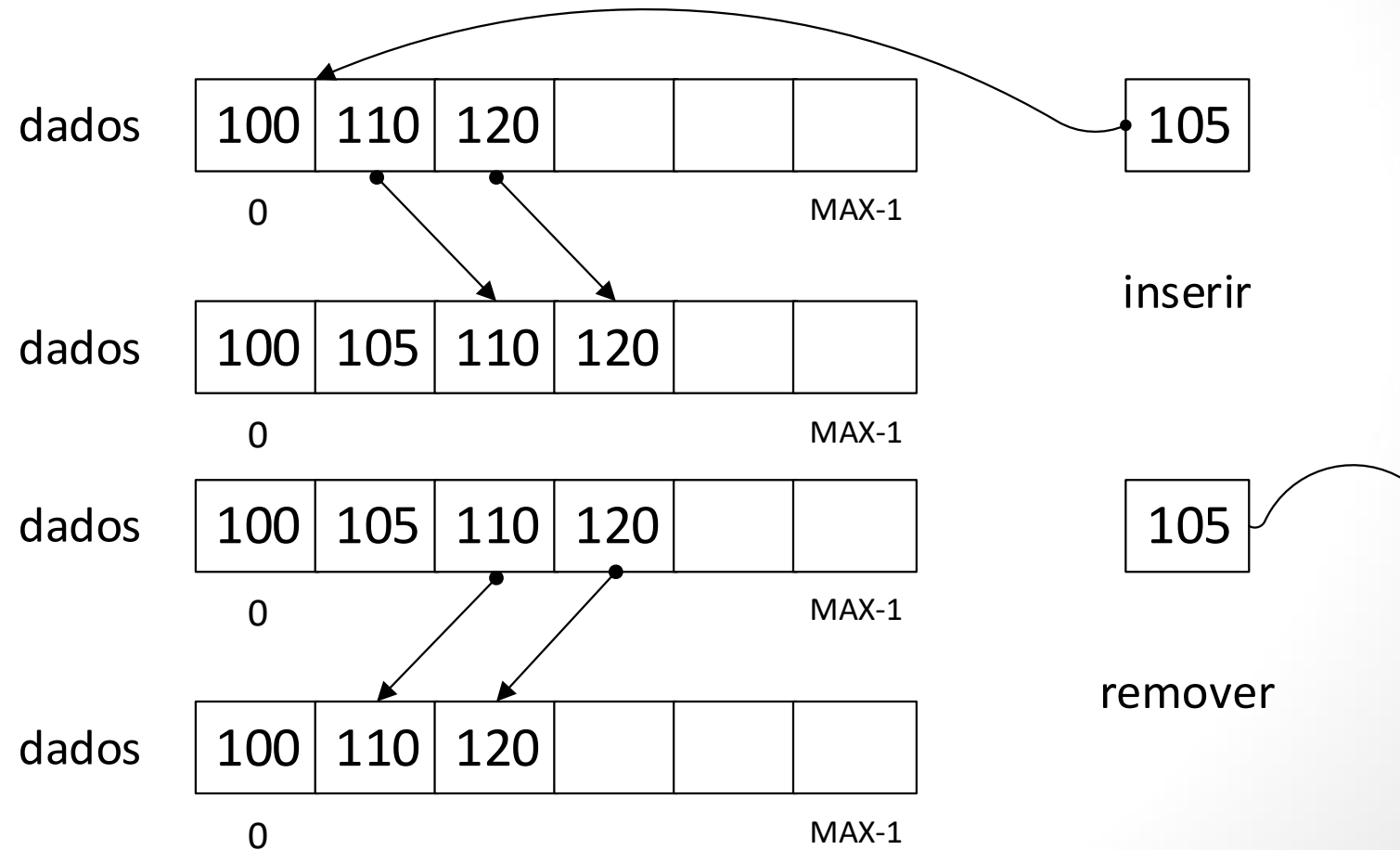
- Desvantagens do uso de *arrays*:
 - Definição prévia do tamanho do *array*;
 - Dificuldade para **inserir** e **remover** um elemento entre outros dois elementos:
 - É necessário deslocar os elementos.



Estrutura de Dados 1

Lista Sequencial Estática - Implementação

A utilização de *arrays* ou vetores envolve deslocamento em operações de inserção e remoção entre elementos. O custo computacional é alto neste tipo de lista.



Estrutura de Dados 1

Lista Sequencial Estática - Implementação



- Quando utilizar esse tipo de lista?
 - Listas pequenas – é o tipo de lista mais fácil e prática de implementar;
 - Quando a inserção e remoção se dá apenas no final da lista e não há deslocamento;
 - Quando temos um tamanho máximo bem definido, ou seja, há a certeza de que tamanho **MAX** nunca será ultrapassado;
 - Quando a busca é a operação mais frequente, e neste caso seria uma lista de consulta – ela é rápida e o tempo de acesso é sempre o mesmo.

Estrutura de Dados 1

Lista Sequencial Estática - implementação

- Será usado o paradigma da modularização com dois arquivos: “.h” e “.c”.
- listaSequencial.h – Serão definidos:
 - Os protótipos das funções que manipulam o TAD;
 - O tipo de dado que será armazenado na lista (estrutura com os dados de um aluno);
 - O ponteiro “Lista”, que será devolvido pela função de criação da lista;
 - A constante MAX que define tamanho do vetor utilizado na lista.
- listaSequencial.c – Definir:
 - O tipo de dados “Lista”;
 - Implementar as suas funções de manipulação dos dados.

