

Capítulo 3

Filas com Alocação Sequencial e Estática



Seus objetivos neste capítulo

- Entender o que é e para que serve uma estrutura do tipo Fila.
- Desenvolver habilidade para implementar uma estrutura do tipo Fila, como um Tipo Abstrato de Dados (TAD), com Alocação Sequencial e Alocação Estática de Memória.
- Desenvolver habilidade para manipular Filas através dos Operadores definidos para o TAD Fila.
- Iniciar o desenvolvimento do seu segundo jogo, referente ao Desafio 2.

3.1 O que é uma Fila?

Quando temos um conjunto de pessoas esperando para serem atendidas, temos uma Fila. Fila de clientes no caixa do banco, fila no caixa do supermercado, fila de carros no pedágio, e assim por diante.

Em uma Fila séria, quem chega primeiro é atendido primeiro. Ou seja, novos elementos entram sempre no Final da Fila. E o elemento que sai é sempre o Primeiro da Fila.

Definição: Fila

Fila é uma estrutura para armazenar um conjunto de elementos que funciona da seguinte forma:

- Novos elementos entram no conjunto sempre no Final da Fila.
- O único elemento que pode ser retirado da Fila em determinado momento é o Primeiro elemento da Fila.

Do inglês Queue, F.I.F.O.

Uma Fila (em inglês, *Queue*) é uma estrutura que obedece ao critério *First In, First Out* (F.I.F.O.), ou seja, o primeiro elemento que entrou no conjunto será o primeiro elemento a sair do conjunto.

Figura 3.1 Definição de Fila.

Uma Fila é um conjunto ordenado de elementos. A ordem de entrada dos elementos determina sua posição no conjunto. Se temos três elementos em uma Fila (A, B e C) e se eles entraram na Fila na ordem A, depois B e depois C, sabemos que o Primeiro elemento da Fila é A. Sabemos também que, nesse momento, A é o único elemento que podemos retirar da Fila. Se, depois disso, quisermos inserir na Fila o elemento D, esse elemento D passará a ser o Último elemento da Fila ([Figura 3.2](#))

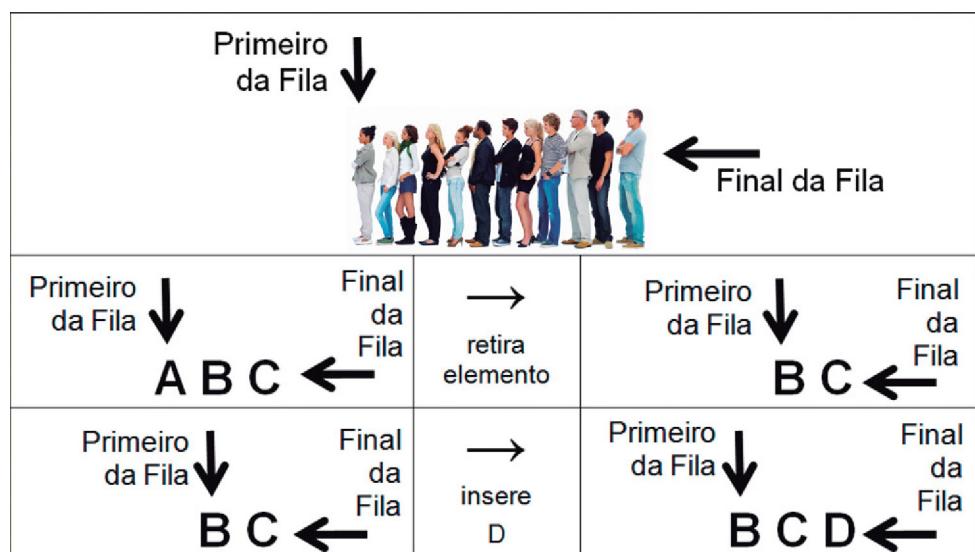


Figura 3.2 Retirando e inserindo elementos em uma Fila.

3.2 O jogo Snake

O seu segundo desafio é desenvolver uma adaptação do jogo Snake. Nas versões coloridas do *Snake*, uma *Cobra* é formada por pequenos *quadradinhos coloridos*. No decorrer do jogo, a Cobra vai “comendo” quadradinhos coloridos que estão na tela.

Na versão do *Snake* descrita no Desafio 2, ao comer um quadradinho colorido de cor diferente da cor da Cabeça da Cobra, o jogador marca pontos, e o quadradinho comido é inserido no final da Cobra ([Figura 3.3a](#)). Se a Cobra come um quadradinho colorido da mesma cor de sua Cabeça da Cobra, o quadradinho colorido que representa a Cabeça da Cobra é retirado ([Figura 3.3b](#)).

Inserir elementos sempre no Final da Cobra; retirar somente o Primeiro elemento da Cobra: essa Cobra funciona como uma Fila!

