SCE 202 – Algoritmos e Estruturas de Dados I

# Filas

# Implementação Sequencial

# Conceito

- Coleção ordenada de itens (lista ordenada) em que a inserção de um novo item se dá em um dos lados – no fim – e  a remoção no outro lado – no início.
  - Listas FIFO/LILO (**F**irst **I**n **F**irst **O**ut/ **L**ast **I**n **L**ast **O**ut).
- Modelos intuitivos de filas são as linhas para comprar bilhetes de cinema e de caixa de supermercado.
- A fila, como a pilha, é conceitualmente uma estrutura dinâmica que está continuamente mudando pois itens são adicionados/retirados.

# TAD – Fila – Operações

```
void definir (fila *q);
  /*Cria uma fila vazia. Deve ser usado antes de
   qualquer outra operação*/

void tornar_vazia (fila *q);
  /*Reinicializa uma fila existente, q, como uma
   fila vazia. Dependendo da implementação da
   estrutura de dados, deve remover todos os seus
   elementos.*/

boolean vazia (fila *q);
  /*Retorna true se fila não contém elementos, false
   caso contrário*/

boolean inserir (fila *q, tipo_info item);
  /*Adiciona um item no fim da fila q. Retorna true
   se operação realizada com sucesso, false caso
   contrário*/
```

# TAD – Fila – Operações

```
boolean remover(fila *q, tipo_info *item);
  /*Remove um item do início da fila q. Retorna true
   se operação realizada com sucesso, false caso
   contrário*/

int tamanho (fila *q);
  /*Retorna o tamanho da fila*/

boolean começo_fila (fila q, tipo_info *item);
  /*Mostra o começo da fila sem remover o item.
   Retorna true se operação realizada com sucesso,
   false caso contrário*/
```

# Implementações de Filas: Sequencial

Há um meio de se utilizar de um array na implementação de uma fila?

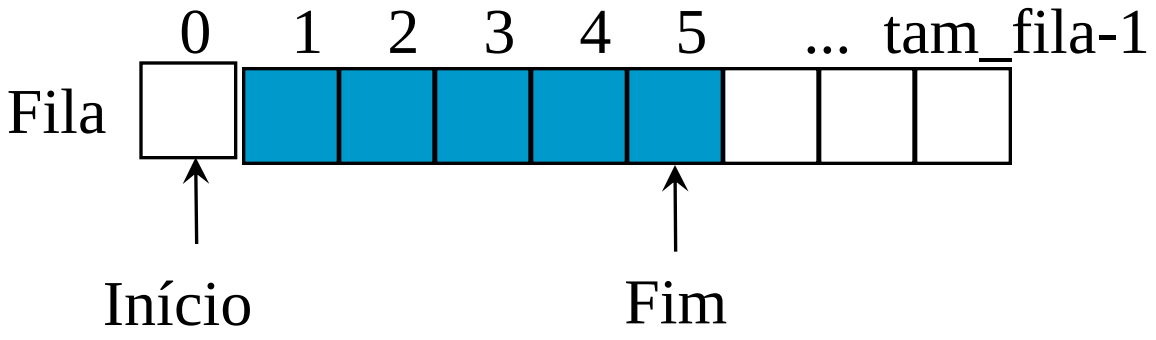
SIM, se nós dimensionarmos o array com um tamanho que dê para acomodar o tamanho máximo da fila, e além disso precisamos dos ponteiros FIM e COMEÇO.

# Implementações de Filas: Sequencial

```
#define tam_fila 100
#define indice int

typedef struct{
    tipo_info A[tam_fila];
    indice inicio, fim;
}fila;
```

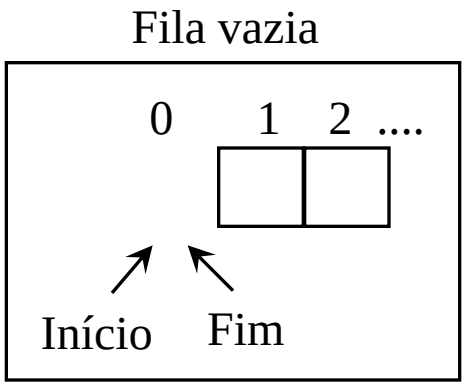
**inicio**: aponta para a posição **anterior** ao 1º elemento.
**fim**: aponta para a posição do **último** elemento.

