# FILA ESTÁTICA

# Tipos de Dados Abstratos (ADT)

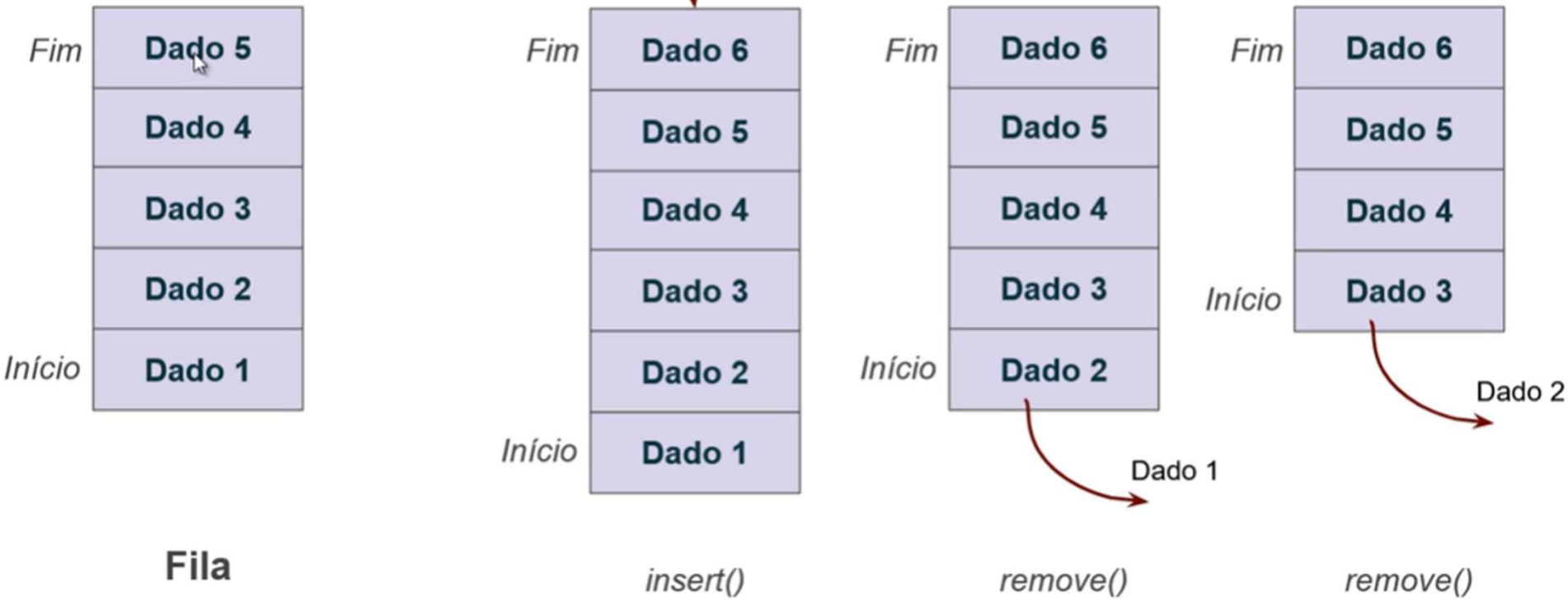
- ADT é um tipo de dados que implementa objetos cujo comportamento é definido por um conjunto de valores e operações.
- O conceito da estrutura é separado de sua implementação subjacente.
- O que importa é ***o que*** ela faz, e não ***como*** ela faz.
- Empregadas para simplificar diversas operações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

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

# Fila

- Estrutura linear na qual a inserção e remoção de itens segue o princípio FIFO (first-in-first-out)
- Os objetos podem ser inseridos a qualquer momento, mas apenas o objeto que está há mais tempo na fila pode ser removido.
- Elementos somente são inseridos em um lado da lista, o final, e somente podem ser excluídos do outro lado, o início da lista.
- Cada nó contém um ponteiro para os dados e um ponteiro de ligação (link) para o próximo elemento na fila.