Este “tipo” fica oculto do programa principal através do encapsulamento. O usuário/programador só o acessa por meio das funções disponibilizadas pelo TAD



Estrutura de Dados 1

Lista Sequencial Estática – implementação

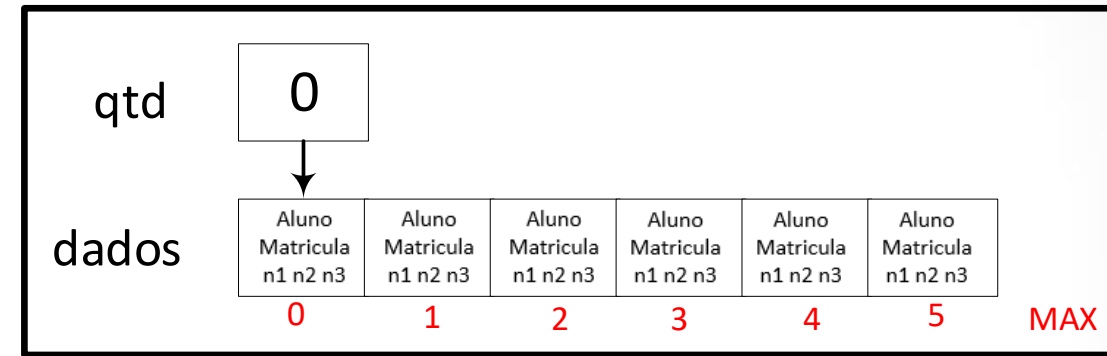
```
//Arquivo listaSequencial.h  
#define MAX 100
```

Dado que será
guardado
dentro da Lista

```
struct aluno{  
    int matricula;  
    float n1, n2, n3;  
};
```

```
typedef struct lista Lista;
```

Li



```
//Arquivo principal - main()  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include "listaSequencial.h"
```

```
int main()  
{  
    int x; //será usado para código de erro  
    Lista *li;
```

Tipo de dado
opaco que fica
oculto, e é
manipulado pelo
encapsulamento

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

```
struct lista{  
    int qtd;  
    struct aluno dados[MAX];  
};
```



Estrutura de Dados 1

Lista Sequencial Estática – implementação

- Função para criar a Lista:

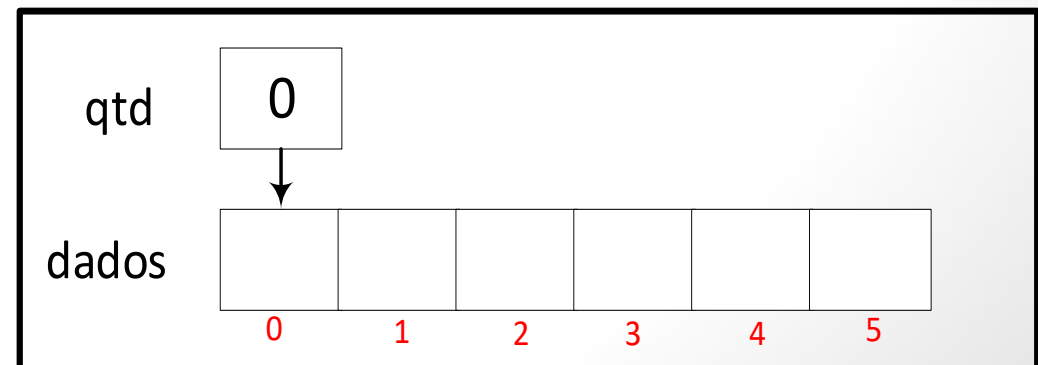
```
//Arquivo listaSequencial.h  
Lista *cria_lista();
```

Retorna somente
endereços de memória,
ou seja, ponteiros.

```
//Arquivo listaSequencial.c  
Lista *cria_lista(){  
    Lista *li;  
    li = (Lista*) malloc(sizeof(struct lista));  
    if(li != NULL){  
        li->qtd = 0;  
    }  
    return li;  
}
```

```
//programa principal main()  
Lista *li;  
li = cria_lista();
```

Lista *Li



Estrutura de Dados 1

Lista Sequencial Estática – implementação

- Função para destruir (ou liberar) a Lista:

```
//Arquivo listaSequencial.h  
void libera_lista(Lista *li);
```

```
//Arquivo listaSequencial.c  
void libera_lista(Lista *li){  
    free(li);  
}
```

```
//programa principal main()  
libera_lista(li);
```

Atenção!!!
Esta deve ser a última função
chamada pelo main()!

Se é só para liberar memória através de um simples `free()`, porque não fazê-lo no programa principal?

Liberar uma lista sequencial é muito simples, mas em uma lista dinâmica, um único `free()`, não libera a lista. Será preciso liberar todos os nós, que compõe a lista. Então cria-se a função, para efeitos de compatibilidade entre os tipos de Lista, estática e dinâmica, uma vez que as duas terão módulos que poderão ser intercambiáveis.

O usuário/programador não precisa saber como o código foi implementado internamente, se é estático ou se é dinâmico, somente precisa saber usar o TAD propriamente dito.



Estrutura de Dados 1

Lista Sequencial Estática – implementação

- Como obter informações básicas sobre a Lista Estática?
 - Tamanho – quantos elementos já possui?
 - Está cheia?
 - Está vazia?
- Qual a importância desses dados?
 - Se está cheia: não é possível colocar mais nenhum elemento dentro da lista, só retirá-los;
 - Se está vazia: não é possível retirar nenhum elemento, só adicioná-los;
 - Tamanho: Fornece uma ideia de quanto falta para a lista estar cheia e a noção de quantos elementos ela possui.



Estrutura de Dados 1

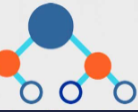
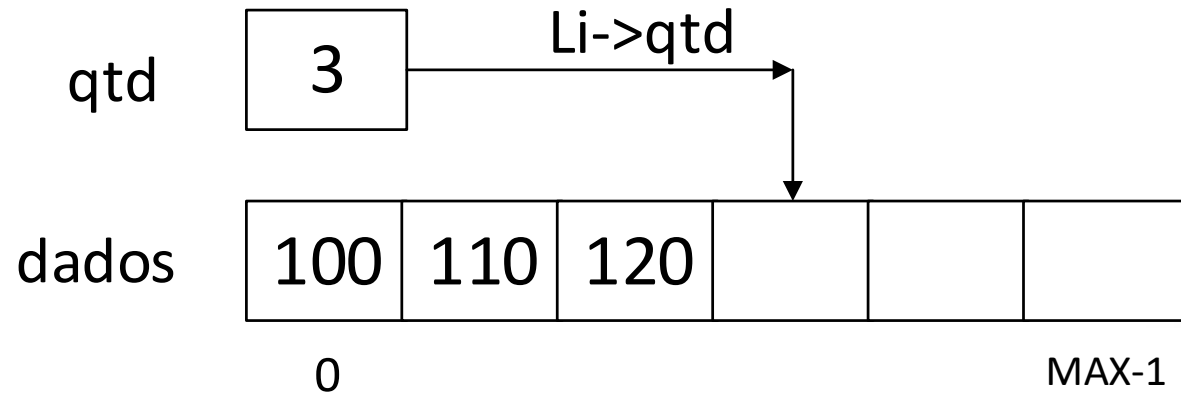
Lista Sequencial Estática – implementação

