LISTA ESTÁTICA

Tipos de Dados Abstratos (ADT)

- ADT é um tipo de dados que implementa objetos cujo comportamento é definido por um conjunto de valores e operações.
- O conceito da estrutura é separado de sua implementação subjacente.
- O que importa é ***o que*** ela faz, e não ***como*** ela faz.
- Empregadas para simplificar diversas operações em programação.

Os principais tipos de ADTs são a lista, pilha e fila.

Listas Lineares

- Forma simples de interligar os elementos de um conjunto.
- Agrupa informações referentes a um conjunto de elementos que se relacionam entre si de alguma forma.
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulação e compiladores.
- Inúmeros tipos de dados podem ser representados por listas. Alguns exemplos de sistemas de informação são: informações sobre os funcionários de uma empresa, notas de alunos, itens de estoque, etc.

LISTAS LINERARES

- Estrutura em que as operações inserir, retirar e localizar são definidas.
- Itens da lista podem ser acessados, inseridos ou retirados.
- Podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda.
- Duas listas podem ser concatenadas para formar uma lista única, ou uma pode ser partida em duas ou mais listas.
- Podem ser adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.

TAD Lista: Exemplos

- Exemplos de operações possíveis:
  - Criar uma lista linear vazia.
  - Inserir um novo item imediatamente após o *i*-ésimo item.
  - Retirar o *i*-ésimo item.
  - Localizar o *i*-ésimo item para examinar e/ou alterar o conteúdo de seus componentes.
  - Combinar duas ou mais listas lineares em uma lista única.
  - Partir uma lista linear em duas ou mais listas.
  - Fazer uma cópia da lista linear.
  - Ordenar os itens da lista em ordem ascendente ou descendente, de acordo com alguns de seus componentes.
  - Pesquisar a ocorrência de um item com um valor particular em algum componente.

Listas Lineares em Alocação Seqüencial e Estática

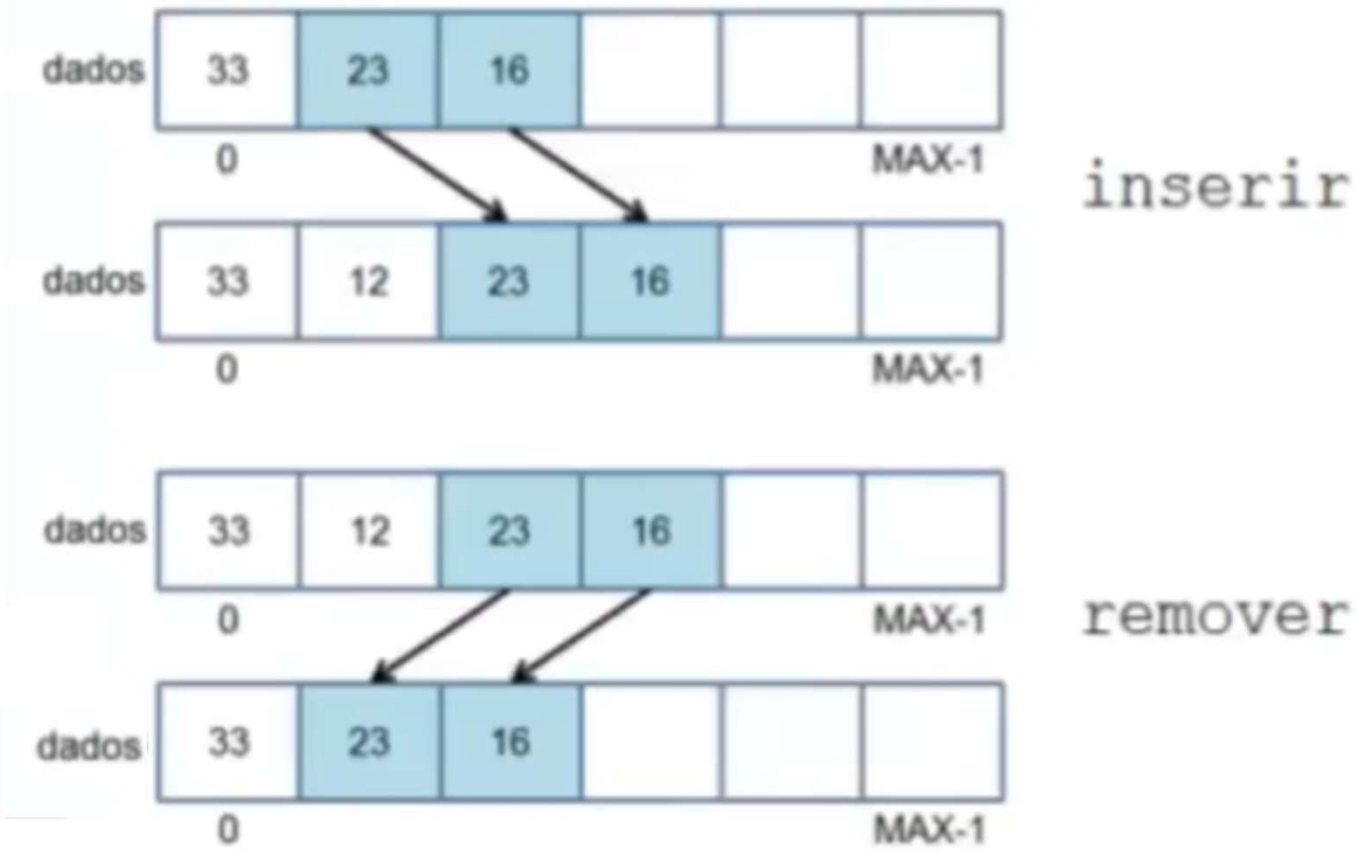
- Armazena itens em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.
- A inserção de um novo item pode ser realizada após o último item com custo constante.
- A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
- Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.