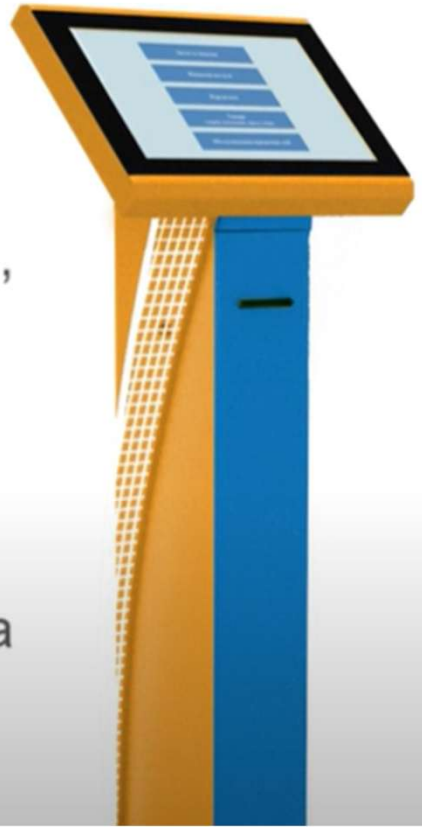
# Fila: funcionamento



# Operações em Filas

Diversas operações podem ser realizadas em filas, tais como:

- enqueue(): Insere um item no final da fila
- dequeue(): Retira e retorna o elemento que está no início da fila, se ela não estiver vazia.
- peek(): Retorna um elemento da fila, sem removê-lo, se ela não estiver vazia.
- size(): retorna o tamanho da pilha (n° de itens)
- isEmpty(): retorna um booleano informando se a pilha está vazia
- isFull(): retorna um booleano informando se a pilha está cheia



# Aplicações de Filas

São usadas como ferramentas para criação de algoritmos, assim como as pilhas. Exemplos de uso de filas incluem:

- Resposta a requisições de serviços compartilhados, como filas de impressão e acesso a disco
- Armazenamento de teclas digitadas no teclado
- Agendamento de tempo de CPU
- Transferência de dados assíncrona entre processos (buffers, E/S em arquivos, etc.)
- Contadores de tíquetes e gerenciamento de filas por senhas (em hospitais, por exemplo) e listas de espera em geral

| Fila |
| --- |
| Baseadas em FIFO: o primeiro elemento a ser inserido é tb o primeiro a ser retirado |
| Inserção e exclusão ocorrem nos dois lados da lista: inserção no final e exclusão no início. |
| Operação de inserção: enqueue()<br>Operação de exclusão: dequeue() |
| Dois ponteiros são necessários: um aponta para o primeiro elemento e o outro, para o último |
| Usada quando precisamos acessar os elementos em ordem de chegada, ou prioridade |