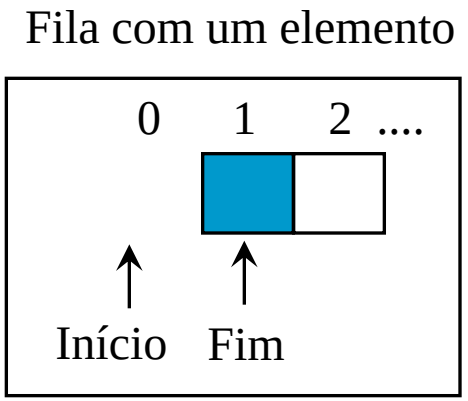


# Implementações de Filas: Sequencial

O que é então uma fila vazia?

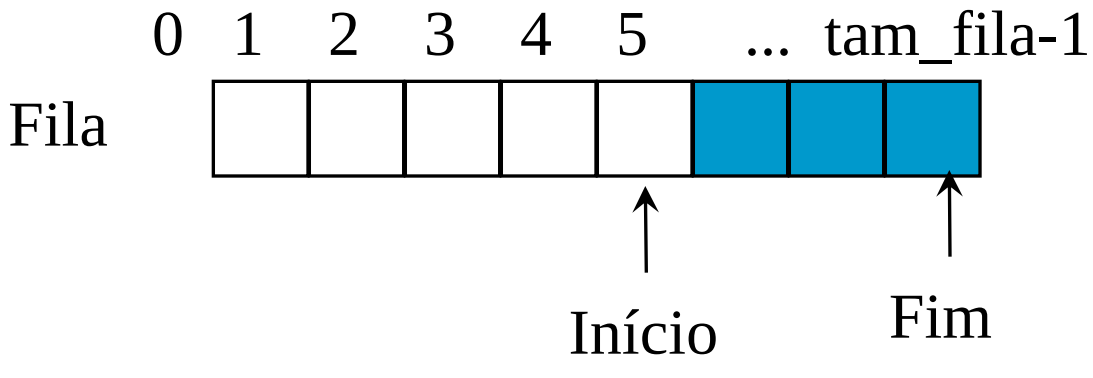
No começo: inicio = fim = 0



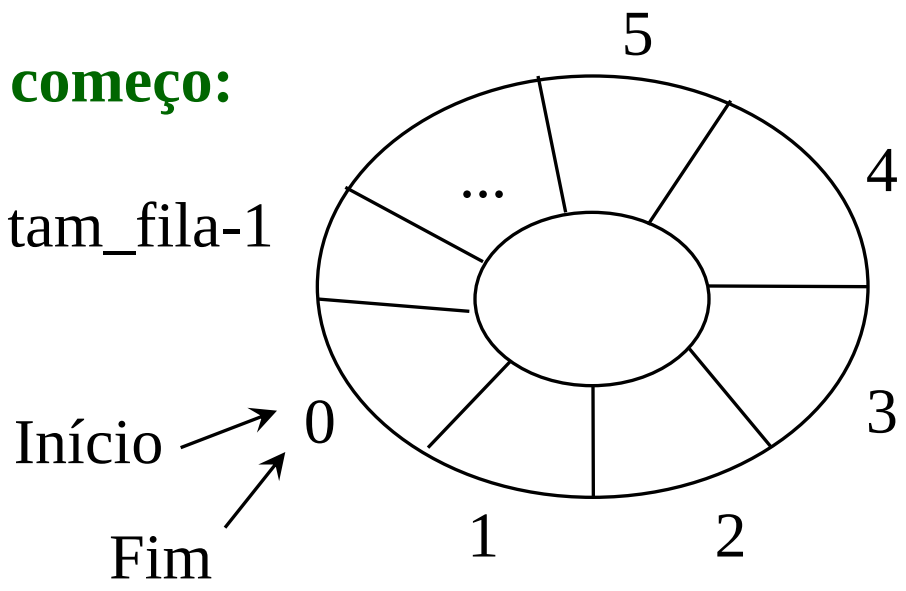

Num instante qualquer: Inicio = Fim

E o que é uma fila cheia? Fim = tam_fila?