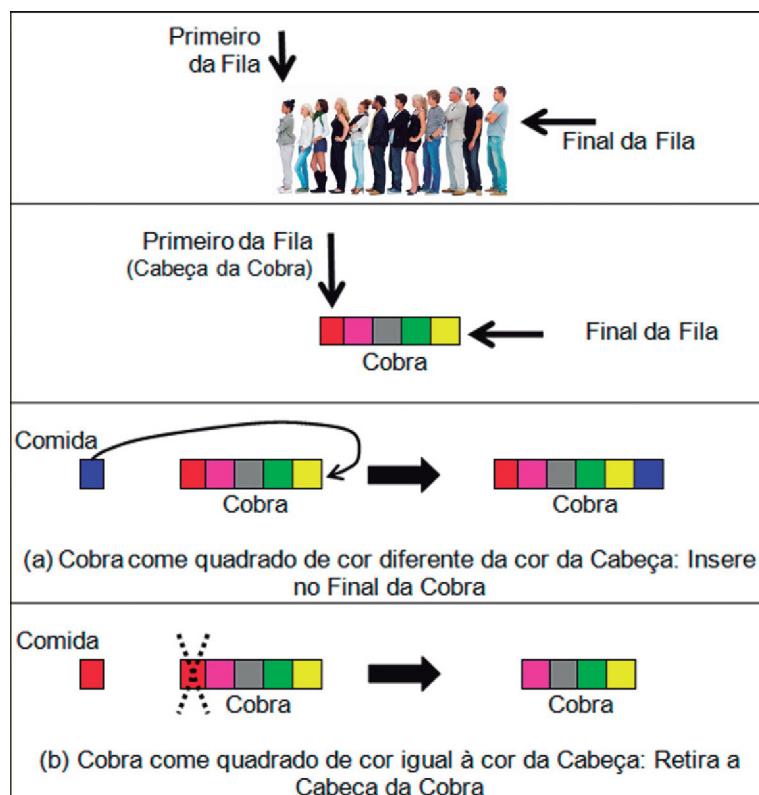


Figura 3.3 Projeto *Snake*: a Cobra como uma Fila.

3.3 Operações do tipo abstrato de Dados Fila

A partir da própria definição de Fila, já podemos identificar duas operações: inserir elemento na Fila e retirar elemento da Fila. A Operação para retirar um elemento da Fila deve retirar, precisamente, o Primeiro elemento. E a Operação para inserir um novo elemento deve inseri-lo, precisamente, no Final da Fila. A essas duas Operações vamos acrescentar outras três: Operação para Criar uma Fila, Operação para testar se a Fila está Vazia e Operação para testar se a Fila está Cheia. A [Figura 3.4](#) apresenta uma descrição dos parâmetros e do funcionamento das Operações Insere, Retira, Cria, Vazia e Cheia.

Operações e parâmetros	Funcionamento
Insere (F, X, DeuCerto)	Insere o elemento X, passado como parâmetro, na Fila F, também passada como parâmetro. O parâmetro DeuCerto indica se a execução da operação foi bem-sucedida ou não.
Retira (F, X, DeuCerto)	Retira o primeiro elemento da Fila F, passada no parâmetro, retornando o valor do elemento que foi retirado da Fila no parâmetro X. O parâmetro DeuCerto indica se a operação foi bem-sucedida ou não.
Vazia (F)	Verifica se a Fila F, passada como parâmetro, está vazia. Uma Fila vazia é uma Fila sem elementos.
Cheia (F)	Verifica se a Fila F, passada como parâmetro, está cheia. Uma Fila cheia é uma Fila em que não cabe mais nenhum elemento.
Cria (F)	Cria uma Fila F, iniciando sua situação como vazia.

Figura 3.4 Operações do TAD Fila.

Para elaborar os Exercícios 3.1 e 3.2, use os Operadores definidos na [Figura 3.4](#). Você terá que propor soluções sem saber como o Tipo Abstrato de Dados Fila é efetivamente implementado. Você encontrará uma solução para o Exercício 3.1 na sequência do texto, mas não consulte essa solução ainda. Tente propor uma solução. É assim que desenvolvemos habilidade para manipular Filas e outras estruturas pelos seus Operadores Primitivos ou, ainda, pelos “botões da televisão”.

Exercício 3.1 Junção dos elementos de duas Filas

Desenvolva um algoritmo para reunir os elementos de duas Filas F1 e F2 em uma terceira Fila F3. F3 deve conter todos os elementos de F1 seguidos de todos os elementos de F2. Considere que as Filas F1 e F2 já existem e são passadas como parâmetros. Preserve os elementos de F1 e F2: ao final da Operação, F1 e F2 devem possuir os mesmos elementos que possuíam no início da Operação, e os elementos devem estar na mesma ordem. Para propor a solução, utilize os Operadores da [Figura 3.4](#).

JunçãoDosElementos (parâmetro por referência F1, F2, F3 do tipo Fila);

```
/* recebe F1 e F2, e cria F3, com os elementos de F1 seguidos dos elementos de F2. Ao final da Operação, F1 e F2 mantêm seus elementos originais, na mesma ordem */
```

Conseguiu propor uma solução? Muito bom se conseguiu! Se não conseguiu, consulte a solução na [Figura 3.5](#) e fique atento ao modo como o exercício foi resolvido. A lógica do algoritmo é a seguinte: retiramos um a um todos os elementos da Fila F1. Cada elemento retirado de F1 é inserido em F3 e também em uma Fila Auxiliar. Depois os elementos retornam da Fila Auxiliar para F1. Fazemos o mesmo para F2 e, então, temos F1 e F2 em seu estado inicial (mesmos elementos, na mesma ordem), e F3 com os elementos de F1 seguidos dos elementos de F2.

```

JunçãoDosElementos (parâmetro por referência F1, F2, F3 do tipo Fila) {
    /* recebe F1 e F2, e cria F3 com os elementos de F1 seguidos dos elementos de F2. Ao
    final da Operação, F1 e F2 mantêm seus elementos originais, na mesma ordem */
    Variável ElementoDaFila do tipo Char;
    Variável FilaAuxiliar do tipo Fila;
    Cria (FilaAuxiliar);
    Cria (F3);

    /* Passando os elementos de F1 para F3 e também para FilaAuxiliar */
    Enquanto (Vazia(F1) == Falso) Faça { // enquanto F1 não for vazia...
        Retira(F1, X, DeuCerto);           // retira elemento de F1 - retorna valor em X
        Insere(F3, X, DeuCerto);          // insere X em F3
        Insere(FilaAuxiliar, X, DeuCerto); // insere X na FilaAuxiliar

    /* Retornando os elementos da FilaAuxiliar para F1 */
    Enquanto (Vazia(FilaAuxiliar) == Falso) Faça { // enquanto FilaAuxiliar tiver elemento
        Retira(FilaAuxiliar, X, DeuCerto);           // retira X da FilaAuxiliar
        Insere(F1, X, DeuCerto);                      // insere X em F1

    /* Passando os elementos de F2 para F3 e também para FilaAuxiliar */
    Enquanto (Vazia(F2) == Falso) Faça { // enquanto F2 não for vazia
        Retira(F2, X, DeuCerto);           // retira elemento de F2 - retorna valor em X
        Insere(F3, X, DeuCerto);          // insere X em F3
        Insere(FilaAuxiliar, X, DeuCerto); // insere X na FilaAuxiliar

    /* Retornando os elementos da FilaAuxiliar para F2 */
    Enquanto (Vazia(FilaAuxiliar) == Falso) Faça { // enquanto FilaAuxiliar tiver elemento
        Retira(FilaAuxiliar, X, DeuCerto);           // retira X da FilaAuxiliar
        Insere(F2, X, DeuCerto);                      // insere X em F2
    } /* fim do procedimento JunçãoDosElementos */
}