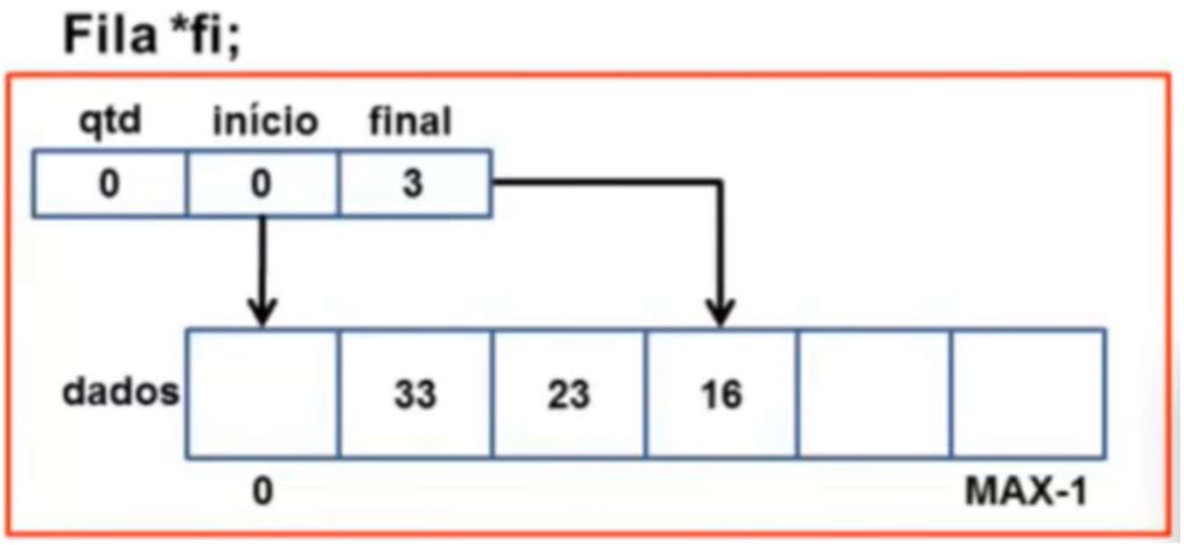
**Fila**

- cria uma fila vazia;
- insere um elemento no fim da fila;
- remove o elemento do inicio da fila;
- verifica se a fila esta vazia;
- libera a fila.

```
Implementando uma "Fila Estática"
    "FilaEstatica.h": definir
        - os protótipos das funções
        - o tipo de dado armazenado na fila
        - o ponteiro "fila"
        - tamanho do vetor usado na fila

    "FilaEstatica.c": definir
        - o tipo de dados "fila"
        - implementar as suas funções
```

# Fila estática → uso de arrays



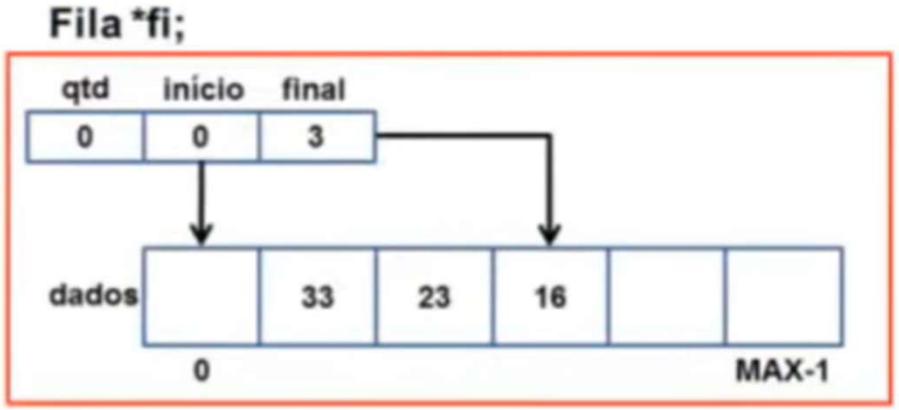

Estrutura com a quantidade

Vetor de elementos

```c
//Arquivo FilaEstatica.h
#define MAX 100
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct fila Fila;

//Arquivo FilaEstatica.c
struct fila{
    int inicio, final, qtd;
    struct aluno dados[MAX];
};

//programa principal
Fila *fi;
```

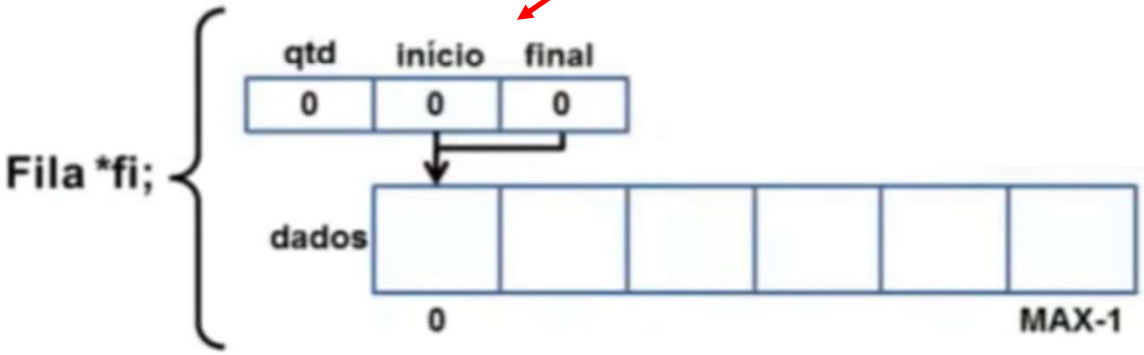


# Criar a fila

```
//programa principal
fi = cria_Fila();
//Arquivo FilaEstatica.h
Fila* cria_Fila();
//Arquivo FilaEstatica.c
Fila* cria_Fila(){
    Fila *fi = (Fila*) malloc(sizeof(struct fila));
    if(fi != NULL){
        fi->inicio = 0;
        fi->final = 0;
        fi->qtd = 0;
    }
    return fi;
}
```