# Solução: Fila do Tipo Anel

**No começo:**



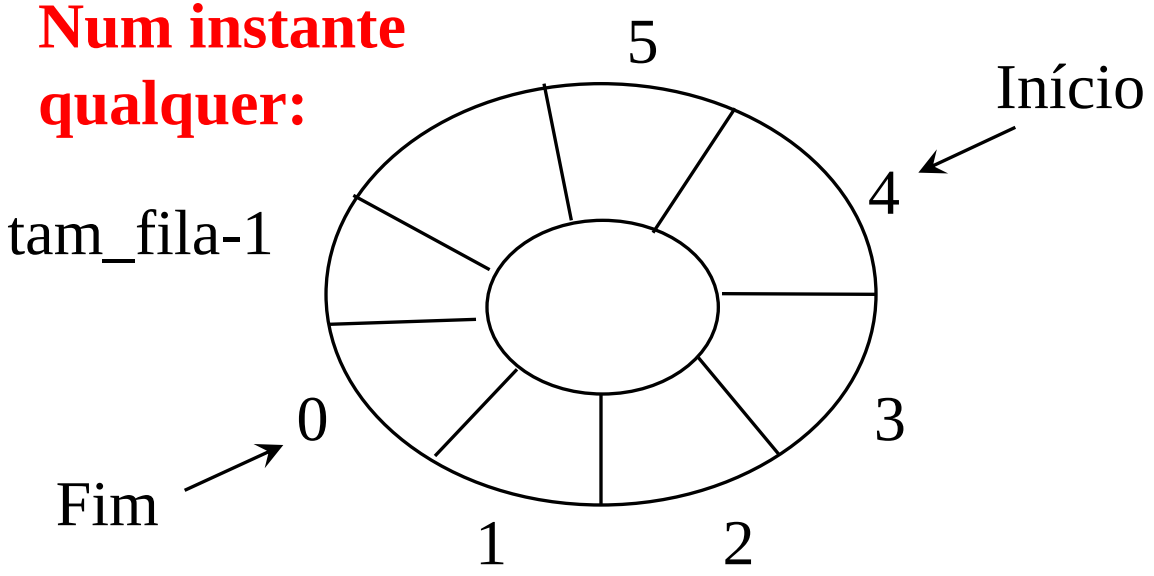

fila vazia =
(Início=Fim=0)

Inserir: incrementa Fim, se não estiver cheia
Eliminar: se não estiver vazia (Início=Fim), incrementa Início.
Incrementar, no Anel, implica ignorar limite do tam_fila:

**Início = (Início +1) % tam_fila**
**Fim = (Fim +1) % tam_fila**

# Fila tipo Anel

**Num instante qualquer:**




**Se permitirmos o uso da posição Início para inserção, as condições de fila cheia e vazia seriam idênticas.**

**Providência: uma posição é sacrificada; apenas tam_fila-1 posições são utilizadas pela fila.**

**Assim:**

**Condição de Fila Cheia = (Fim +1) % tam_fila = Início**

**Condição de Fila Vazia = Fim = Início**

# Implementações de Filas: Circular Sequencial

```
#define tam_fila 100  /*nº máx. itens na fila*/
#define indice int
/*permite um espaço em branco para diferenciar lista
  cheia de vazia*/

typedef struct{
  tipo_info A[tam_fila];
  indice inicio, fim;
}fila;

fila q; /*tipo de declaração*/
```