```

Figura 3.5 Junção dos elementos de duas Filas.

Note que ainda não temos a menor ideia de como as Operações Primitivas do TAD Fila (Insere, Retira, Vazia, Cheia e Cria) são efetivamente implementadas. Consideramos as Filas como caixas-pretas e manipulamos os valores armazenados apenas através de seus Operadores (ou “botões da televisão”). É assim que manipulamos estruturas de armazenamento de modo abstrato. É assim que utilizamos os Tipos Abstratos de Dados.

Proponha uma solução para o Exercício 3.2 com a mesma estratégia: manipulando os valores armazenados exclusivamente através dos Operadores do TAD Fila descritos na [Figura 3.4](#).

Exercício 3.2 Troca dos elementos entre duas Filas

Desenvolva um algoritmo para trocar os elementos de duas Filas F1 e F2. Ao final da operação, a Fila F2 deve conter os elementos que estavam em F1 antes do início da operação, na mesma ordem, e a Fila F1 deve conter os elementos que estavam em F2 antes do início da operação, também na mesma ordem. Considere que F1 e F2 já existem e são passadas como parâmetros. Para propor a solução, utilize os Operadores da [Figura 3.4](#).

```

TrocadosElementos (parâmetros por referência F1, F2 do tipo Fila);
/* recebe F1 e F2, e troca os elementos: ao final da operação, F1 deverá ter os elementos que estavam em F2 antes do inicio
da operação, na mesma ordem; e F2 deverá ter os elementos que estavam em F1 antes do inicio da operação, também
na mesma ordem */

```

3.4 Implementando um TAD Fila com Alocação Sequencial e Estática de Memória

No Capítulo 2 implementamos uma Pilha com Alocação Sequencial e Estática. Como poderíamos implementar uma Fila com a mesma técnica de alocação?

Conforme estudamos no Capítulo 2, na Alocação Sequencial os elementos do conjunto são armazenados juntos, em posições de memória adjacentes. A ordem dos elementos no conjunto é refletida na sequência em que são armazenados na memória ([Figura 3.6](#)).

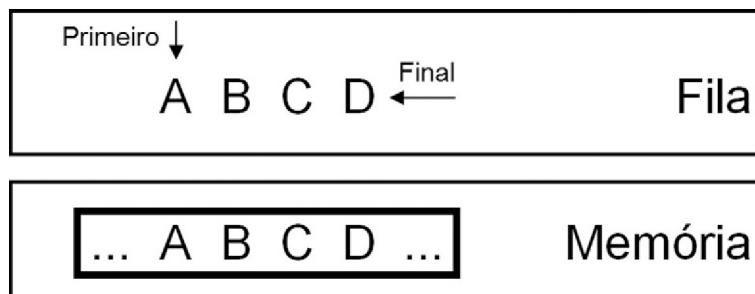


Figura 3.6 Fila com Alocação Sequencial — armazenamento em posições de memória adjacentes.

Na Alocação Estática de Memória, definimos previamente o número máximo de elementos que poderão fazer parte do conjunto e então reservamos espaço para todos eles.

Exercício 3.3 Esquema da implementação de uma Fila

Revise a Figura 2.11, que apresenta o esquema da implementação de uma Pilha com Alocação Sequencial e Estática. Em seguida, faça um esquema da implementação de uma Fila com Alocação Sequencial e Estática de Memória. Faça um desenho da estrutura adotada e descreva o funcionamento das Operações Insere, Retira, Cria, Vazia e Cheia. Não prossiga a leitura antes de refletir e propor uma solução.

3.4.1 Primeira solução: Fila com realocação dos elementos

A [Figura 3.7](#) apresenta o esquema de uma primeira implementação de Fila através de um vetor denominado Elementos e de uma variável denominada Último. A variável Último indica a posição do Último elemento da Fila no vetor. Nessa primeira implementação de Fila, o Primeiro elemento ficará sempre na primeira posição do vetor. Estamos considerando o início do vetor Elementos em 1. Mas o início do vetor poderá ser ajustado para zero caso este for o padrão na linguagem de programação escolhida.

Se quisermos inserir um elemento na Fila, colocaremos esse novo elemento no Final da Fila, ou seja, na posição Último + 1. Em seguida, o valor da variável Último precisa ser incrementado. A inserção de um elemento leva à situação da [Figura 3.8a](#) à situação da [Figura 3.8b](#).