- Tamanho da lista:

```
//Arquivo listaSequencial.h  
int tamanho_lista(Lista *li);
```

```
//Arquivo listaSequencial.c  
int tamanho_lista(Lista *li){  
    if(li == NULL){  
        return -1;  
    }else{  
        return li->qtd;  
    }  
}
```

```
//programa principal main()  
x = tamanho_lista(li);  
printf("\nTamanho da lista e: %d", x);
```



Estrutura de Dados 1

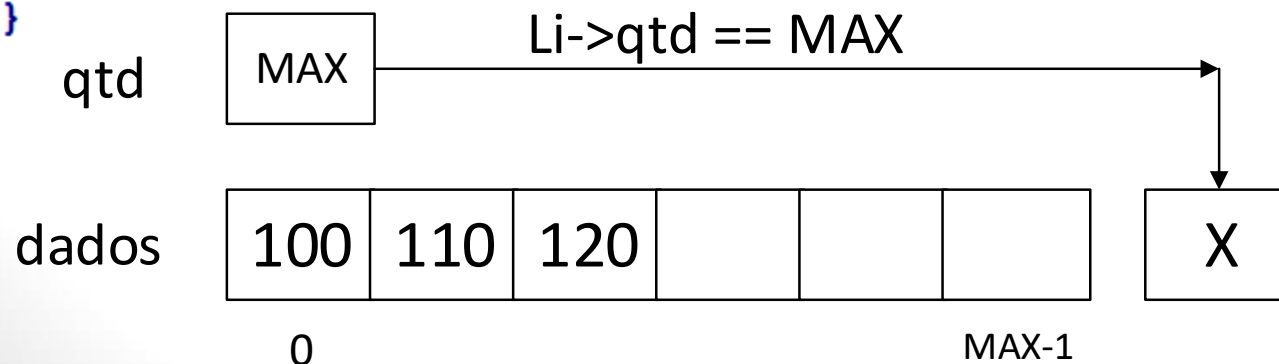
Lista Sequencial Estática – implementação

- Verifica se Lista está cheia:

```
//Arquivo listaSequencial.h  
int lista_cheia(Lista *li);
```

```
//Arquivo listaSequencial.c  
int lista_cheia(Lista *li){  
    if(li == NULL){  
        return -1;  
    }else{  
        return (li->qtd == MAX);  
    }  
}
```

```
//programa principal main()  
x = lista_cheia(li);  
if(x){  
    printf("\nLista cheia!");  
}else{  
    printf("\nLista nao esta cheia!");  
}
```



Esta expressão retorna o resultado:
Se for verdadeira : 1
Se for falsa: 0



Estrutura de Dados 1

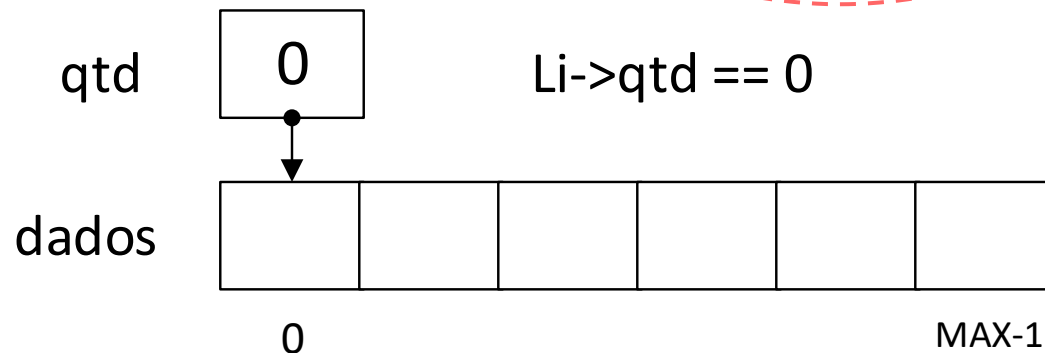
Lista Sequencial Estática – implementação

- Verificar se Lista está vazia:

```
//Arquivo listaSequencial.h  
int lista_vazia(Lista *li);
```

```
//Arquivo listaSequencial.c  
int lista_vazia(Lista *li){  
    if(li == NULL){  
        return -1;  
    }else{  
        return (li->qtd == 0);  
    }  
}
```

```
//programa principal main()  
x = lista_vazia(li);  
if(x){  
    printf("\nLista esta vazia!");  
}else{  
    printf("\nLista nao esta vazia!");  
}
```



Esta expressão retorna o resultado:
Se for verdadeira : 1
Se for falsa: 0



Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Existem três tipos de inserção em Listas Sequenciais Estáticas:

- Início;
- Meio;
- Fim.

Quando inserir no meio da Lista?

Quando estamos inserindo de forma ordenada.

dados	100	110	120			
	0					MAX-1

Início dados	90	100	110	120		
	0					MAX-1

Final dados	110	120	130	140		
	0					MAX-1

Meio dados	100	105	110	120		
	0					MAX-1



Estrutura de Dados 1

- Dados para inserções e buscas:

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

Será usado na busca por um elemento específico

Será usado na busca por um elemento em uma determinada posição

```
struct aluno al, al2, al3, dados_aluno;
```

Matricula: 100
n1: 5.3
n2: 6.9
n3: 7.4

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

Será usado na inserção no início

Matricula: 120
n1: 4.0
n2: 2.9
n3: 8.4

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

Será usado na inserção no final

Matricula: 110
n1: 1.3
n2: 2.9
n3: 3.4

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

Será usado na inserção ordenada



Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Inserção no final da lista:

```
//Arquivo listaSequencial.h  
int insere_lista_final(Lista *li, struct aluno al);
```

```
//Arquivo listaSequencial.c  
int insere_lista_final(Lista *li, struct aluno al){  
    if(li == NULL){  
        return 0;  
    }  
    if(lista_cheia(li)){  
        return 0;  
    }  
    li->dados[li->qtd] = al;  
    li->qtd++;  
    return 1;  
}
```