Lista com alocação sequencial e estática: vantagens e desvantagens

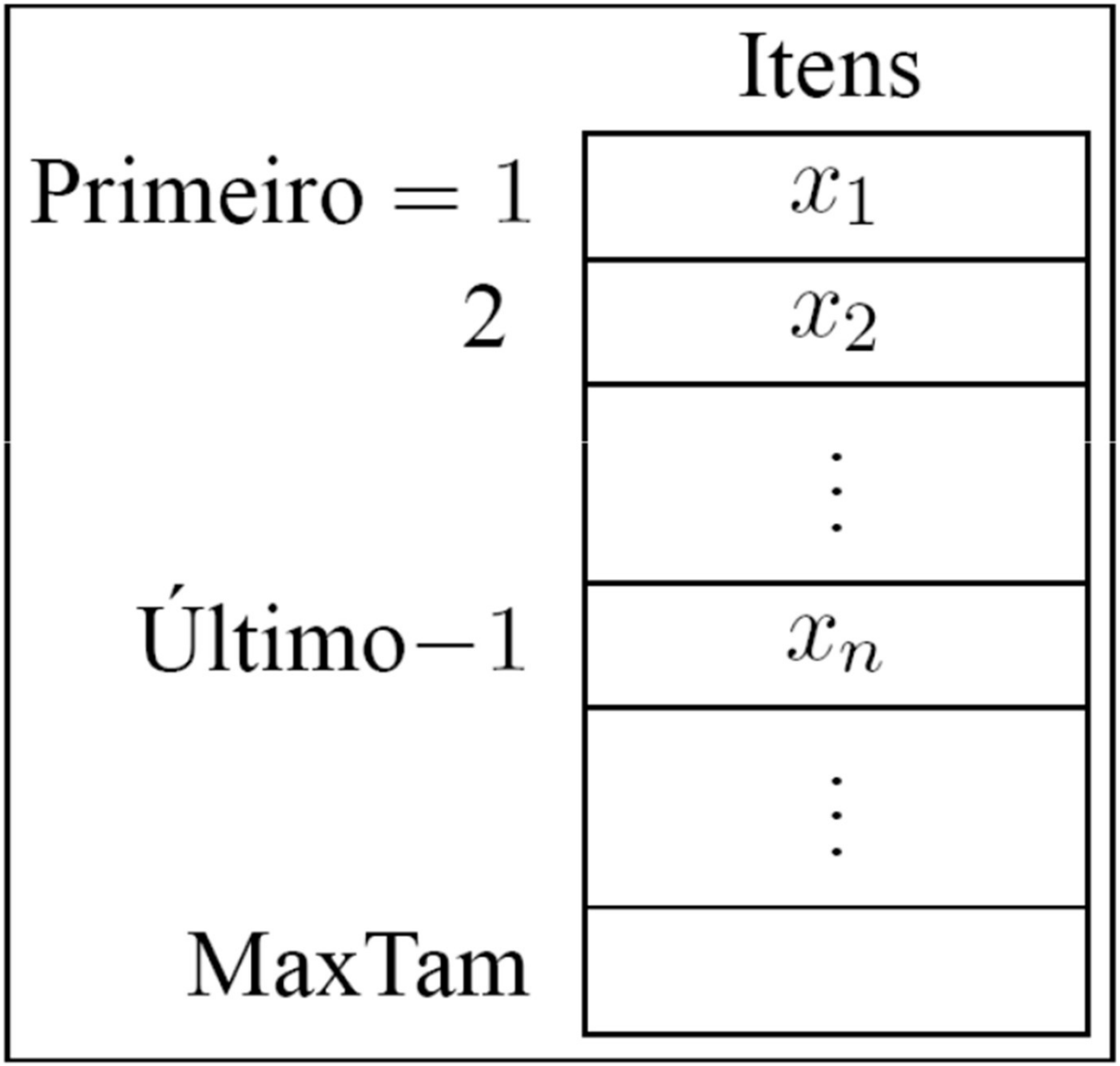
- Vantagem: economia de memória (os ponteiros são implícitos nesta estrutura).
- Desvantagens:
  - custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso;
  - em aplicações em que não existe previsão sobre o crescimento da lista, a utilização de arranjos em linguagens como o Pascal e o C pode ser problemática pois, neste caso, o tamanho máximo da lista tem de ser definido em tempo de compilação.

