# FILAS (QUEUE)

Usamos filas no nosso dia-a-dia! Filas para caixas em supermercados e lojas, filas para atendimento em bancos, filas para atendimento em serviços públicos, etc. Além disso usamos filas no computador, por exemplo a fila de impressão!

Vamos compreender como funciona o Tipo Abstrato de Dados Fila!

Filas são estruturas sequenciais, onde novos elementos são sempre incluídos no final da fila e o elemento que será utilizado (atendido) será aquele que estiver no início da fila, sendo então removido dela. As filas atendem a um critério de funcionamento denominado FIFO (First-In-First-Out), ou seja, o primeiro a entrar na fila será o primeiro a sair. Uma fila inicia vazia, considere uma fila de banco quando este está recém abrindo, quando chega a primeira pessoa esta se torna a primeira e a última pessoa da fila. Na sequência, novas pessoas irão para o fim da fila e quando o caixa do banco for atender, atenderá a primeira pessoa que está na fila, então ela sai da fila e a próxima se torna a primeira da fila. Portanto precisamos saber quem está no início e quem está no fim da fila, estes são os dados básicos.

Assim como TAD PILHA, podemos implementar o TAD FILA utilizando vetores (que também pode ser chamado de alocação estática) ou encadeamento (alocação dinâmica). Porém, há alguns inconvenientes na implementação da FILA com vetores.

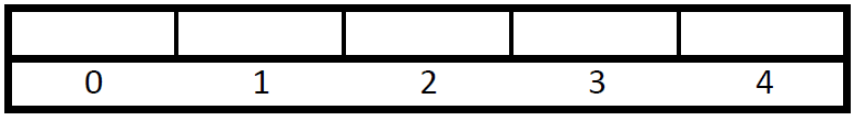
Normalmente filas não tem um limite, portanto fica difícil definir o tamanho do vetor que deverá conter a fila. Diferente do TAD PILHA em que a base da pilha está sempre na mesma posição e tanto as operações de empilhar como desempilhar são sempre realizadas no topo. No TAD FILA a operação de incluir um novo elemento sempre o faz no fim da fila, já a operação de remover um elemento da fila sempre se faz no início da fila, portanto sempre estaremos utilizando duas posições do vetor, uma indicando o início da fila e outra seu fim. E a principal característica a considerar é que o início da fila poderá passar a não coincidir com o início físico do vetor e isso torna esta

implementação mais complexa. Além disso, o fim da fila, também acabará por não coincidir com o fim físico do vetor. Isso ocorre porque vão sendo retirados elementos do início e estes espaços do vetor poderão ser aproveitados para inclusão de novos elementos.

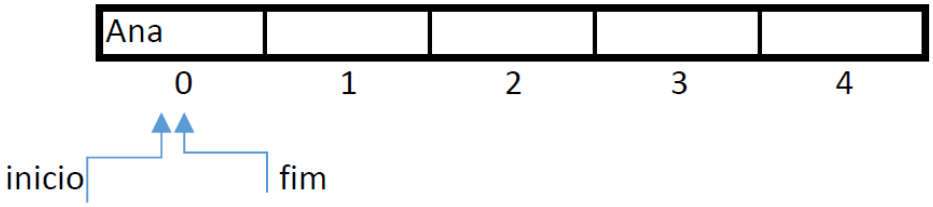
Para entender melhor essa situação, vejas a sequência de imagens a seguir que apresentam a estruturação de uma fila utilizando vetores.

Estamos utilizando um vetor de 5 posições, com os índices entre 0 e 4, e uma variável para conter o tamanho. Iniciamos com a fila vazia, tamanho = 0. Na sequência adicionaremos a cada etapa um elemento novo, no final da fila. Assim, percebas que o tamanho será incrementado e que o fim da fila será deslocado, sempre apontando para o último elemento que foi incluído. Fazemos isso até encher a fila. Depois passamos a atender pessoas da fila e removê-las, nesta situação a cada remoção o início da fila passar a ser alterado. E começam a existir espaços vagos no início físico do vetor. No fim, demonstramos o que irá acontecer ao adicionar mais um elemento na fila, que será o último, mas acabará sendo incluído no início do vetor, pois é a próxima posição livre. Terminando o nosso exemplo, o início da fila está na posição 3 do vetor e o fim da fila está na posição 0.