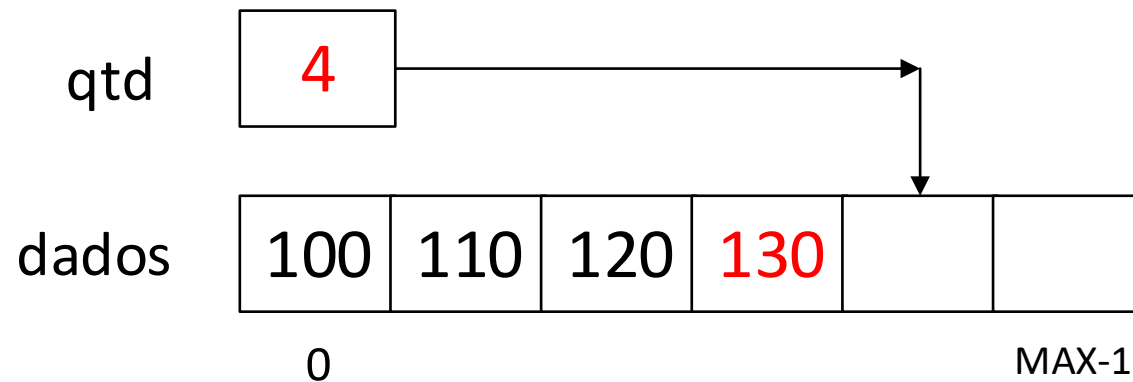
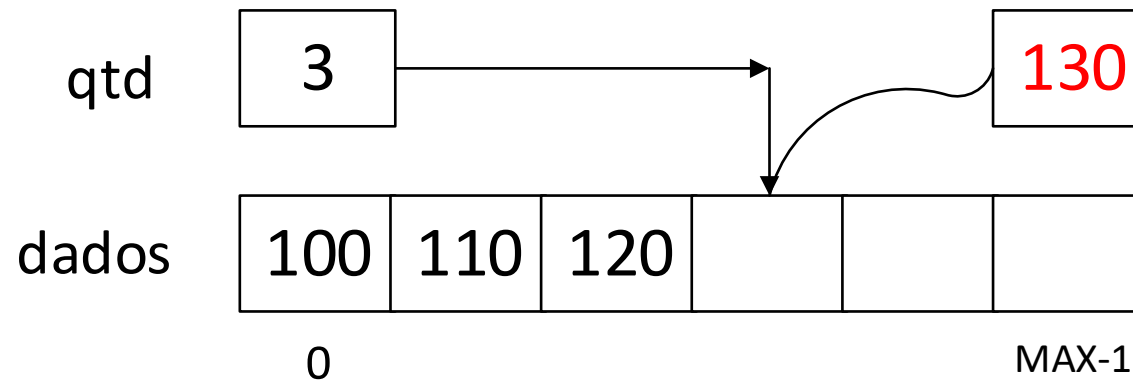
```
//programa principal main()  
x = insere_lista_final(li, al2);  
if(x){  
    printf("\nAluno inserido com sucesso!");  
}else{  
    printf("\nErro aluno nao inserido!");  
}
```



Estrutura de Dados 1

Lista Sequencial Estática – inserção

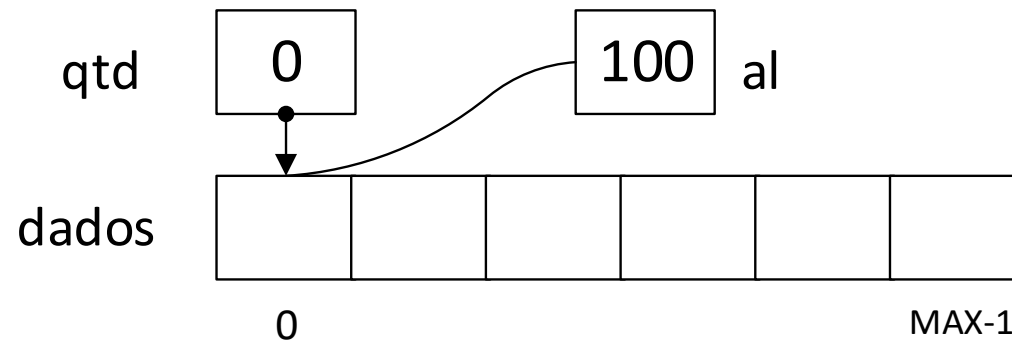
- Inserção no final da lista:



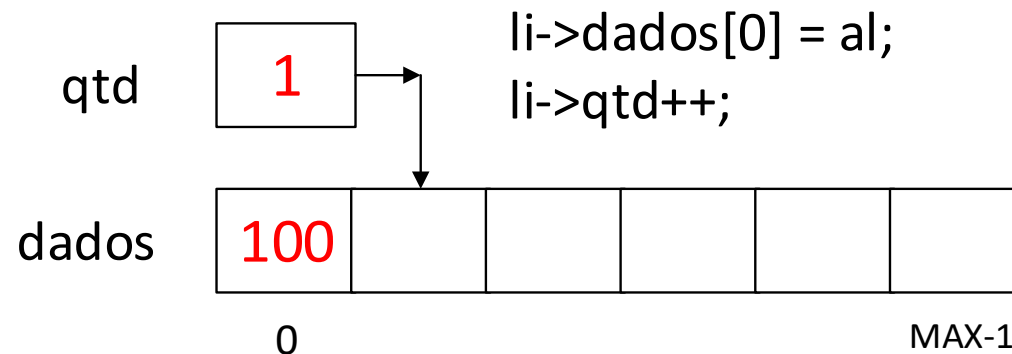
Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Existe também o caso onde a inserção é feita em uma lista vazia:



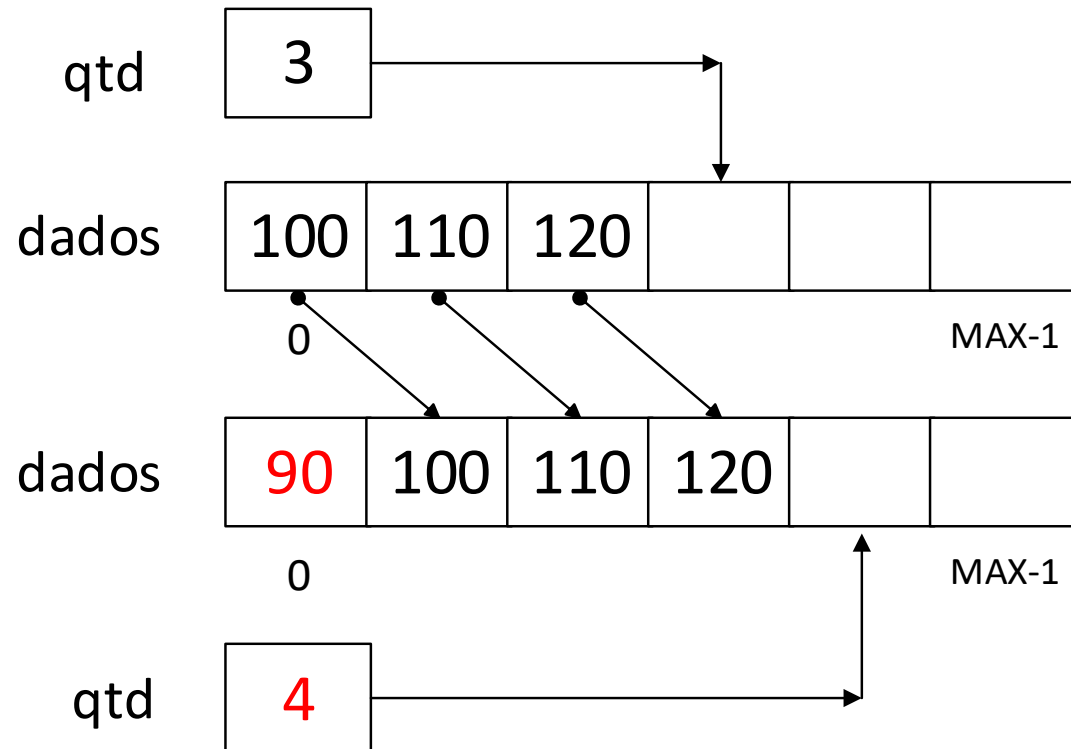
Cuidado!
Não se pode inserir em
uma lista cheia.



Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Inserção no início da lista:



Ordem de movimentação dos dados:
De traz para frente para não
ocasionar perda de dados. Se não for
assim os dados serão sobrescritos

Desloca-se todos os elementos uma
posição para a frente, e, insere-se o
elemento no início.



Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Inserção no início da lista:

```
//Arquivo listaSequencial.h
```

```
int insere_lista_inicio(Lista *li, struct aluno al);
```

```
//Arquivo listaSequencial.c
```

```
int insere_lista_inicio(Lista *li, struct aluno al){
```

```
    if(li == NULL){
```

```
        return 0;
```

```
    }
```

```
    if(lista_cheia(li)){
```

```
        return 0;
```

```
    }
```

```
    int i;
```

```
    for(i = li->qtd - 1; i >= 0; i--){
```

```
        li->dados[i+1] = li->dados[i];
```

```
    }
```

```
    li->dados[0] = al;
```

```
    li->qtd++;
```

```
    return 1;
```

```
}
```

1ª posição vazia disponível

Inserir al no início da lista

```
//programa principal main()
```

```
x = insere_lista_inicio(li, al);
```

```
if(x){
```

```
    printf("\nAluno inserido com sucesso!");
```

```
}else{
```

```
    printf("\nErro aluno não inserido!");
```

```
}
```

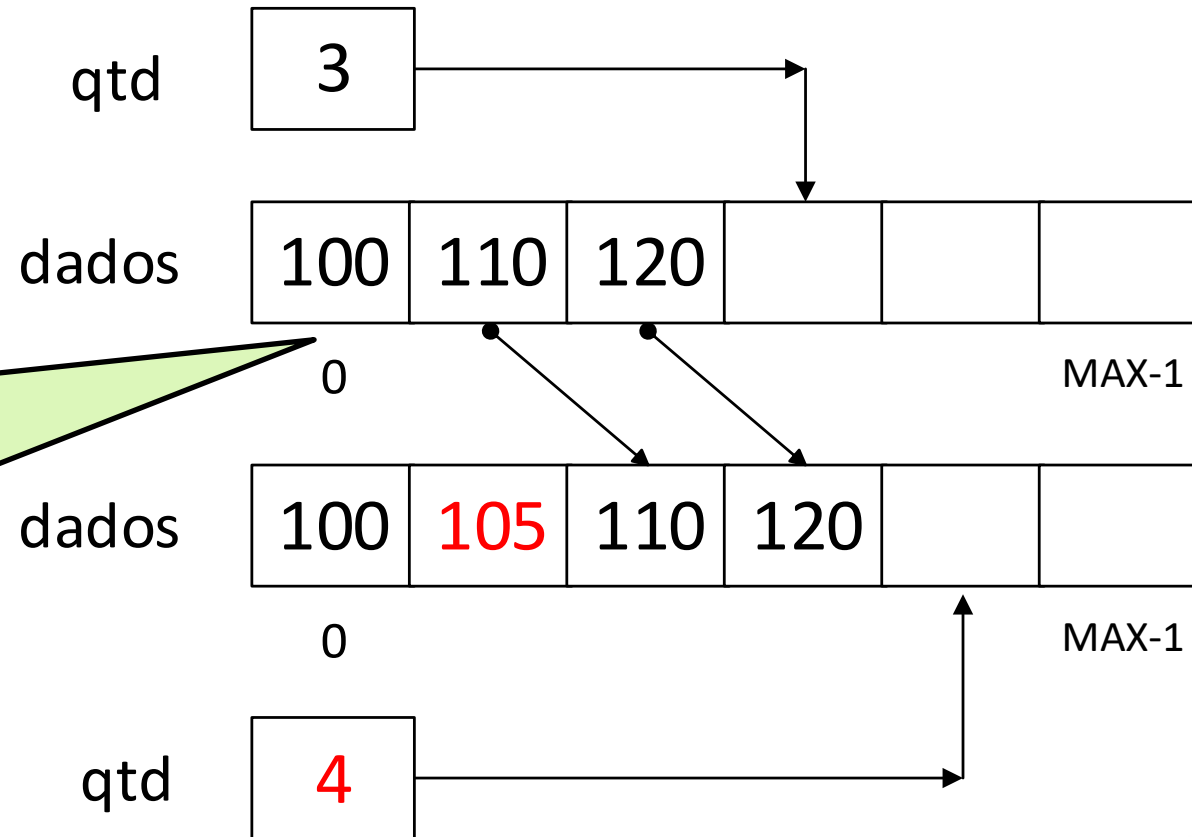
li->dados[i+1] posição a frente do último elemento que está na lista, recebe o último elemento da lista



Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Inserção de forma ordenada:



Estrutura de Dados 1

Lista Sequencial Estática – inserção

- Inserção de forma ordenada:

```
//Arquivo listaSequencial.h  
int insere_lista_ordenada(Lista *li, struct aluno al);
```

```
//programa principal main()  
x = insere_lista_ordenada(li, al3);  
if(x){  
    printf("\nAluno inserido na posicao com sucesso!");  
}else{  
    printf("\nErro aluno nao inserido!");  
}
```



Estrutura de Dados 1

Lista Sequencial Estática – inserção

```
//Arquivo listaSequencial.c
int insere_lista_ordenada(Lista *li, struct aluno al){
    if(li == NULL){
        return 0;
    }
    if(lista_cheia(li)){
        return 0;
    }
    int k, i = 0;
    while(i < li->qtd && li->dados[i].matricula < al.matricula){
        i++;
    }
    for(k = li->qtd - 1; k >= i; k--){
        li->dados[k+1] = li->dados[k];
    }
    li->dados[i] = al;
    li->qtd++;
    return 1;
}
```

Enquanto “i” for menor que a quantidade de elementos que estão na lista, e...

... as matrículas que estão na lista forem menores do que a que se quer inserir, incrementa-se i

Este “for”, executa o deslocamento para frente, dos elementos que ficarão após o elemento a inserir

Dado inserido na posição “encontrada” no laço while

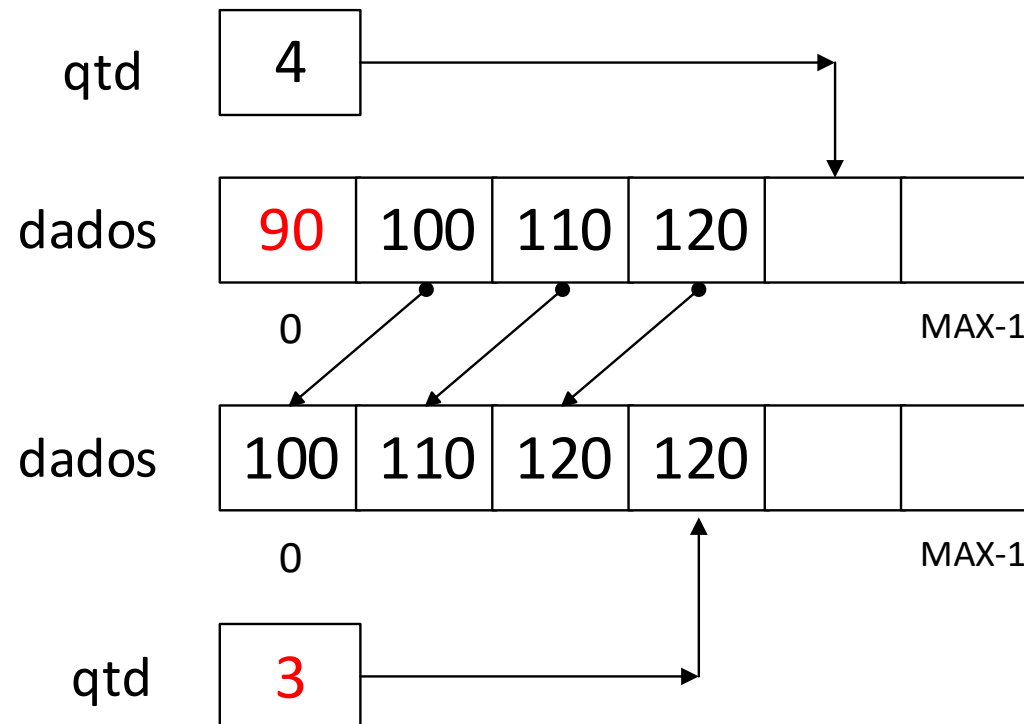


Estrutura de Dados 1

Lista Sequencial Estática – remoção

- Existem três tipos de remoção:

- Início;
- Meio;
- Fim.

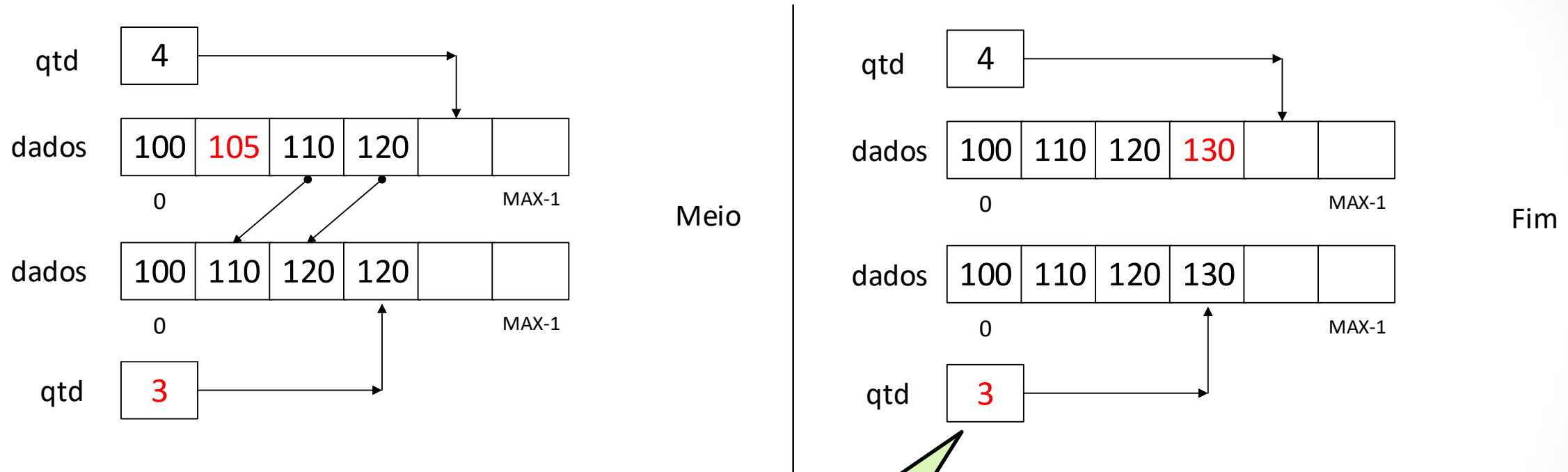


Início



Estrutura de Dados 1

Lista Sequencial Estática – remoção



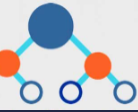
Não há deslocamento, somente é preciso decrementar o campo "qtd"



Estrutura de Dados 1

Lista Sequencial Estática – remoção

- Os três tipos de remoção trabalham juntos. A remoção sempre remove apenas um elemento específico da lista, que pode estar no início, no meio ou no final.
- Cuidado! Não se pode remover elementos de uma lista vazia.