Inserir e remover
→ desloca todos os elementos da lista

Exemplo: inserir o elemento 12 na posição 2 do meu vetor

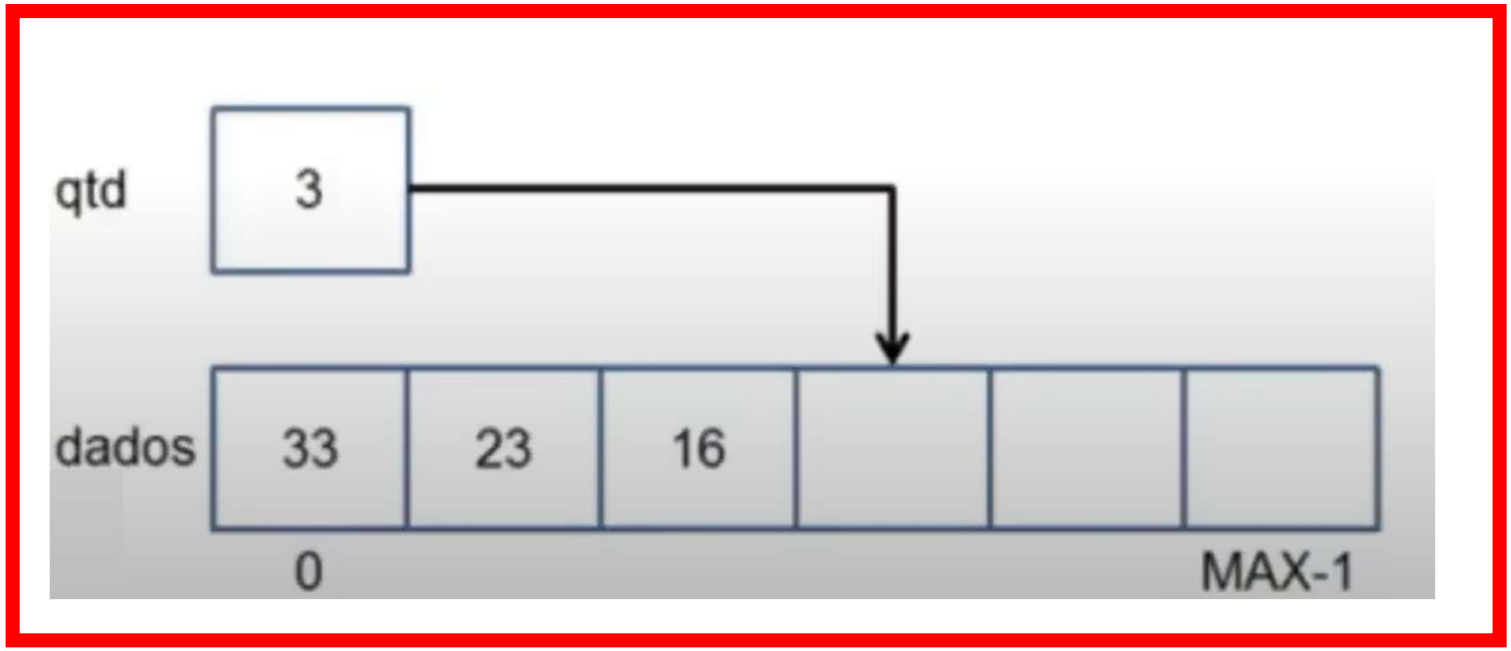


LISTAS LINERARES

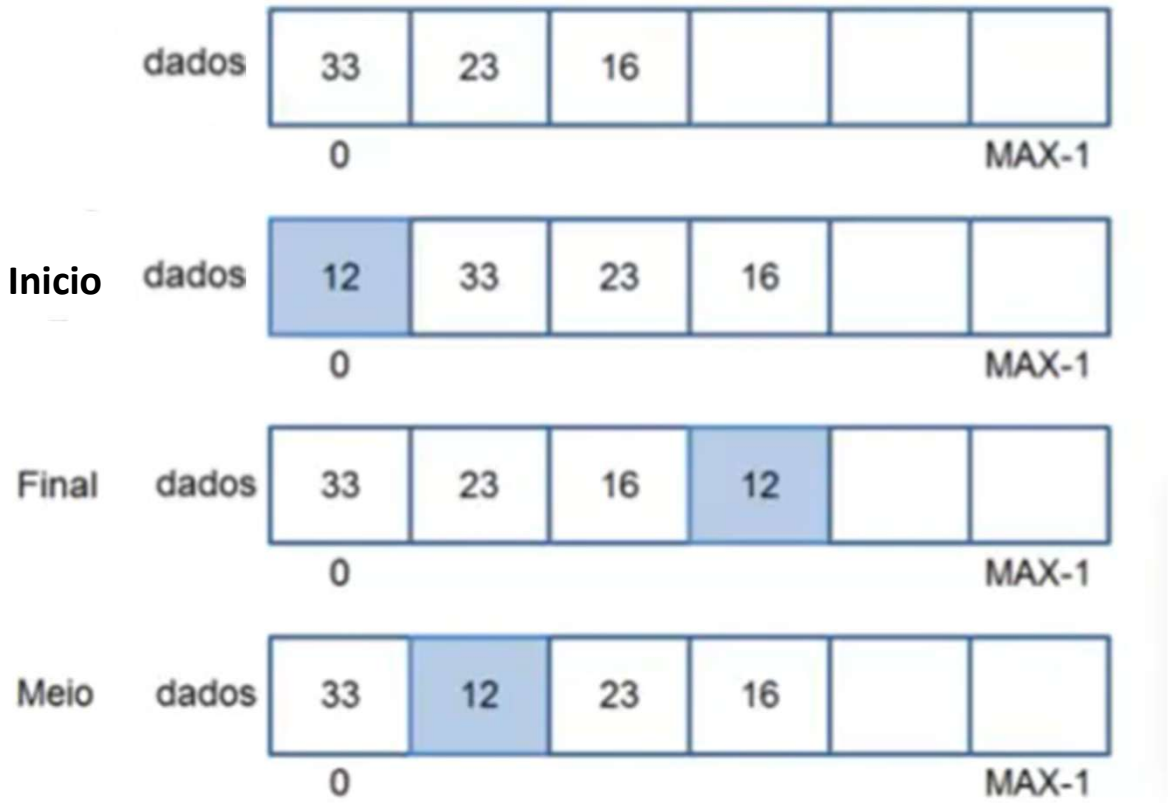


```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   Lista sequencial estática: Tipo de Lista onde o sucessor de um elemento ocupa a posição física do seguinte do mesmo (uso de array)

}
```