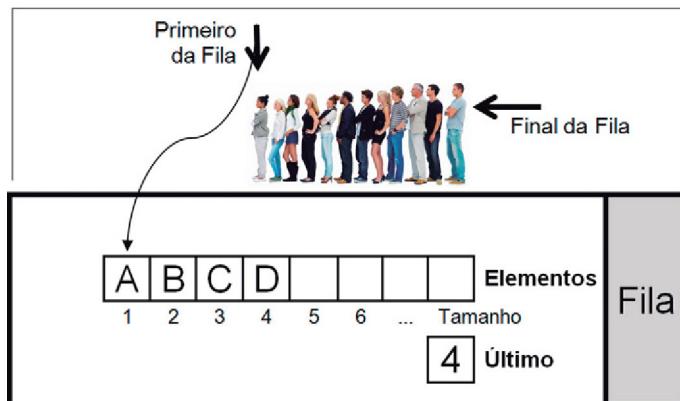


Figura 3.7 Esquema da implementação de uma Fila com Alocação Sequencial e Estática, e realocação de elementos.

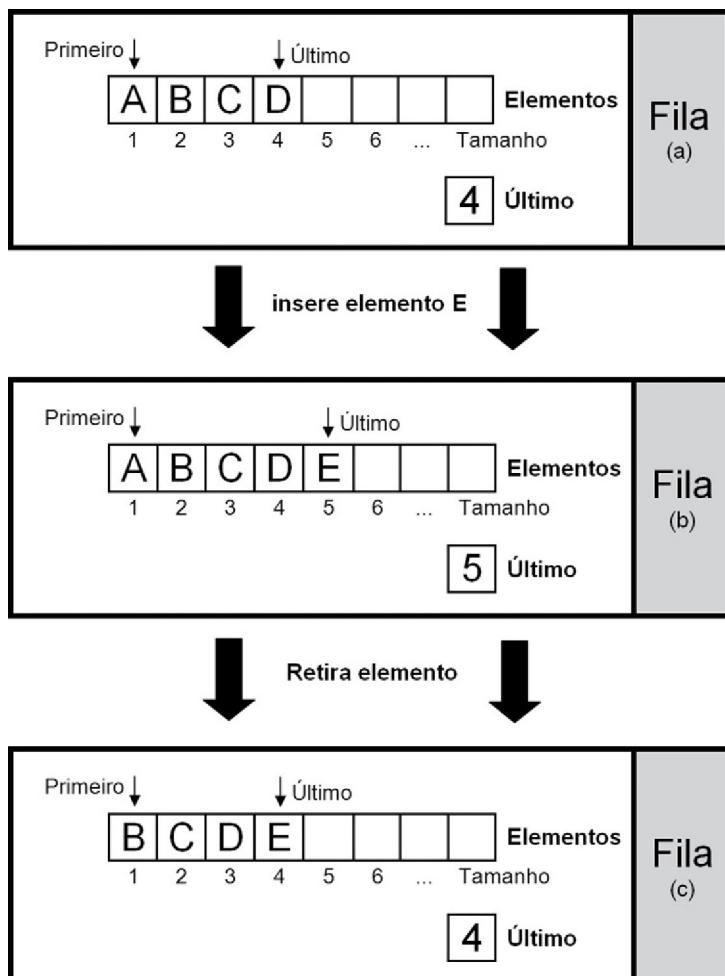


Figura 3.8 Implementação de Fila com realocação de elementos: Operações Insere e Retira.

Retirar um elemento de uma Fila implica, necessariamente, em retirar o Primeiro elemento. No exemplo da [Figura 3.8](#), inicialmente o Primeiro da Fila é o elemento de valor A ([Figuras 3.8a e 3.8b](#)). Se retirarmos A, o primeiro da Fila passará a ser o B. É assim que funciona uma Fila: atendimento por ordem de chegada — o primeiro que entra será sempre o primeiro a ser atendido e a sair da Fila.

Nessa primeira implementação de Fila estamos mantendo o primeiro elemento da Fila sempre na posição 1 do vetor. Assim, ao retirar A, teremos que mover o elemento de valor B para a posição 1. Analogamente, teremos que realocar cada um dos demais elementos da Fila para uma posição anterior à sua posição. Veja na [Figura 3.8c](#) como ficaria a Fila após a retirada do elemento de valor A. Note que o valor da variável Último precisou ser decrementado e que os elementos B, C, D e E tiveram que ser realocados devido à retirada de A. Por isso dizemos que essa primeira implementação de Fila é realizada com *realocação de elementos*.

Exercício 3.4 TAD Fila com realocação de elementos

Implemente o TAD Fila com realocação de elementos, segundo ilustrado pelas [Figuras 3.7 e 3.8](#). Implemente as operações Insere, Retira, Cria, Vazia e Cheia, conforme definidas na [Figura 3.4](#). Ao inserir, verifique se a Fila não está cheia; ao retirar, verifique se a Fila não está vazia.

3.4.2 Segunda solução: Fila sem realocação dos elementos

Para exercitar nossas habilidades, queremos desenvolver uma segunda implementação de Fila, também com Alocação Sequencial e Estática de Memória, mas sem realocação de elementos. Ou seja, queremos retirar um elemento da Fila, mas não queremos mover os demais elementos, como era necessário fazer em nossa primeira solução.

A realocação dos elementos era necessária porque fixamos o primeiro elemento da Fila na posição 1 do vetor. Se permitirmos que o primeiro elemento da Fila seja armazenado em qualquer posição do vetor, não precisaremos realocar os demais elementos a cada operação de retirada. A [Figura 3.9](#) mostra uma Fila que inicialmente contava com os