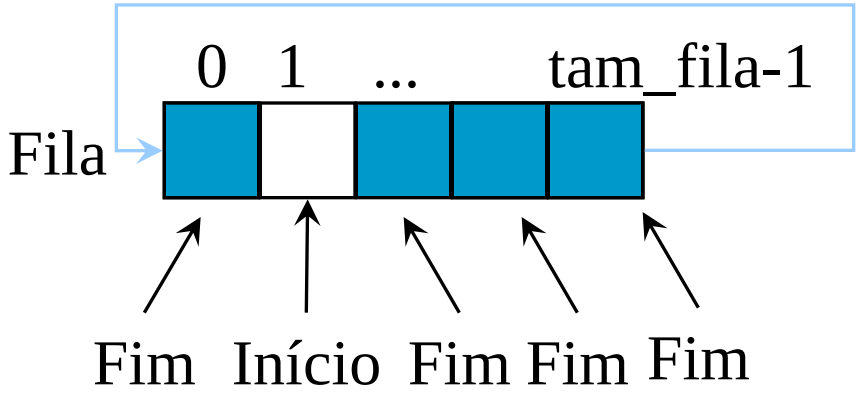
```
void definir (fila *q){
   /*Cria uma fila vazia. Deve ser usado antes de qualquer outra
   operação*/
   q->fim = 0;
   q->inicio = 0;
   /*ponteiro de início atrasado; aponta para uma posição
   anterior ao início*/
}

boolean vazia (fila *q){
   /*Retorna true se fila não contém elementos, false caso
   contrário*/
   return (q->inicio == q->fim);
}

boolean cheia (fila *q){
   /*Retorna true se fila cheia, false caso contrário*/
   return (q->inicio == ((q->fim + 1) % tam_fila);
   /*os dois ponteiros diferem de uma posição*/
}
```

```
boolean inserir (fila *q, tipo_info item){
   /*Adiciona um item no fim da fila q. Retorna true se
   operação realizada com sucesso, false caso contrário*/
   /*uma posição da fila nunca será preenchida*/
   if (cheia(*q))
        return FALSE;

   q->fim = (q->fim + 1) % tam_fila ;
   q->A[q->fim] = item;
   return TRUE;
}
```



```
boolean remover(fila *q, tipo_info *item){
   /*Remove um item do início da fila q. Retorna true se
   operação realizada com sucesso, false caso contrário*/
   if (vazia(*q))
        return FALSE;

   q->inicio = (q->inicio+1) % tam_fila;
   item = q->A[q->inicio]; /*opcional*/
   return TRUE;
}
```

```
int tamanho (fila *q){
   /*retorna o tamanho da fila*/
   if (q->inicio <= q->fim)
        return (q->fim - q->inicio);

   return (tam_fila - (q->inicio - q->fim));
}

boolean começo_fila (fila *q, tipo_info *item){
   /*Mostra o começo da fila sem remover o item. Retorna
   true se operação realizada com sucesso, false caso
   contrário*/
   if (vazia(*q))
        return FALSE;

   item = q->A[(q->inicio+1) % tam_fila];
   return TRUE;
}
```

# Análise do tipo de Representação

- Vantagens da Fila Estática (Anel):
  - não envolve custos da alocação dinâmica
- Desvantagens da Fila Estática:
  - previsão de tamanho máximo

# Quando usar

- Representação Estática (Anel):
  - quando fila tiver tamanho pequeno ou seu comportamento for previsível

# Exercício

- Implemente um procedimento **reverso** que reposiciona os elementos na fila de forma que o início se torne fim e vice-versa. Use uma pilha.
  - I F → I F
  - 1 → 2 → 3 3 → 2 → 1

# Filas de Prioridade

- Filas em que a prioridade de remoção não é cronológica
  - Maior prioridade não é do elemento que ingressou primeiro
- Exemplos de Aplicações
  - Vôos lotados (*standby flyers*)
  - Listas de espera em geral (p. ex. transplantes)
  - Fila de processos para o Sistema Operacional
  - Ordenação

# TAD Fila de Prioridade

- Armazena **Itens**
- **Item**: par (chave, informação)
- Operações principais:
  - remove(F): remove e retorna o item com maior prioridade da fila F
  - insert(F, **x**): insere um item **x** = (k,e) com chave k