LISTA

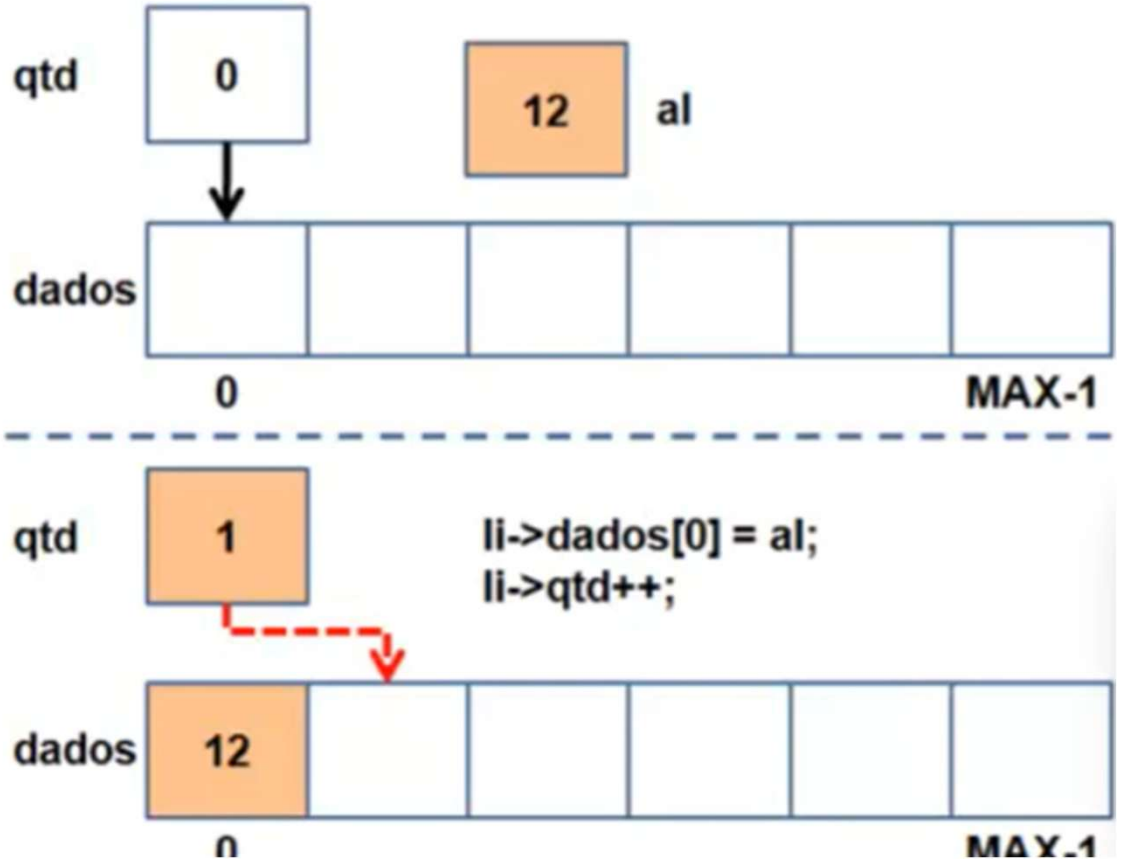


Lista – Inserção
Início, meio e no final



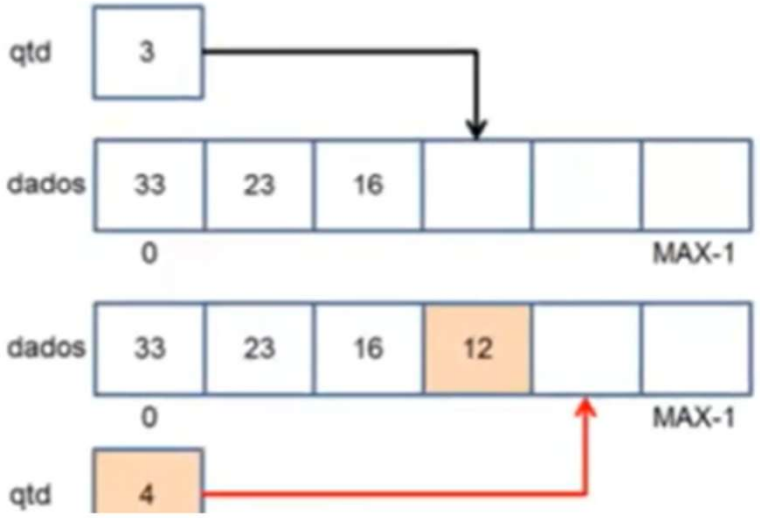
Lista – Inserção lista vazia

Não pode inserir elementos em lista cheia



Lista – inserção no final

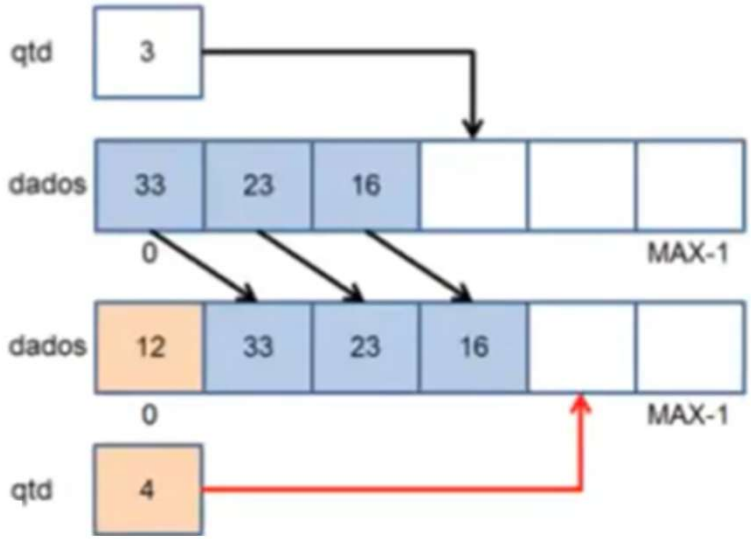
```c
//programa principal
int x = insere_lista_final(li, dados_aluno);
//Arquivo ListaSequencial.h
int insere_lista_final(Lista* li, struct aluno al);
//Arquivo ListaSequencial.c
int insere_lista_final(Lista* li, struct aluno al){
    if(li == NULL)
        return 0;
    if(lista_cheia(li))