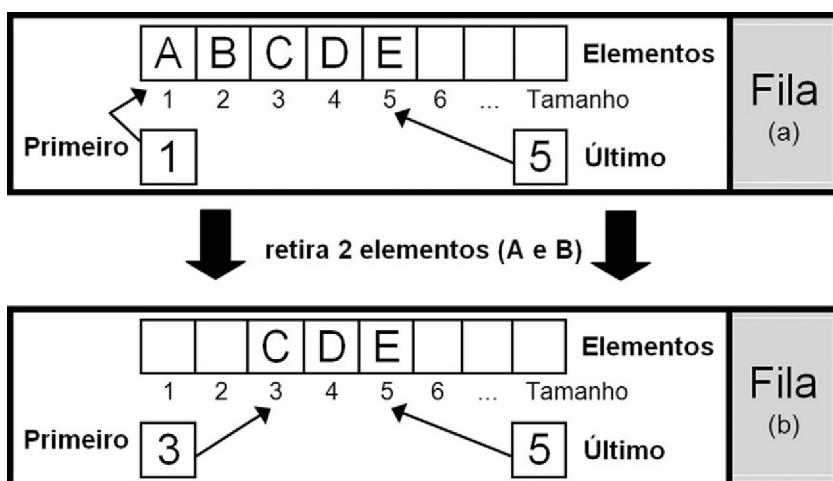


Figura 3.9 Operação Retira, sem realocação de elementos.

elementos A, B, C, D e E. Após a retirada de A e B, os elementos C, D e E não foram realocados. Note que o primeiro elemento da Fila não está mais armazenado na posição 1 do vetor, mas na posição 3, conforme indicado pela variável *Primeiro*.

Considerando que não temos a realocação de elementos, como podemos usar mais de uma vez os espaços do vetor *Elementos*? Por exemplo, na situação final da [Figura 3.9](#), as posições 1 e 2 já foram utilizadas para armazenar os elementos A e B, mas no momento estão vazias. Como poderemos utilizá-las novamente?

Que tal imaginar que o nosso vetor *Elementos* é circular? A [Figura 3.10](#) apresenta um esquema da implementação de uma Fila através de um vetor circular. Na prática, é o mesmo vetor que conhecemos, mas desenhado de modo mais conveniente para nossos objetivos do momento.

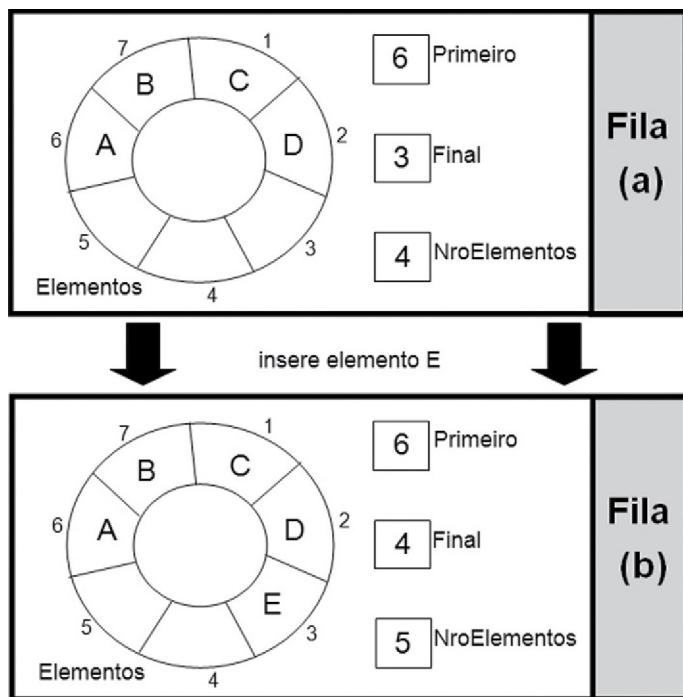


Figura 3.10 Fila em vetor circular: Operação Insere.

Nessa implementação, a Fila é composta pelo vetor *Elementos* e por outras três variáveis: *Primeiro*, *Final* e *NroElementos*. A variável *Primeiro* indica a posição do primeiro elemento da Fila no vetor. No exemplo da [Figura 3.10](#), *Primeiro* tem o valor 6, indicando que o primeiro elemento da Fila está armazenado na posição 6 do vetor.

A variável *Final* indica o final da Fila. Note que *Final* não indica o último elemento da Fila, mas a primeira posição após o último. No exemplo da [Figura 3.10a](#), o último elemento da Fila está na posição 2 do vetor e a variável *Final* tem o valor 3, ou seja, uma posição após o último.

A variável *NroElementos* indica o número total de elementos na Fila em determinado momento. Na situação da [Figura 3.10a](#), *NroElementos* tem valor 4, indicando que, naquele

momento, a Fila possui quatro elementos: A, B, C e D. Armazenar o número total de elementos na Fila nos ajudará a identificar a situação de Fila Cheia e a situação de Fila Vazia. Essa solução foi adotada por [Celes, Cerqueira e Rangel \(2004\)](#) e também por [Pereira \(1996\)](#).

Inserindo elementos na Fila

Para inserir um novo elemento na Fila esquematizada na [Figura 3.10a](#), basta atribuir o valor do novo elemento à posição 3 do vetor Elementos. Essa posição do vetor é indicada pela variável Final. Após isso é preciso atualizar o valor da variável Final e também o valor da variável NroElementos. Se partirmos da situação ilustrada na [Figura 3.10a](#), a inserção do elemento E resultará na situação ilustrada na [Figura 3.10b](#). A variável NroElementos passou a ter o valor 5, indicando que a Fila agora possui cinco elementos. A variável Final passou a ter o valor 4, indicando que, se quisermos inserir outro elemento, deveremos inseri-lo na posição 4.

Exercício 3.5 Operação Insere na Fila

Conforme especificado na [Figura 3.4](#), a operação Insere recebe como parâmetros a Fila na qual queremos inserir um elemento e o valor do elemento que queremos inserir. O elemento só não será inserido se a Fila já estiver cheia. Você encontrará a seguir uma possível solução para este exercício, mas tente propor uma solução antes de consultar a resposta.