Inicializando com zero

Alocação de memória para toda a estrutura da Fila

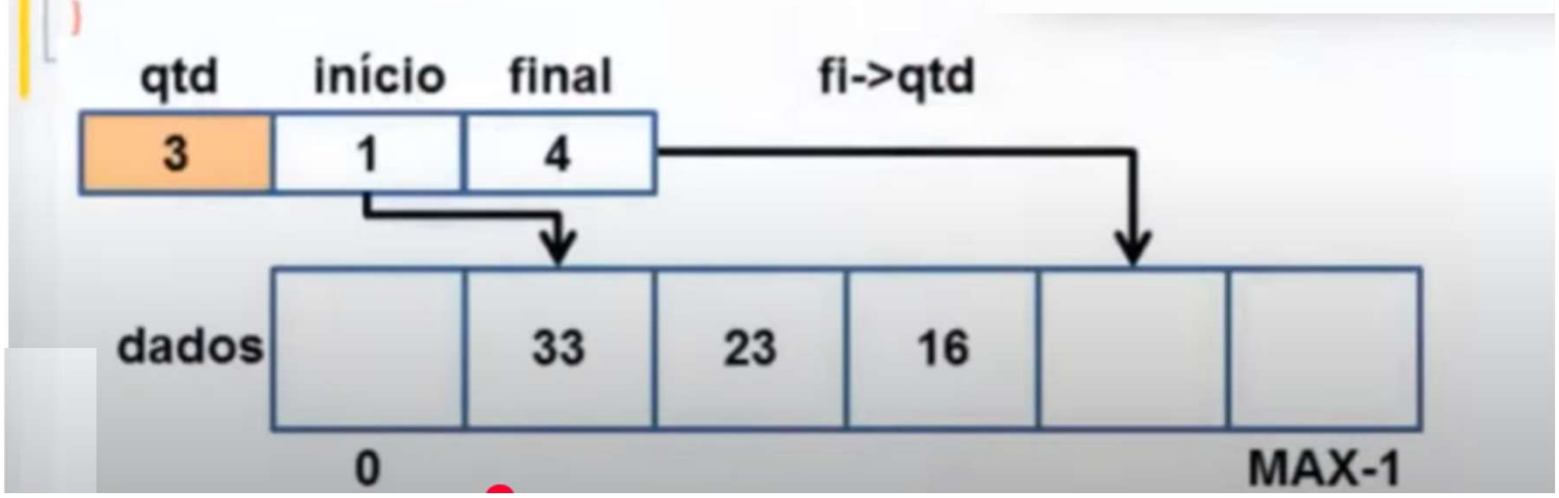


## Liberar a fila

```
//programa principal
libera_Fila(fi);
//Arquivo FilaEstatica.h
void libera_Fila(Fila* fi);
//Arquivo FilaEstatica.c
void libera_Fila(Fila* fi){
    free(fi);
}
```

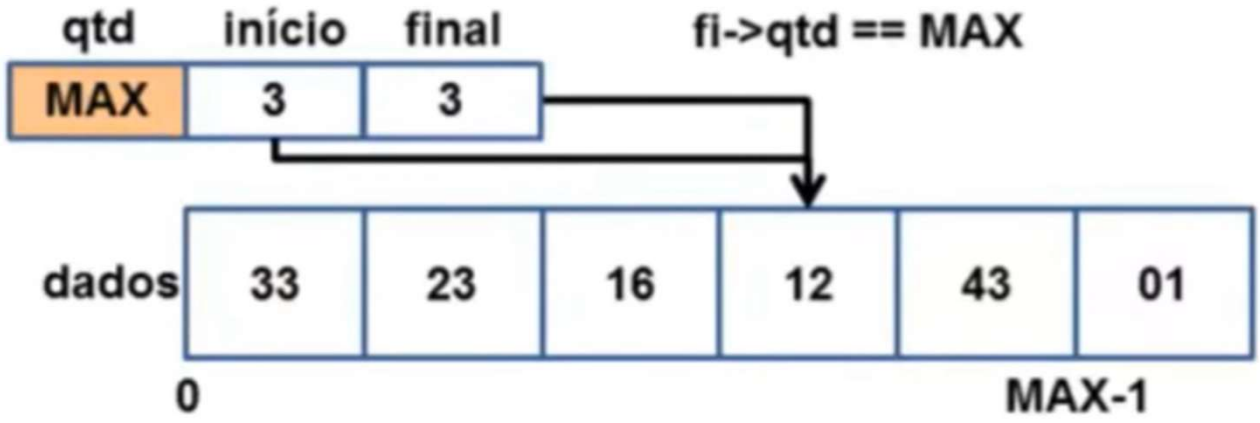
# Tamanho da fila

```c
//programa principal
int x = tamanho_Fila(fi);
//Arquivo FilaEstatica.h
int tamanho_Fila(Fila* fi);
//Arquivo FilaEstatica.c
int tamanho_Fila(Fila* fi){
    if(fi == NULL)
        return -1;
    return fi->qtd;
}
```