        return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



Lista – inserção no início

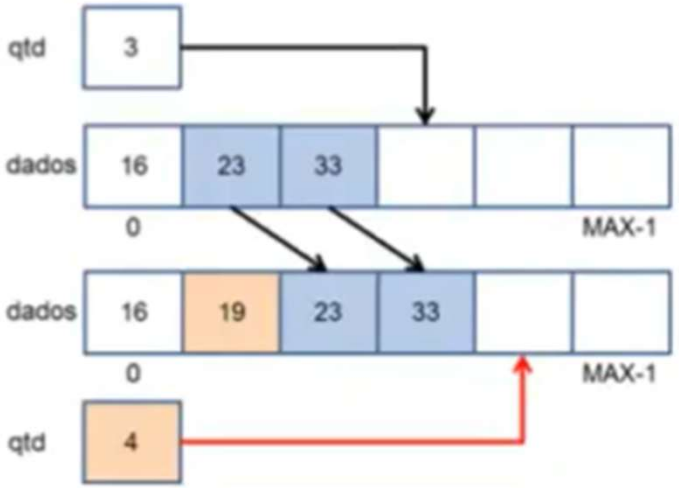
```
//programa principal
int x = insere_lista_inicio(li, dados_aluno);
//Arquivo ListaSequencial.h
int insere_lista_inicio(Lista* li, struct aluno al);
//Arquivo ListaSequencial.c
int insere_lista_inicio(Lista* li, struct aluno al){
    if(li == NULL) return 0;
    if(lista_cheia(li)) return 0;
    int i;
    for(i=li->qtd-1; i>=0; i--)
        li->dados[i+1] = li->dados[i];
    li->dados[0] = al;
    li->qtd++;
    return 1;
}
```