Estrutura de Dados 1

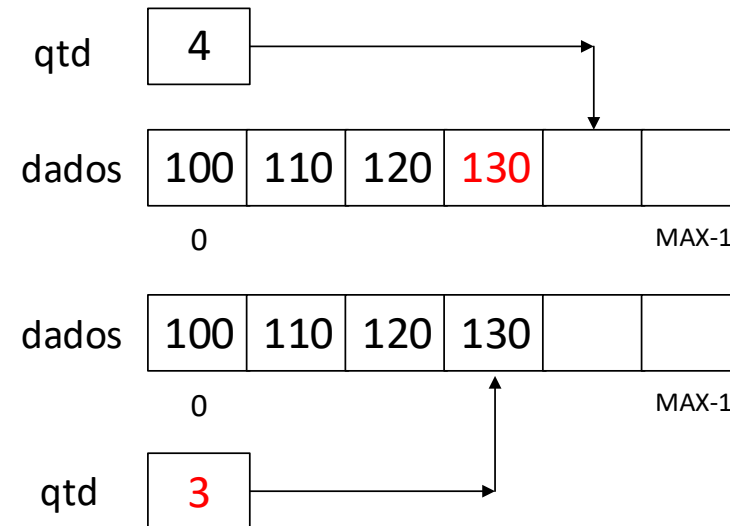
Lista Sequencial Estática – remoção

- Remoção no final:

```
//Arquivo listaSequencial.h  
int remove_lista_inicio(Lista *li);
```

```
//Arquivo listaSequencial.c  
int remove_lista_final(Lista *li){  
    if(li == NULL){  
        return 0;  
    }  
    if(li->qtd == 0){  
        return 0;  
    }  
    li->qtd--;  
    return 1;  
}
```

```
//programa principal main()  
x = remove_lista_final(li);  
if(x){  
    printf("\nAluno removido no final com sucesso!");  
}else{  
    printf("\nErro aluno nao removido!");  
}
```



Fim



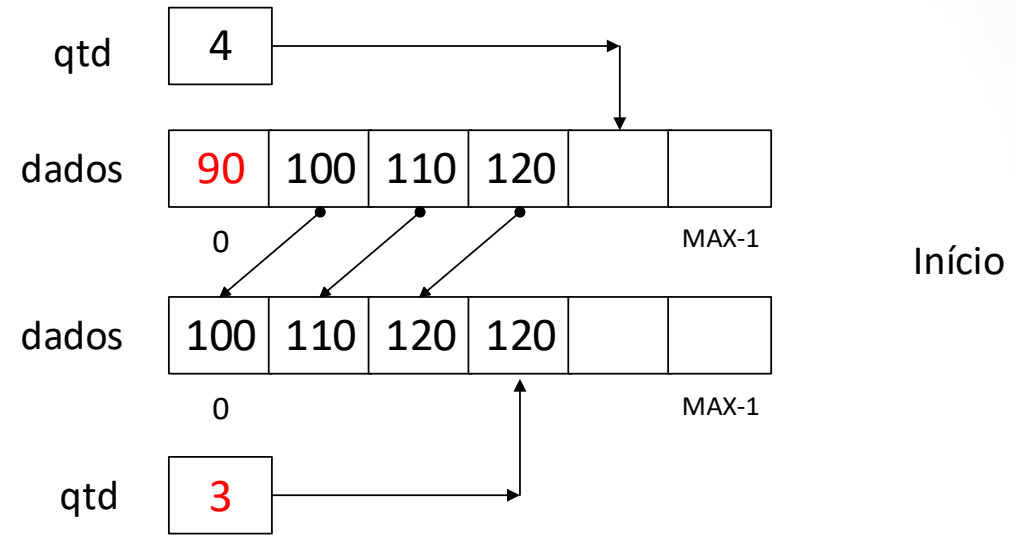
Estrutura de Dados 1

Lista Sequencial Estática – remoção

- Remoção no início (com deslocamento):

```
//Arquivo listaSequencial.h  
int remove_lista_inicio(Lista *li);
```

```
//Arquivo listaSequencial.c  
int remove_lista_inicio(Lista *li){  
    if(li == NULL){  
        return 0;  
    }  
    if(li->qtd == 0){  
        return 0;  
    }  
    int k = 0;  
    for(k = 0; k < li->qtd-1; k++){  
        li->dados[k] = li->dados[k+1];  
    }  
    li->qtd--;  
    return 1;  
}
```



```
//programa principal main()  
x = remove_lista_inicio(li);  
if(x){  
    printf("\nAluno removido do inicio!");  
}else{  
    printf("\nErro aluno nao removido!");  
}
```



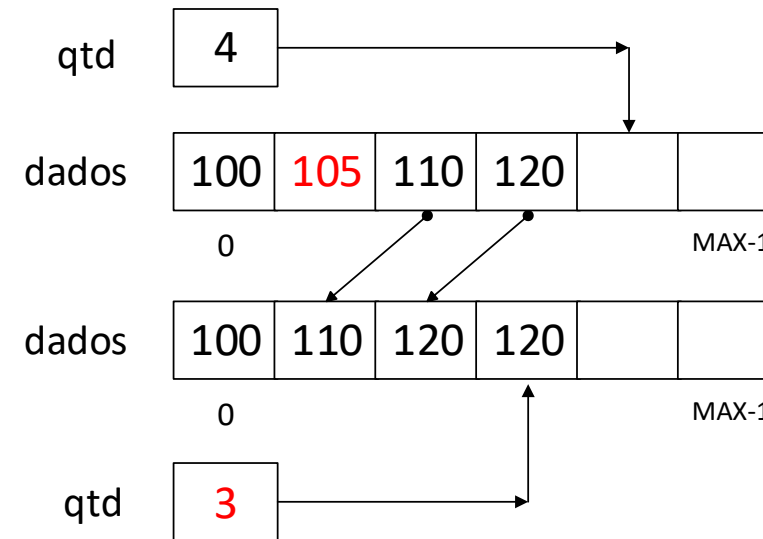
Estrutura de Dados 1

Lista Sequencial Estática – remoção

- Remoção de um elemento qualquer (com deslocamento):

```
//Arquivo listaSequencial.h  
int remove_lista(Lista *li, int matricula);
```

```
//programa principal main()  
x = remove_lista(li, mat);  
if(x){  
    printf("\nAluno removido na posicao especifica com sucesso!");  
}else{  
    printf("\nErro aluno nao removido na posicao especifica!");  
}
```