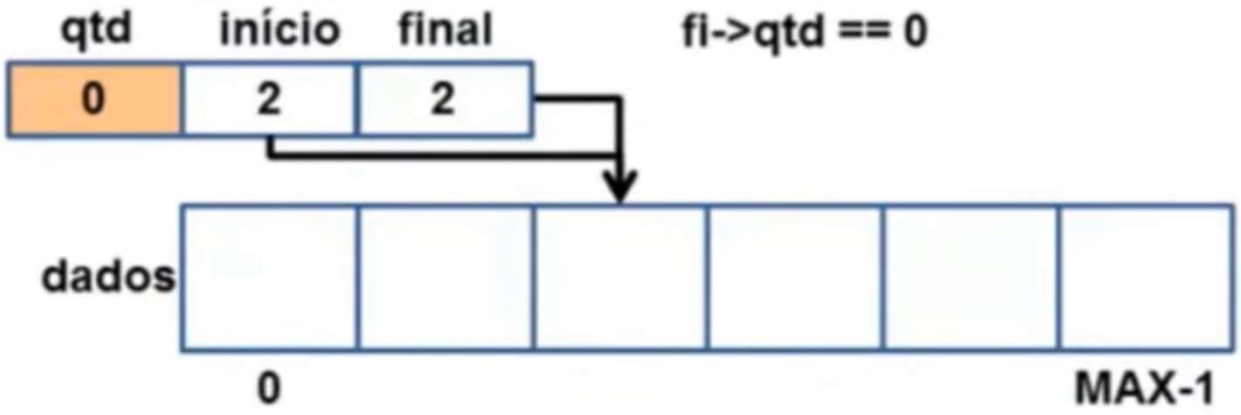
# Fila cheia

```c
//programa principal
int x = Fila_cheia(fi);
if(Fila_cheia(fi))
//Arquivo FilaEstatica.h
int Fila_cheia(Fila* fi);
//Arquivo FilaEstatica.c
int Fila_cheia(Fila* fi){
    if(fi == NULL) return -1;
    if (fi->qtd == MAX)
        return 1;
    else
        return 0;
}
```