Gravar no vetor dados na posição i + 1 para não apagar os dados da posição corrente, uma vez que todo o vetor deve ser movimentado para inserir o item na primeira posição.



Lista – Inserção de um elemento de forma ordenada

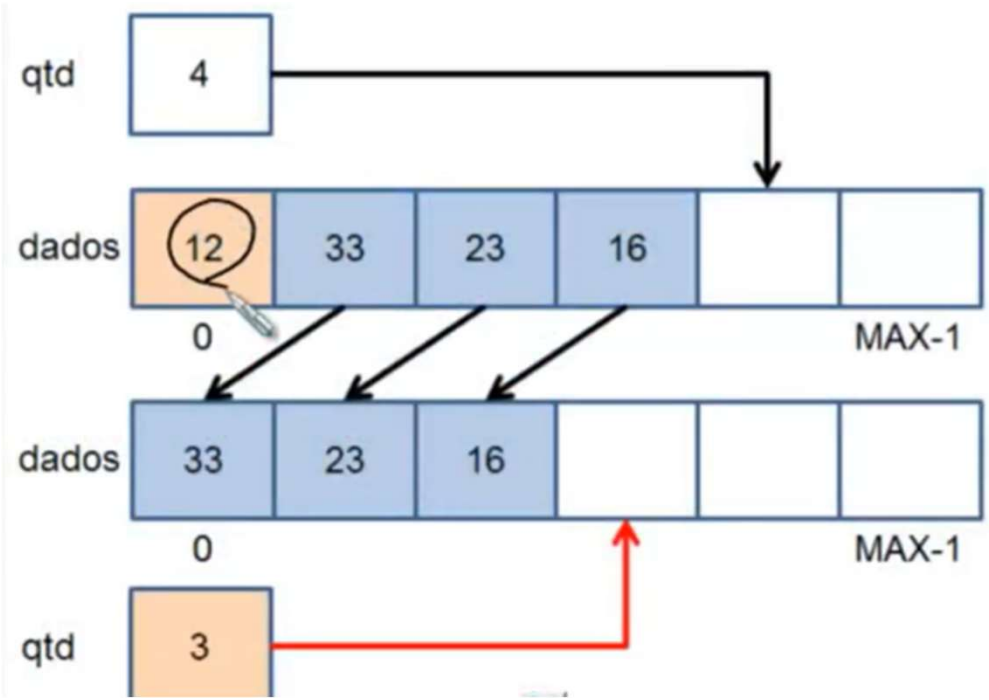
```
//programa principal
int x = insere_lista_ordenada(li, dados_aluno);
//Arquivo ListaSequencial.h
int insere_lista_ordenada(Lista* li, struct aluno al);
//Arquivo ListaSequencial.c
int insere_lista_ordenada(Lista* li, struct aluno al){
    if(li == NULL) return 0;
    if(lista_cheia(li)) return 0;
    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;
```



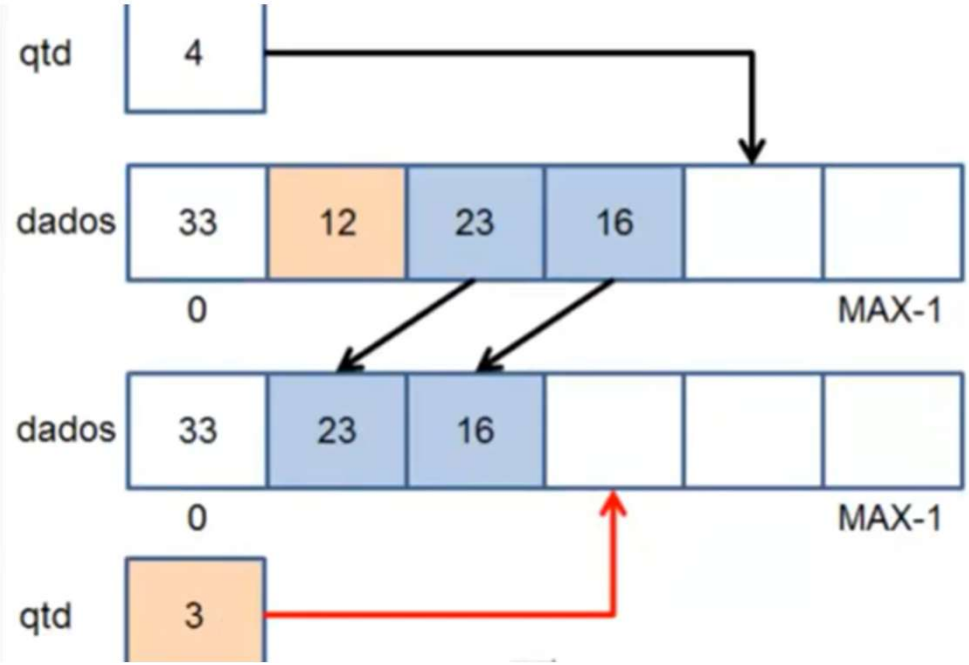