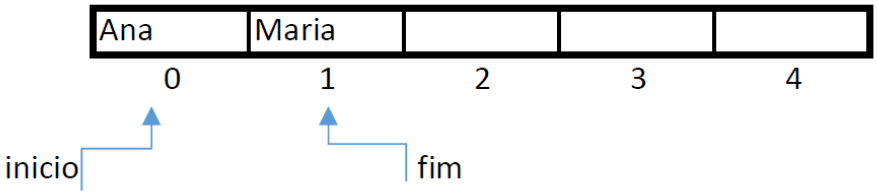
tamanho = 0 Fila Vazia



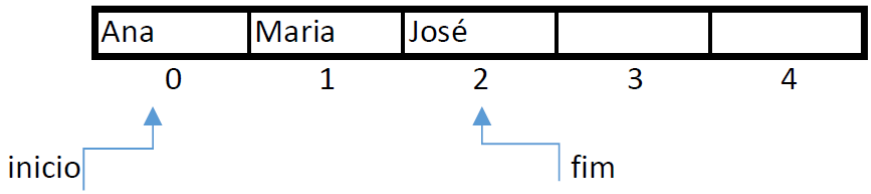

tamanho = 1 Inclusão do Primeiro Elemento



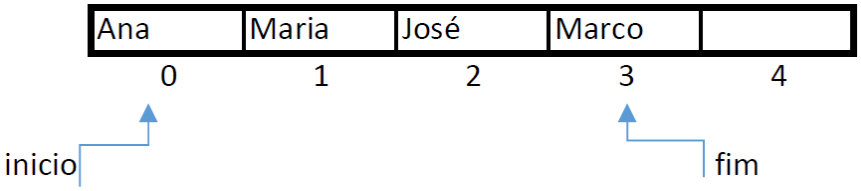

tamanho = 2 Inclusão do Próximo Elemento



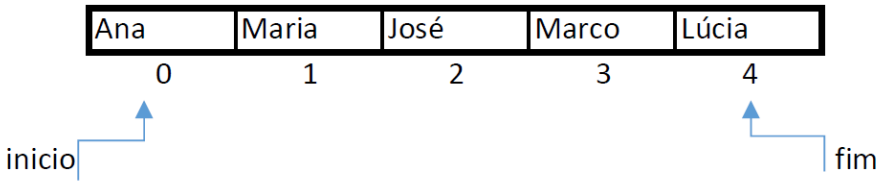
tamanho = 3 Inclusão do Próximo Elemento