# Fila vazia

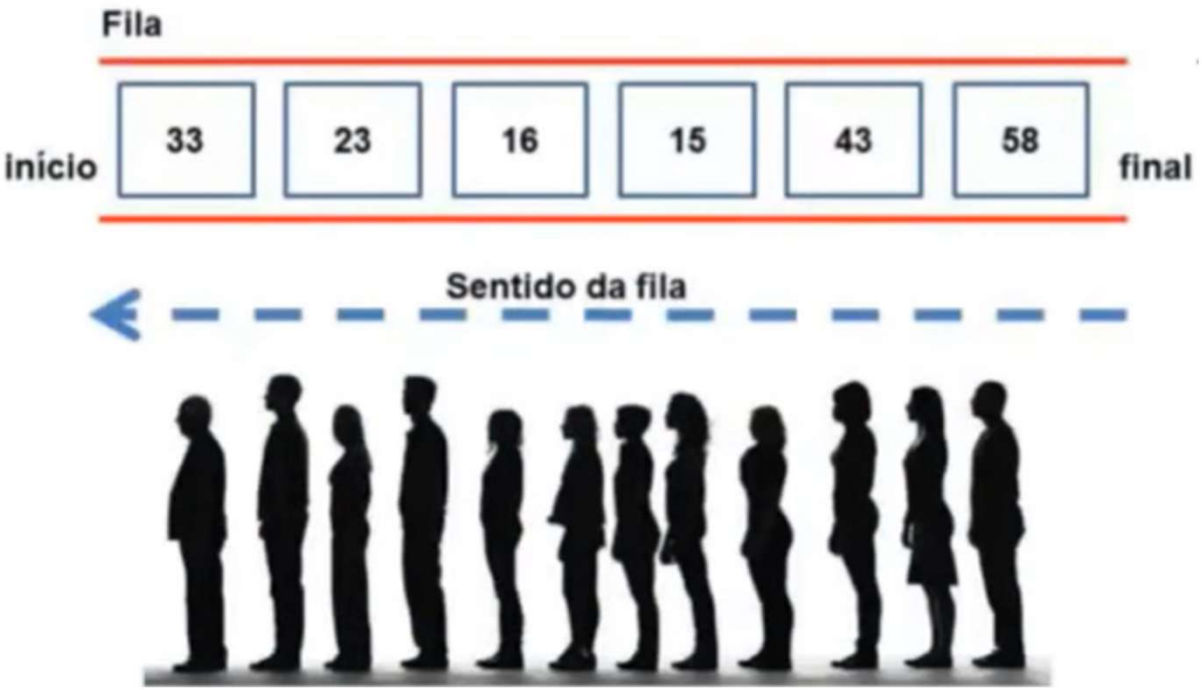
```c
//programa principal
int x = Fila_vazia(fi);
if(Fila_vazia(fi))
//Arquivo FilaEstatica.h
int Fila_vazia(Fila* fi);
//Arquivo FilaEstatica.c
int Fila_vazia(Fila* fi){
    if(fi == NULL) return -1;
    if (fi->qtd == 0)
        return 1;
    else
        return 0;
}
```



**Inserção de um elemento**

**Sempre no final da fila**

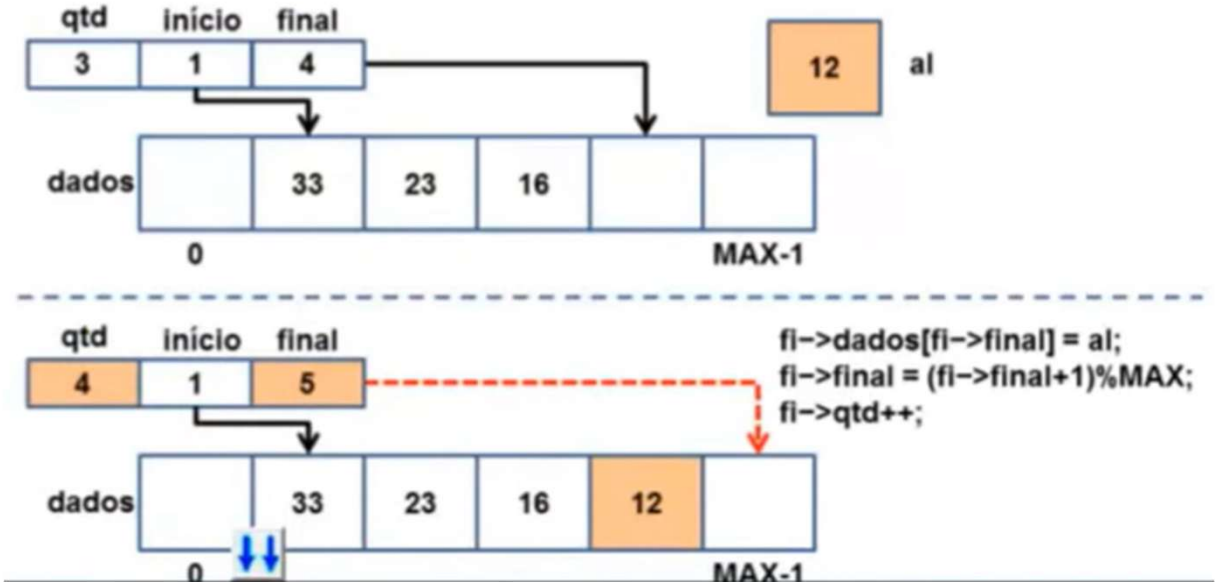
****** não inserir elementos em fila cheia**