Lista – Remoção

Início, meio e no final

Início

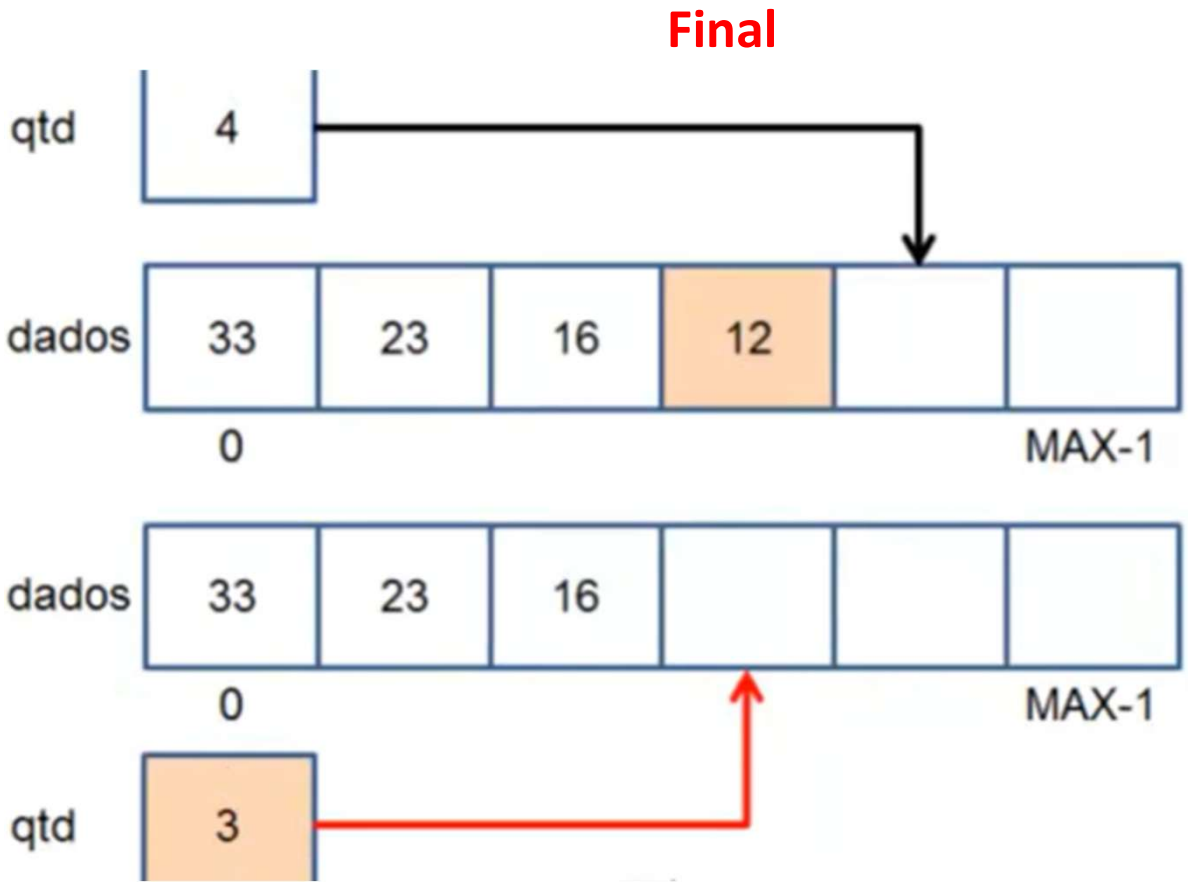


Meio



Lista – Remoção

Início, meio e no final

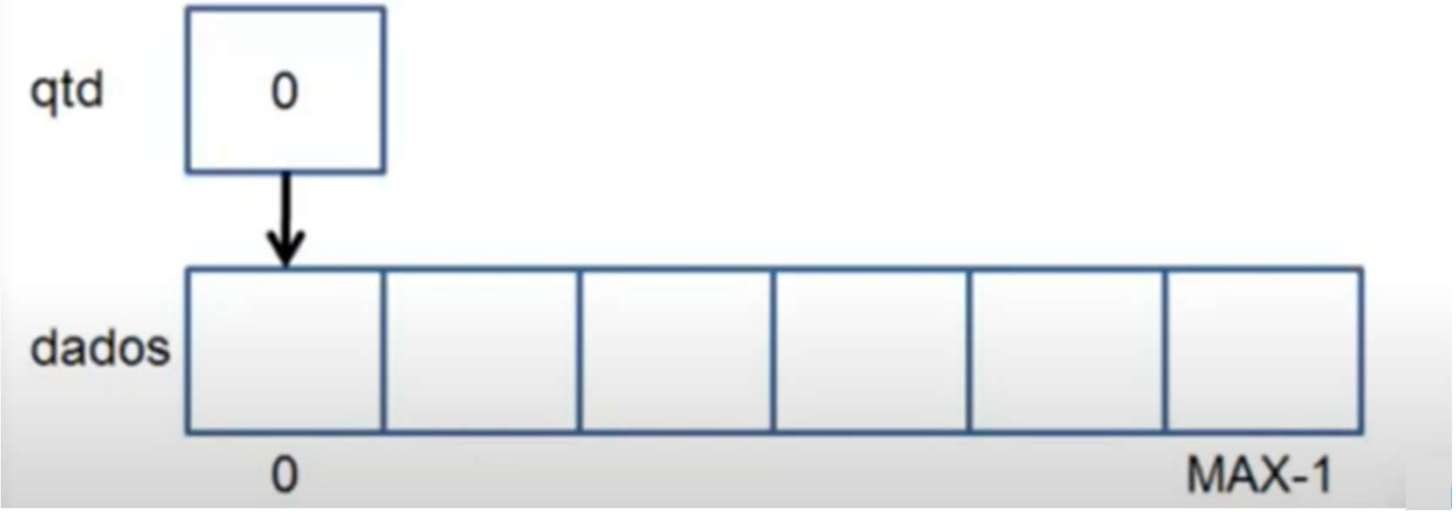


Não precisa deslocar nenhum elemento, só decrementa o campo qtd

Lista- Remoção

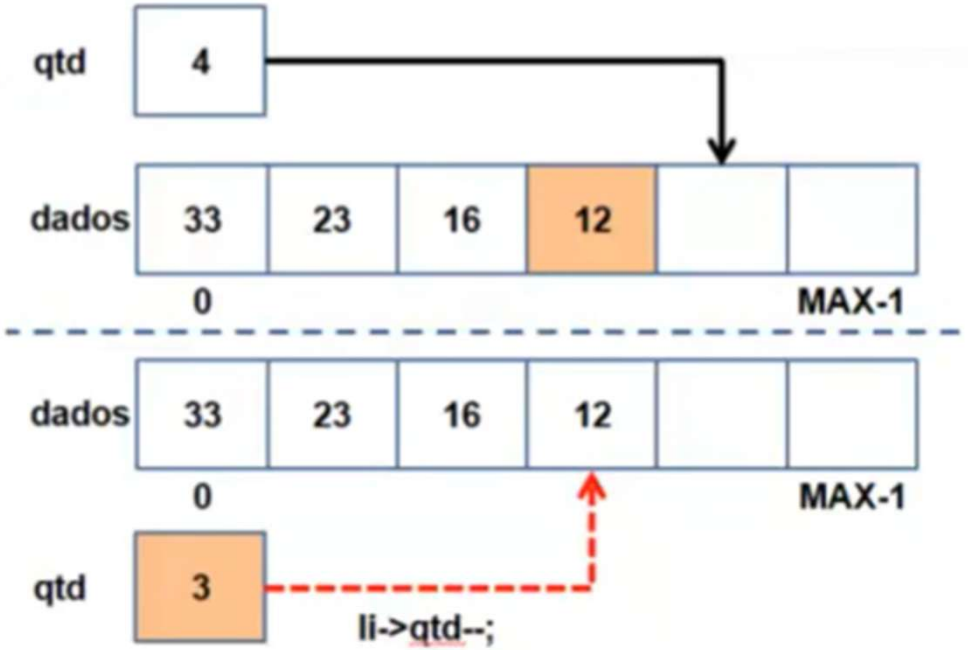
Os 3 tipos de remoção trabalham juntos. A remoção sempre remove um elemento específico da lista, o qual pode estar no início, no meio ou no final da lista

Cuidado: não se pode remover de uma lista vazia



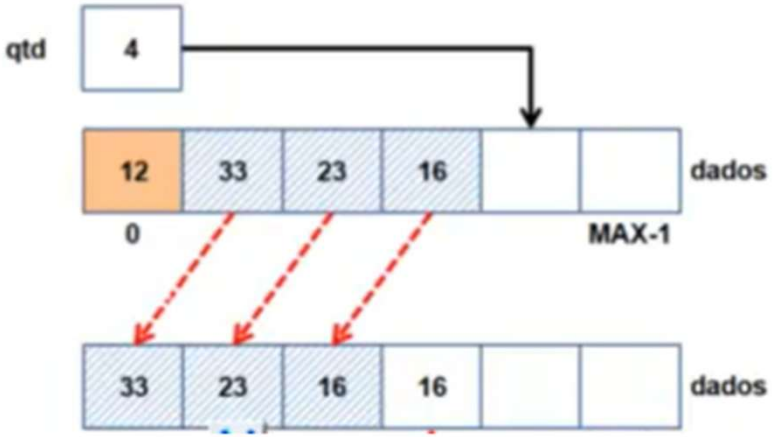
Lista – Remoção do final