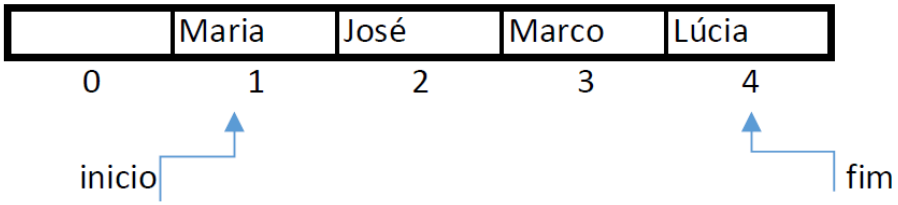

tamanho = 4 Inclusão do Próximo Elemento



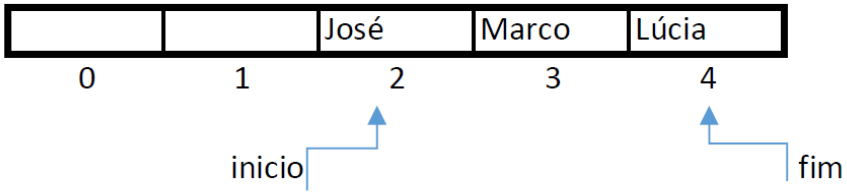

tamanho = 5 Inclusão do Próximo Elemento

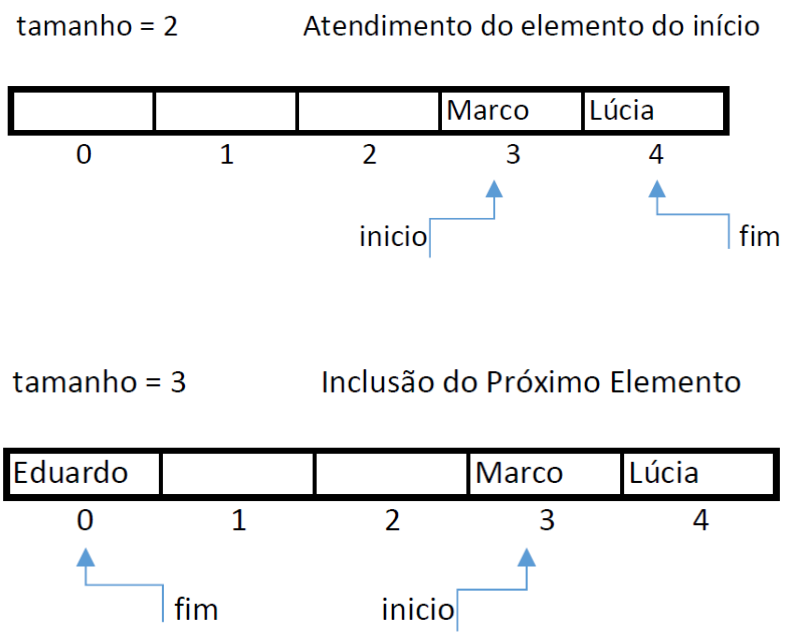



tamanho = 4 Atendimento do elemento do início




tamanho = 3 Atendimento do elemento do início






Fonte: autor

O vetor não é uma estrutura de dados flexível, pois precisamos definir a quantidade máxima de elementos no código do programa, que se tornará fixa nas execuções. Se o número de elementos exceder o limite previsto para o vetor, teremos um problema. Por outro lado, se esse número de elementos for muito menor que o limite, teremos desperdício de espaço de memória reservado para o vetor.

Normalmente, a intenção de utilizar vetores para implementação do TAD é oferecer simplicidade e facilidade de desenvolvimento, mas isto não ocorrerá com a implementação do TAD FILA com vetores. Portanto, optaremos por não trabalhar a implementação dessa alternativa neste livro.

## IMPLEMENTAÇÃO DO TAD FILA COM ENCADEAMENTO


Veremos que a implementação do TAD FILA utilizando encadeamento é semelhante a implementação do TAD PILHA com encadeamento.

A diferença está nas informações que são necessárias para gerenciar a fila, pois agora precisamos saber o endereço do **início e fim da fila** e também seu **tamanho**, esse é importante para que não tenhamos que varrer toda a fila para obter essa informação. Esses dados formarão o que denominamos cabeçalho da fila (*queue header*, em inglês). Desta forma podemos definir a estrutura deste cabeçalho na linguagem C conforme segue:

```
struct fila {
  int tamanho;
  nodo* inicio, fim;
};
typedef struct fila Fila;//estrutura do tipo pilha
```

Fonte: autor

Já a estrutura do nodo da fila fica muito semelhante ao nodo da pilha, a diferença aqui é o ponteiro para o endereço, que neste caso denominamos próximo, pois irá apontar para o próximo elemento da fila. Por questões de simplicidade, estamos considerando que o conteúdo do elemento é apenas um valor real (float).

```
typedef struct nodo{
 float  elemento;
 nodo* proximo;
};
```

Fonte: autor

Precisamos definir a interface de funções do TAD FILA o que pode ser visualizado a seguir:

```
Fila* cria ();//função para criação da fila
void insere (Fila* f, float v);//função para incluir v na fila f
float remove (Fila* f);//função que retorna o valor do início da fila f
e o exclui
int vazia (Fila* f);//função que testa se a fila está vazia
void libera (Fila* f);//função que libera a memória
int elementos(Fila* f);//função que retorna o tamanho da fila
```

Fonte: autor

Vamos analisar a implementação de cada uma dessas funções.

A função para criação da fila, deve alocar o cabeçalho da fila e inicializar os seus campos, de maneira que tamanho = 0 (zero elementos) e início e fim recebem NULL (nulo) indicando que não apontam para nodos.

```
//função para criação da fila
Fila* cria (){
 Fila* f = (Fila) malloc (sizeof (Fila));//aloca cabeçalho
 f->tamanho = 0;
 f->inicio = NULL;
 f->fim = NULL;
 return(f);
}
```

Fonte: autor

A função que inclui elementos no final da fila deve alocar o novo nodo (verificando se há memória suficiente para este processo), depois o novo elemento pode ser encadeado. Para isso, é necessário que o último elemento da fila passa a apontar para esse novo elemento. Mas a exceção é quando a fila está vazia, pois neste caso não há último elemento, então o início e fim apontarão para o novo elemento, que será o primeiro da fila. Em ambas as situações, o fim da fila para ser o novo elemento.

```
//função para incluir v na fila f
void insere (Fila* f, float v){
  nodo* anterior;
  nodo* novo = (nodo) malloc (sizeof (nodo));
  if (novo == NULL){
     printf (“Memoria insuficiente!!”);
  }
  else {
    novo -> elemento = v;
    novo -> proximo = NULL;//o novo elemento será o último da fila
    f->tamanho = f->tamanho + 1;//incrementa tamanho
    //agora temos que encadear a fila
    if (f->inicio == NULL){//fila está vazia
      f->inicio = novo;
      f->fim = novo;
    }
    else{
    anterior = f->fim;
    anterior->próximo = novo;//o último elemento aponta para o novo
    f->fim = novo;//o novo elemento se torna o novo fim da fila
    }
  }
}
```