# Inserção de um elemento

```
//programa principal
int x = insere_Fila(fi, dados_aluno);
//Arquivo FilaEstatica.h
int insere_Fila(Fila* fi, struct aluno al);
//Arquivo FilaEstatica.c
int insere_Fila(Fila* fi, struct aluno al){
    if(fi == NULL) return 0;
    if(Fila_cheia(fi)) return 0;
    fi->dados[fi->final] = al;
    fi->final = (fi->final+1)%MAX;
    fi->qtd++;
    return 1;
}
```

**A função %MAX garante que não caminho para uma posição inexistente. Cheguei para o último elemento, volte para o inicio em que existem posição vazia. Crio uma circularidade.**

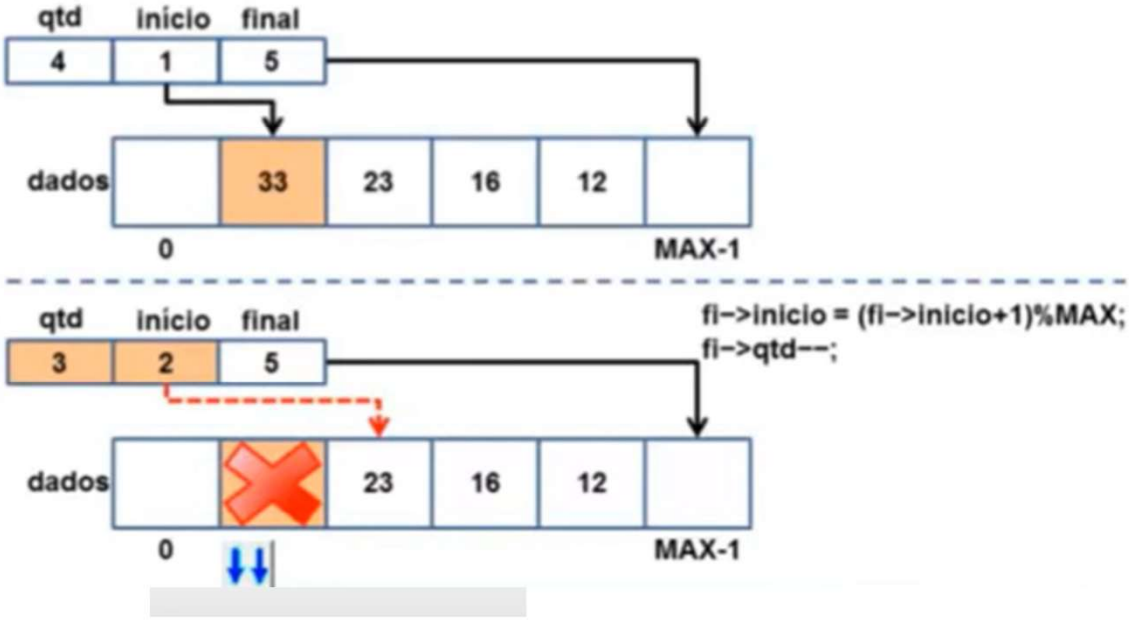


## Remoção de um elemento

## Sempre no inicio da fila

## ****** Não posso remover itens de uma lista vazia

```c
//programa principal
int x = remove_Fila(fi);
//Arquivo FilaEstatica.h
int remove_Fila(Fila* fi);
//Arquivo FilaEstatica.c
int remove_Fila(Fila* fi){
    if(fi == NULL || Fila_vazia(fi))
        return 0;
    fi->inicio = (fi->inicio+1)%MAX;
    fi->qtd--;
    return 1;
}
```



## Consulta elemento

A consulta em uma Fila se dá pelo elemento que está no início da Fila.

```
//programa principal
int x = consulta_Fila(fi, &dados_aluno);
//Arquivo FilaEstatica.h
int consulta_Fila(Fila* fi, struct aluno *al);
//Arquivo FilaEstatica.c
int consulta_Fila(Fila* fi, struct aluno *al){
    if(fi == NULL || Fila_vazia(fi))
        return 0;
    *al = fi->dados[fi->inicio];
    return 1;
}
```