```
//programa principal
int x = remove_lista_final(li);
//Arquivo ListaSequencial.h
int remove_lista_final(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;
}
```



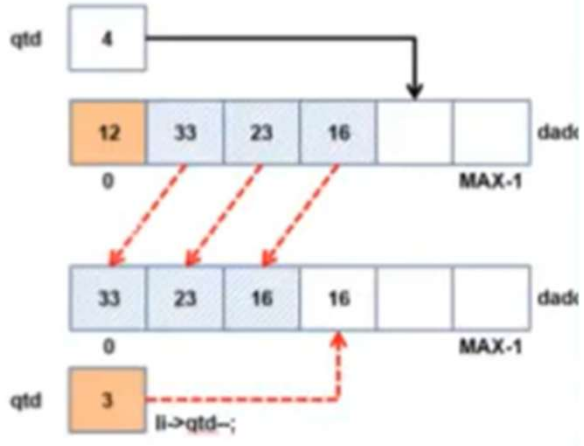
Lista – Remoção do início

```
//programa principal
int x = remove_lista_inicio(li);
//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;
}
```



Lista – Remoção de um elemento qualquer

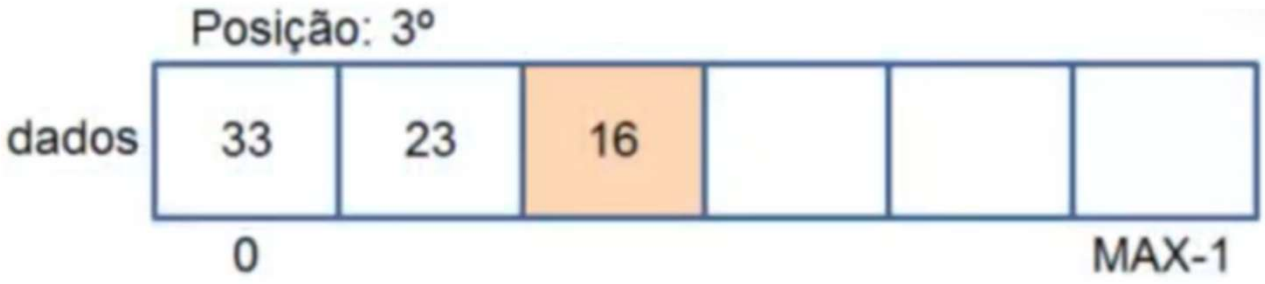
```c
//Arquivo ListaSequencial.h
int remove_lista(Lista* li, int mat);
//Arquivo ListaSequencial.c
int remove_lista(Lista* li, int mat){
    if(li == NULL) return 0;
    if(li->qtd == 0) return 0;
    int k,i = 0;
    while(i<li->qtd && li->dados[i].matricula != mat)
        i++;
    if(i == li->qtd)//elemento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

    for(k=i; k< li->qtd-1; k++)
        li->dados[k] = li->dados[k+1];
    li->qtd--;
    return 1;
}
//programa principal
int x = remove_lista(li, matricula_aluno);
```

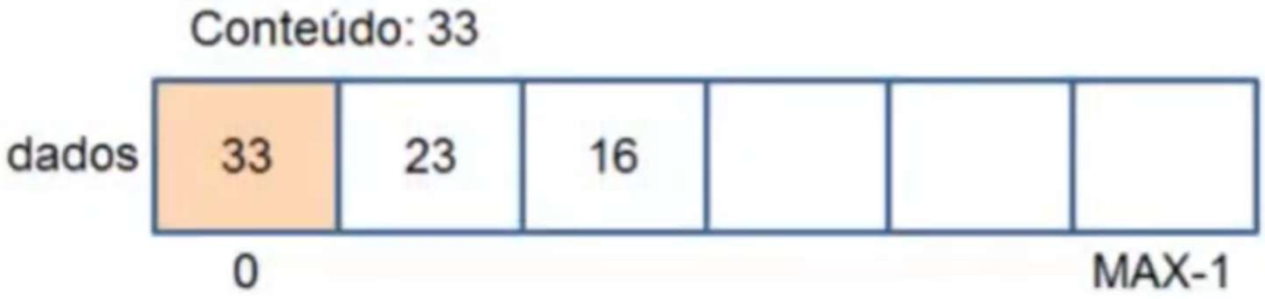


Lista - Consulta

Existem 2 maneiras de consultar um elemento de uma lista:

- pela posição (acesso direto)
- pelo conteúdo (necessidade de busca)





Lista – Consulta pela posição

```c
//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
int x = consulta_lista_pos(li, posicao, &dados_aluno);
```

Lista – Consulta pelo conteúdo

```c
//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 nao encontrado
        return 0;

    *al = li->dados[i];
    return 1;
}
```