Insere (parâmetro por referência F do tipo Fila, parâmetro X do tipo Char, parâmetro por referência DeuCerto do tipo Boolean);
/* Insere o elemento X na Fila F. O parâmetro DeuCerto deve indicar se a operação foi bem-sucedida ou não. A operação só não será bem-sucedida se tentarmos inserir um elemento em uma Fila cheia */

A [Figura 3.11](#) apresenta um algoritmo conceitual para a Operação Insere. Se a Fila F estiver cheia, sinalizamos através do parâmetro DeuCerto que o elemento X não foi inserido. Mas, se a Fila F não estiver cheia, X será inserido. Para isso, incrementamos o número de elementos ($\text{NroElementos} = \text{NroElementos} + 1$) e armazenamos X no vetor Elementos, na posição indicada pela variável Final ($\text{F.Elementos}[\text{F.Final}] = \text{X}$). Ou seja, o elemento X entra no Final da Fila F.

```
Insere (parâmetro por referência F do tipo Fila, parâmetro X do tipo Char, parâmetro por referência DeuCerto do tipo Boolean) {
    /* Insere o elemento X na Fila F. O parâmetro DeuCerto deve indicar se a operação foi bem-sucedida ou não. A operação só não será bem-sucedida se tentarmos inserir um elemento em uma Fila cheia */

    Se (Cheia(F)== Verdadeiro)           // se a Fila F estiver cheia...
        Então DeuCerto = Falso;          // ... não podemos inserir
    Senão { DeuCerto = Verdadeiro;       // inserindo X na Fila F... operação deu certo..
            F.NroElementos = F. NroElementos + 1; // aumenta o número de elementos
            F.Elementos[ F.Final ] = X;           // insere X no final da Fila F
            // avançando o apontador Final... Atenção: o vetor é circular
            Se (F.Final == Tamanho) // se Final estiver na última posição do vetor..
                Então F.Final = 1;      // ..avança para a primeira posição (ajustar para 0?)
            Senão F.Final = F.Final + 1; // senão avança para a próxima posição
        }; // fim do senão
    } // fim da operação Insere
```

Figura 3.11 Insere em uma Fila — vetor circular.

Após inserir X na posição indicada pela variável Final, precisamos atualizar o valor da variável Final. Se a variável Final não estiver apontando para a última posição do vetor, passará a apontar para a posição seguinte à sua posição atual. É o que acontece no exemplo da [Figura 3.10](#). Como estamos tratando o vetor como circular, se a variável Final estivesse apontando para a última posição do vetor deveria passar a apontar para a primeira posição do vetor. Na [Figura 3.10](#), a primeira posição do vetor é 1 e a última posição é 7. Pode ser conveniente ajustar esses valores para 0 e 6, respectivamente, para compatibilização com o padrão da linguagem de programação escolhida.

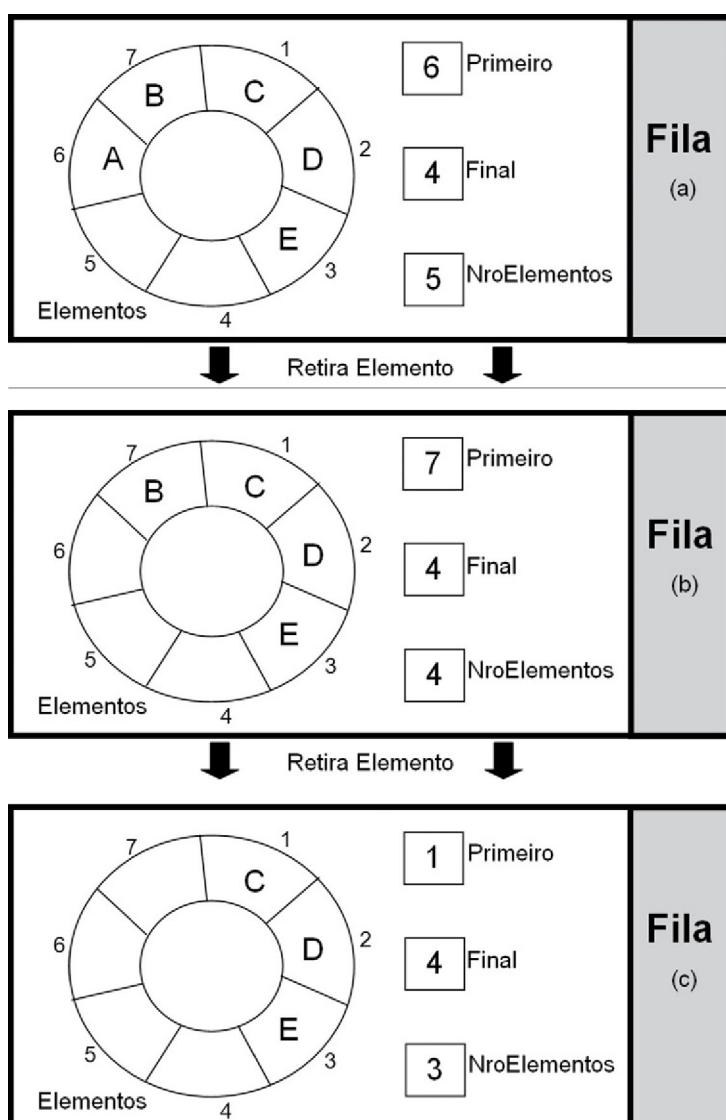


Figura 3.12 Avançando os apontadores Primeiro e Final em um vetor circular: após a Última posição do vetor, retorna para a Primeira posição.

Retirando elementos da Fila

A [Figura 3.12](#) exemplifica a operação que retira um elemento da Fila. A partir da situação inicial ([Figura 3.12a](#)), retiramos um elemento — o elemento de valor A — e chegamos à situação retratada na [Figura 3.12b](#). Ao retirar A, diminuímos o número de elementos da Fila de cinco para quatro e avançamos o apontador Primeiro da posição 6 para a posição 7. Se retirarmos mais um elemento — agora o elemento de valor B —, chegaremos à situação da [Figura 3.12c](#). O número de elementos passou de quatro para três, e o apontador Primeiro avançou de 7 para 1. Estamos tratando o vetor Elementos como um vetor circular. Se o apontador Primeiro ou o apontador Final estiver na última posição do vetor, “avançar” significa retornar à primeira posição.