Fonte: autor

A função que remove o elemento do início da fila, precisa verificar se a fila possui elementos, não está vazia. Se tiver elementos, é necessário guardar o ponteiro do primeiro elemento para que possa ser liberado da memória. Além disso, é necessário atualizar o início da fila, que será o próximo (o seguinte) da fila. Deve-se ter o cuidado de verificar se a fila ficou vazia, ou seja, se ao remover o elemento não restaram elementos na fila, neste caso será necessário atualizar o fim da fila. Por fim libera-se o ponteiro do elemento removido e é retornada a sua informação. Vejas sua implementação a seguir:

```
//função que retorna o valor do início da fila f e o exclui
float remove (Fila* f){
  nodo* primeiro;
  float v;
  if (vazia(f)){
     printf (“Fila Vazia!!”);
     return(0);
  }
  else {
    primeiro = f->inicio;
    v = primeiro->elemento;
    //agora temos que atualizar o novo início da fila
    f->inicio = primeiro->proximo;
    f->tamanho = f->tamanho - 1;//decrementa tamanho
    if (f->inicio == NULL){//verifica se a fila ficou vazia
      f->fim = NULL;
    }
    free(primeiro);
    return(v);
  }
}
```

Fonte: autor

A função que testa se a fila está vazia é simples, basta testar se o inicio da fila é NULL (nulo), indicando que não tem elementos.

```
//função que testa se a fila está vazia
int vazia (Fila* f){
    if (f->inicio == NULL)
      return(1);
    else
      return(0);
}
```

Fonte: autor

Por fim, temos a função que libera memória, que irá liberar o espaço alocado pelo cabeçalho da fila.

```
//função que libera a memória
void libera (Fila* f){
    if (f != NULL)
      free(f);
}
```

Fonte: autor

Para sabermos quantos elementos estão armazenados na pilha, temos a função elementos que retorna a informação que está armazenada no campo tamanho do cabeçalho da pilha. Lembres que a função insere incrementa esse campo e que a função remove o decrementa.

```
//função que retorna o tamanho da fila
int elementos(Fila* f){
    if (f != NULL)
      return(f->tamanho);
    else
      return(0);
}
```

Fonte: autor

Esse conjunto de operações primitivas do TAD fila permite que diversas outras operações possam ser construídas utilizando-as como base. É importante que ao utilizar um TAD, isso sempre seja feito através de sua interface, ou seja, apenas utilizando as operações primitivas. Se necessário, pode-se ampliar a interface, adicionando novas operações. Quem utiliza o TAD não deve manipular a estrutura de armazenamento dele.

**Amplie seus conhecimentos!**

**Pesquise sobre o funcionamento da fila de impressão! Quais funcionalidades possui, quais informações são registradas de cada requisição de impressão. A partir disso, vejas que podem existir filas com prioridades, então pesquise e refita como poderia ser feito esse tipo de implementação!**

## 1.1 SUGESTÕES DE AUTOESTUDO

1. Qual a lógica de funcionamento do TAD FILA?
2. O que é o critério FIFO?
3. Quais as diferenças entra o TAD FILA e o TAD PILHA?
4. Explique como se poderia utilizar um vetor para implementar o TAD FILA?
5. Descreva quais os inconvenientes de utilizar vetor para implementar o TAD FILA?
6. Descreva a interface do TAD FILA?
7. Explique as operações primitivas do TAD FILA?
8. Você precisa construir uma função que esvazia a FILA, que pode conter diversos elementos, ou seja, enquanto a fila não estiver vazia você deve remover um elemento. Você **somente** pode utilizar as **operações primitivas** da fila. A assinatura desta função é:
   - void esvazia(Fila* f);
9. Construa, usando **somente operações primitivas** da fila, uma função que faça a cópia de uma fila f1 para uma fila f2. Utilize a seguinte assinatura para esta função:
   - void copia(Fila* f1, Fila* f2);
10. Se desejarmos procurar um elemento específico dentro da fila (por exemplo o elemento = 3), é possível fazer essa busca utilizando apenas as operações primitivas ou precisaríamos implementar uma nova operação primitiva? Por que?
11. Considere que estamos usando um sistema para controlar as filas de atendimento de caixas de um supermercado, onde cada pessoa entra em uma fila específica e para nosso exemplo temos apenas 2 filas f1 e f2, porém ocorreu um problema em um dos caixas e apenas um se manteve funcionando. Assim, agora você deve construir uma função que irá transferir os elementos da fila f2 colocando-os no fim da fila f1, para isto você apenas pode utilizar as operações primitivas do TAD fila, sem manipular diretamente sua estrutura interna. A assinatura desta função é:
    - void transferir(Fila* f1, Fila*f2);