Meio

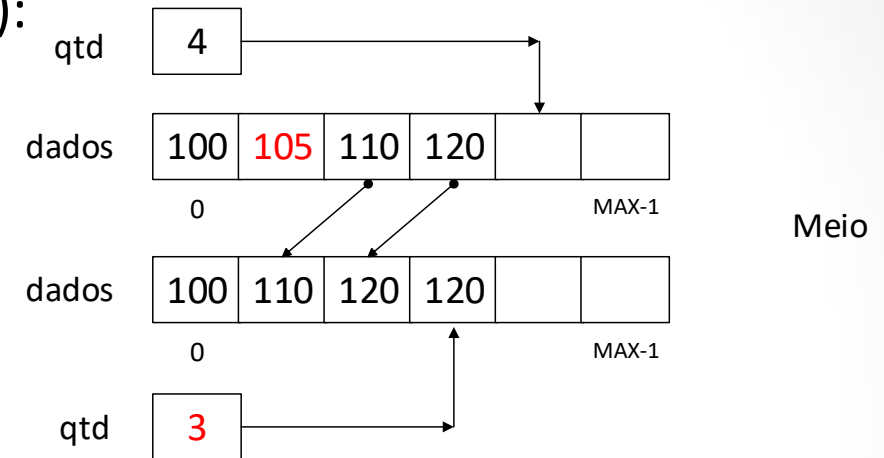


Estrutura de Dados 1

Lista Sequencial Estática – remoção

- Remoção de um elemento qualquer (com deslocamento):

```
//Arquivo listaSequencial.c
int remove_lista(Lista *li, int matricula){
    if(li == NULL){
        return 0;
    }
    if(li->qtd == 0){
        return 0;
    }
    int k, i = 0;
    while(i < li->qtd && li->dados[i].matricula != matricula){
        i++;
    }
    if(i == li->qtd){ //elemento não encontrado!
        return 0;
    }
    for(k = i; k < li->qtd-1; k++){
        li->dados[k] = li->dados[k+1];
    }
    li->qtd--;
    return 1;
}
```



Se o elemento não for encontrado (caso em que i deixará de ser menor do que o campo "qtd"), significa que a chegamos ao final da lista e " i " será igual ao campo "qtd", deixando de satisfazer a condição para que o laço "while" continue a executar seus comandos.



Estrutura de Dados 1

Lista Sequencial Estática – consulta

- Existem duas maneiras de consultar um elemento de uma lista:
 - Pela posição – acesso direto através do índice;
 - Pelo conteúdo – necessidade de busca, comparando o conteúdo campo matrícula.

Posição – 3º

dados	100	110	120			
	0					MAX-1

Conteúdo - 110

dados	100	110	120			
-------	-----	-----	-----	--	--	--



Estrutura de Dados 1

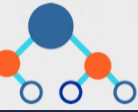
Lista Sequencial Estática – consulta

- Busca por posição:

```
//Arquivo listaSequencial.h  
int consulta_lista_pos(Lista * li, int pos, struct aluno *al);
```

```
//Arquivo listaSequencial.c  
int consulta_lista_pos(Lista * li, int pos, struct aluno *al){  
    if(li == NULL || pos <= 0 || pos >= li->qtd){  
        return 0;  
    }  
    *al = li->dados[pos-1];  
    return 1;  
}
```

```
//programa principal main()  
x = consulta_lista_pos(li, posicao, &dados_aluno);  
if(x){  
    printf("\nConsulta por posicao realizada com sucesso!");  
    printf("\nMatricula: %d", dados_aluno.matricula);  
    printf("\nNota 1: %.2f", dados_aluno.n1);  
    printf("\nNota 2: %.2f", dados_aluno.n2);  
    printf("\nNota 3: %.2f", dados_aluno.n3);  
}else{  
    printf("\nNao foi possivel consultar na posicao especifica!");  
}
```



Estrutura de Dados 1

Lista Sequencial Estática – consulta

- Busca por conteúdo:

```
//Arquivo listaSequencial.h  
int consulta_lista_mat(Lista *li, int mat, struct aluno *al);
```

```
//Arquivo listaSequencial.c  
int consulta_lista_mat(Lista *li, int mat, struct aluno *al){  
    if(li == NULL){  
        return 0;  
    }  
    int k, i = 0;  
    while(i < li->qtd && li->dados[i].matricula != mat){  
        i++;  
    }  
    if(i == li->qtd){//elemento não encontrado  
        return 0;  
    }  
    *al = li->dados[i];  
    return 1;  
}
```



Estrutura de Dados 1

Lista Sequencial Estática – consulta

- Busca por conteúdo:

```
//programa principal main()
x = consulta_lista_mat(li, mat, &dados_aluno);
if(x) {
    printf("\nConsulta por matricula realizada com sucesso!");
    printf("\nMatricula: %d", dados_aluno.matricula);
    printf("\nNota 1: %.2f", dados_aluno.n1);
    printf("\nNota 2: %.2f", dados_aluno.n2);
    printf("\nNota 3: %.2f", dados_aluno.n3);
} else {
    printf("\nNao foi possivel consultar na posicao especifica!");
}
```



Estrutura de Dados 1

Atividade

- Monte todos os módulos e funções deste tipo de lista, chamando todas as funções de forma coerente. Utilize as variáveis de dados sugeridas no slide 22, somente para teste de inserções, buscas e remoções no programa principal - `main()`.
- Crie uma nova função para coleta de dados de novos alunos. Os dados serão informados pelo teclado (usuário vai inserir). Esta função não recebe nenhum parâmetro, e devolve uma estrutura aluno pronta e preenchida, que em seguida será passada à função de **inserção ordenada** que já está pronta.
- Gere entrada para 10 alunos novos, além dos já inseridos inicialmente para teste (os 3 primeiros)
- Entregue no Moodle uma versão funcional, como atividade 1.