Exercício 3.6 Operação Retira da Fila

Conforme especificado na [Figura 3.4](#), a operação Retira recebe como parâmetro a Fila da qual queremos retirar um elemento. Caso a Fila não estiver vazia, a operação retorna o valor do elemento retirado. O elemento a ser retirado deve ser sempre o primeiro da Fila.

Retira (parâmetro por referência F do tipo Fila, parâmetro por referência X do tipo Char, parâmetro por referência DeuCerto do tipo Boolean);

/* Caso a Fila F não estiver vazia, retira o primeiro elemento da Fila e retorna o seu valor no parâmetro X. Se a Fila F estiver vazia, o parâmetro DeuCerto retorna Falso */

Operações Cria, Vazia e Cheia

A [Figura 3.13](#) ilustra a situação em que a Fila está vazia (2.13a) e a situação em que a Fila está cheia (2.13b). Em uma Fila vazia, o número de elementos é zero, e em uma Fila cheia, o número de elementos é igual ao Tamanho da Fila ou, ainda, igual ao tamanho do vetor. Na [Figura 3.13b](#), o tamanho do vetor é 7, e o número de elementos também é sete. Logo, a Fila está cheia.

Conforme já mencionamos, armazenar o número total de elementos na Fila nos ajuda a identificar a situação de Fila Cheia e a situação de Fila Vazia. Essa solução foi adotada por [Celes, Cerqueira e Rangel \(2004\)](#) e por [Pereira \(1996\)](#).

Exercício 3.7 Operação Cria a Fila

Conforme especificado na [Figura 3.4](#), criar a Fila significa inicializar os valores de modo a indicar que a Fila está vazia.

Cria (parâmetro por referência F do tipo Fila);

/* Cria a Fila F, inicializando a Fila como vazia - sem nenhum elemento. */

Exercício 3.8 Operação para testar se a Fila está vazia

Conforme especificado na [Figura 3.4](#), a operação Vazia testa se a Fila passada como parâmetro está vazia (sem elementos), retornando o valor Verdadeiro (Fila vazia) ou Falso (Fila não vazia).

Boolean Vazia (parâmetro por referência F do tipo Fila);

/* Retorna Verdadeiro se a Fila F estiver vazia - sem nenhum elemento; retorna Falso caso contrário */

Exercício 3.9 Operação para testar se a Fila está cheia

Conforme especificado na [Figura 3.4](#), a operação Cheia testa se a Fila passada como parâmetro está cheia. A Fila estará cheia se na estrutura de armazenamento não couber mais nenhum elemento.

Boolean Cheia (parâmetro por referência F do tipo Fila);

/* Retorna Verdadeiro se a Fila F estiver cheia, ou seja, se na estrutura de armazenamento não couber mais nenhum elemento; Retorna Falso caso contrário */

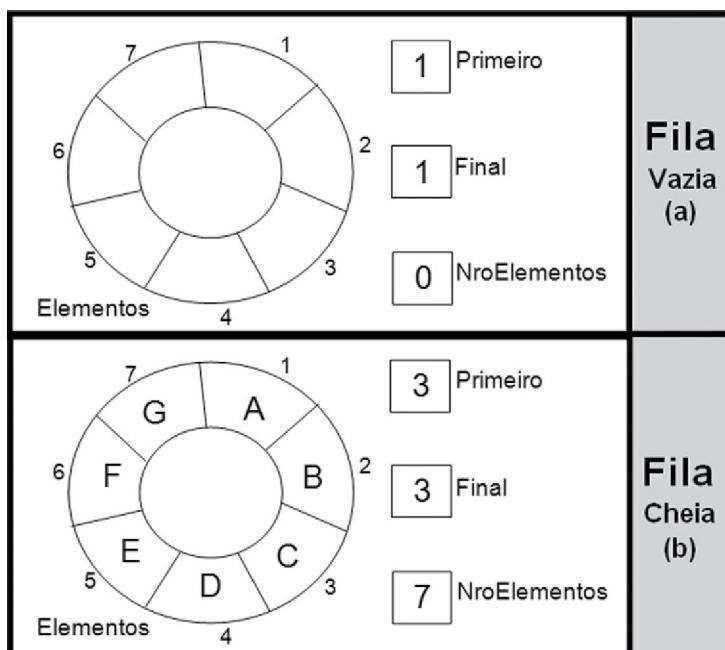


Figura 3.13 Fila Vazia e Fila Cheia em um vetor circular.

Exercício 3.10 Fila em vetor circular sem o total de elementos

Nos exercícios anteriores, implementamos uma Fila com um vetor circular e utilizamos a variável Total para indicar o total de elementos. Seria possível implementar uma Fila com vetor circular sem armazenar o total de elementos? Como você faria para diferenciar a situação de Fila Vazia e a situação de Fila Cheia?

Exercício 3.11 Número de elementos de uma Fila

Desenvolva um algoritmo para calcular o número de elementos de uma Fila F. Considere que a Fila F já existe, ou seja, não precisa ser criada. Você pode criar Filas auxiliares, se necessário. Ao final da execução dessa Operação, F precisa estar exatamente como estava no início da Operação — os mesmos elementos, na mesma ordem. Considere que a Fila F possui elementos do tipo Char.

```
Int NúmeroDeElementos(parâmetro por referência F do tipo Fila);
/* retorna o número de elementos da uma Fila F, passada como parâmetro */
```

3.5 Botão da televisão ou chave de fenda?

Ao propor uma solução para o Exercício 3.11, você usou o botão de volume ou uma chave de fenda?

