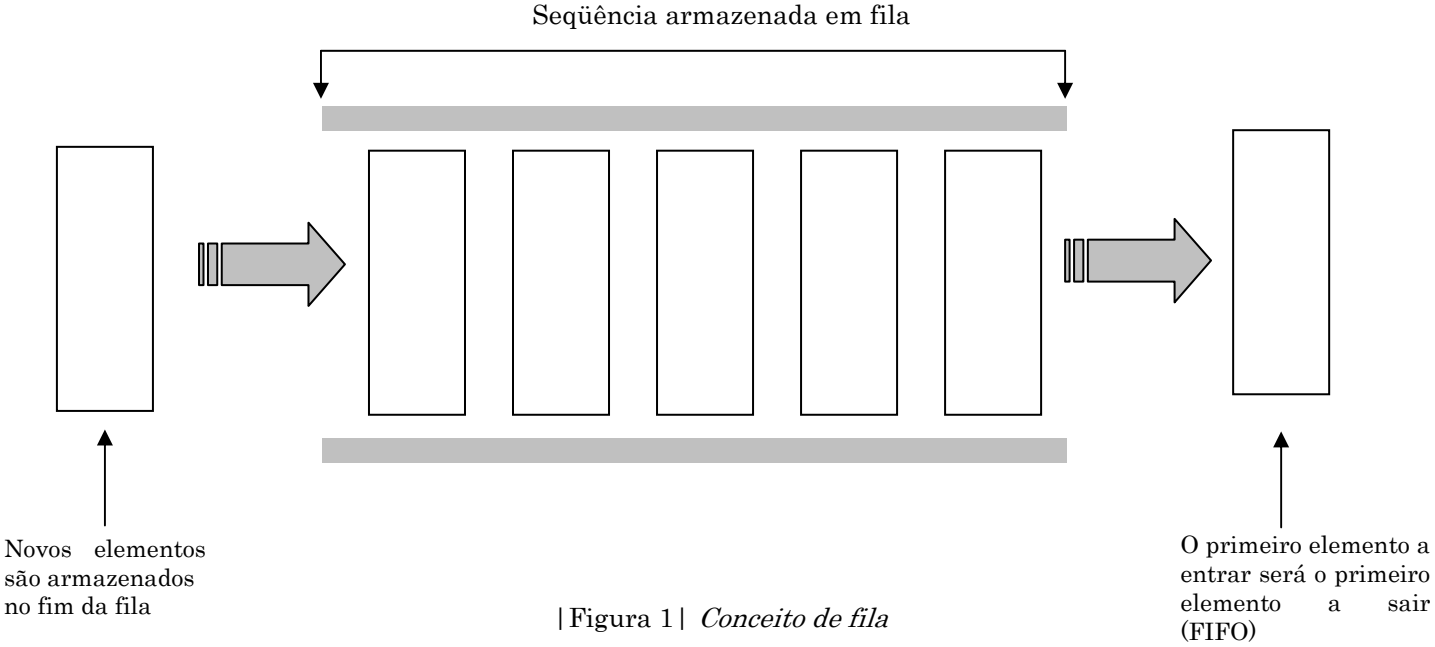
Lógica de Programação

FILAS

Acho que todos nós já ficamos em uma fila. Fila para comprar ingressos para shows, pegar dinheiro no banco e, às vezes, até para comprar o pãozinho da manhã. O conceito de fila em programação é o mesmo dessas filas em que esperamos para ser atendidos em ordem: o primeiro elemento a entrar na fila será o primeiro elemento a sair. Esse conceito é conhecido como 'First In, First Out' ou FIFO, expressão conhecida em português como PEPS ou 'Primeiro que entra, Primeiro que sai'. Então no conceito de fila, os elementos são atendidos, ou utilizados, seqüencialmente na ordem em que são armazenados.

As filas (*queues*) são conjuntos de elementos (ou listas) cujas operações de inserção são feitas por uma extremidade e de remoção, por outra extremidade.

Como exemplo pode-se implementar uma fila de impressão, em que os arquivos a ser impressos são organizados em uma lista e serão impressos na ordem de chegada, à medida que a impressora estiver disponível.



| Figura 1 | *Conceito de fila*

Conforme comentamos na introdução de listas, a implementação das listas, filas, pilhas e árvores pode ser feita por meio de arranjos ou de ponteiros. Ate agora fizemos as implementações utilizando ponteiros, então iremos exemplificar a implementação de filas e pilhas por meio de arranjos, uma vez que o exemplo pode ser facilmente adaptado para o uso de ponteiros.

> *Lembre-se: Ao implementarmos a fila por meio de arranjos, estaremos utilizando um vetor como contêiner para o armazenamento dos elementos que estarão na fila.*

Para definir a estrutura de uma fila, implementada por arranjo, é necessário construir um registro que contenha as informações da fila, como o início, o final e o contêiner de elementos, que é um vetor; nesse caso cada um dos elementos será representado por uma posição no vetor. Veja a estrutura:

EXEMPLO 1: PSEUDOCÓDIGO QUE REPRESENTA UMA FILA IMPLEMENTADA COM ARRANJO.

```
1.  Algoritmo Fila
2.  var
3.      Tipo fila_reg =  registro
4.                           inicio: inteiro
5.                           fim: inteiro
6.                           elemento: vetor [1.50] de inteiro
7.                       fim
8.      total: inteiro
9.      fila: fila_reg
10. inicio
11.     fila.inicio ← 0
12.     fila.fim ← 0
13.     total ← 0
```

Observe que a variável *elemento*, que é do tipo vetor, comporta 50 números inteiros. Essa característica é um limitador para trabalharmos com arranjos, pois podemos inserir apenas 50 valores!

Nota: Para implementar uma fila com o uso de ponteiros (alocação dinâmica), basta utilizar o nó e fazer as manipulações de acordo com o conceito PEPS.

Algoritmo que representa uma fila implementada com arranjo

```
14. Função vazia ( ): lógica
15.     inicio
16.       Se(total = 0) entao
17.           return .v.
18.       Senão
19.           return .f.
20.       fim-se
21.     fim
22. Função cheia ( ): lógica
23.     inicio
24.       Se(total >=50) então
25.           return .v.
26.       Senão
27.           return .f.
28.       fim-se
29.     fim
30. Procedimento enfileirar (elem: inteiro)
31.     inicio
32.       Se(cheia ( ) = .f..) então
33.          fila.elemento[inicio] ← elem
34.          fila.fim ← fila.fim + 1
35.          total ← total + 1
36.          Se(fila.fim >= 50) então
```

```
37.                fila.fim = 0
38.             fim-se
39.             Senão
40.                 Mostre("Fila cheia")
41.             fim-se
42.         fim
43. Funcao desenfileirar ( ): inteiro
44.         var
45.             excluido: inteiro
46.         inicio
47.             Se (vazia ( ) = .f.) então
48.                 excluido  ← fila.elemento[inicio]
49.                 fila.inicio ← fila.inicio + 1
50.                 Se (fila.inicio > = tamanho) então
51.                     fila.inicio ← 0
52.                 fim-se
53.                 total ← total – 1              retorne excluido
54.             Senão
55.             excluído ← nulo
56.             retorne excluído
57.             fim-se
58.         fim
59. Procedimento exibefila ( )
60.         var
61.             i: inteiro
62.         inicio
63.             Para (i ← 0 até total) faça
64.                 Mostre ("Posição ", i, " valor ", elemento [i] )
65.             fim-para
66.         fim
67. fim
```