A [Figura 3.14](#) apresenta duas possíveis soluções para o Exercício 3.11. A primeira solução é bem simples, mas é dependente da implementação: só funciona na implementação da Fila com vetor circular, que fizemos na Seção 3.4.2. Não funciona, por exemplo, na implementação de Fila com realocação de elementos que fizemos na Seção 3.4.1, pois nessa implementação não temos a variável que armazena o total de elementos.

```
Int NúmeroDeElementos (parâmetro por referência F do tipo Fila) {
    /* retorna o número de elementos de uma Fila F, passada como parâmetro */

    /* solução 1: dependente da implementação - abre a televisão para aumentar o volume */

    Retorne F.Total // F.Total armazena o total de elementos da Fila

    /* solução 2: independente da implementação - aumenta o volume pelo botão de volume*/

    Variável X do tipo Char; // elemento da Fila
    Variável Contador do tipo Int;
    Variável FilaAuxiliar do tipo Fila;

    Contador = 0;
    Cria (FilaAuxiliar);

    /* Contando... retira um a um os elementos de F, atualizando o Contador */
    Enquanto (Vazia(F) == Falso) Faça { // enquanto F não for vazia
        Retira(F, X, DeuCerto); // retira X de F
        Contador = Contador + 1; // incrementa o Contador
        Insere(FilaAuxiliar, X, DeuCerto); } // insere X na FilaAuxiliar

    /* Retornando os elementos da FilaAuxiliar para F */
    Enquanto (Vazia(FilaAuxiliar) == Falso) Faça { // enquanto FilaAuxiliar tiver elemento
        Retira(FilaAuxiliar, X, DeuCerto); // retira X da FilaAuxiliar
        Insere(F, X, DeuCerto); } // insere X em F

    /* Dando o resultado */
    Retorne Contador;

} /* fim do procedimento NúmeroDeElementos */
```

Figura 3.14 Número de elementos de uma Fila: duas soluções.

A segunda solução é independente da implementação, pois manipula a Fila exclusivamente através de seus Operadores Primitivos, definidos na [Figura 3.4](#). Essa segunda solução funciona tanto na implementação de Fila com realocação de elementos quanto na implementação de Fila com vetor circular. Por ser independente da implementação, essa segunda solução proporciona maior portabilidade e reusabilidade de código.



O Exercício 3.11 é um exemplo cruel, pois a solução dependente da implementação é realmente muito simples e óbvia, mas abre a televisão com uma chave de fenda para aumentar o volume e, por isso, a portabilidade e o potencial para reuso de software são menores.

Botão do volume ou chave de fenda? Reflita e escolha a melhor estratégia caso a caso, ao propor soluções para os exercícios seguintes.

No final deste capítulo são apresentadas respostas ou sugestões para alguns dos exercícios propostos, mas para desenvolver as habilidades pretendidas, proponha suas próprias soluções antes de consultar as respostas e sugestões.

Exercício 3.12 Mesmo número de elementos?

Desenvolva um algoritmo para testar se uma Fila F1 possui o mesmo número de elementos que uma Fila F2. Considere que as Filas F1 e F2 já existem e são passadas como parâmetro. Considere que as Filas F1 e F2 possuem elementos do tipo Char.

```
Boolean MesmoNúmeroDeElementos (parâmetros por referência F1, F2 do tipo Fila);
/* retorna Verdadeiro se as Filas F1 e F2 tiverem o mesmo número de elementos */
```

Exercício 3.13 As Filas são iguais?

Desenvolva um algoritmo para testar se duas Filas F1 e F2 são iguais. Duas Filas são iguais se possuem os mesmos elementos, exatamente na mesma ordem. Você pode utilizar Filas auxiliares, caso necessário. Considere que as Filas F1 e F2 já existem e são passadas como parâmetro. Considere que as Filas F1 e F2 possuem elementos do tipo Char.

```
Boolean Iguais (parâmetros por referência F1, F2 do tipo Fila);
/* retorna Verdadeiro se a Fila F1 for igual à Fila F2, ou seja, se as Filas possuírem exatamente os mesmos elementos, na mesma ordem. Se ambas as Filas forem vazias, devem ser consideradas iguais */
```

Exercício 3.14 Implementar e testar uma Fila

Implemente uma Fila em uma linguagem de programação como C ou C++. A Fila pode ser definida como uma Classe ou como um Tipo Estruturado. Os elementos da Fila devem ser do tipo Char. Implemente a Fila como um Tipo Abstrato de Dados. Em um arquivo separado, faça o programa principal bem simples, para testar o funcionamento das Operações Primitivas da Fila.

Exercício 3.15 Identificar outras aplicações de Filas

Identifique e liste aplicações de Filas. Identifique e anote alguns jogos que podem ser implementados com o uso de uma Fila e também outras aplicações. Sugestão de uso acadêmico: faça uma discussão em grupo. Ao final, cada grupo apresenta a todos os estudantes as aplicações que identificou. Pode ser uma boa fonte de ideias para definir bons jogos que também sejam aplicações de Filas.

3.6 Projeto *Snake*: Cobra como uma Fila de cores

Na versão do *Snake* descrita no Desafio 2, ao comer um quadradinho colorido de cor diferente da cor da Cabeça da Cobra, o jogador marca pontos, e o quadradinho comido é inserido no final da Cobra. Se a Cobra come um quadradinho colorido da mesma cor de sua Cabeça, o quadradinho colorido que representa a Cabeça da Cobra é retirado. A Cobra funciona como uma Fila de quadradinhos coloridos; uma Fila de cores (veja novamente a [Figura 3.3](#)).

Agora que você já sabe como usar uma Fila e já sabe também como implementar uma Fila, pode dar prosseguimento ao seu Projeto *Snake*. Adapte as regras do jogo, mas observe as seguintes recomendações na implementação do seu *Snake*:

- Armazene as cores que compõem a Cobra em um TAD Fila.
- A aplicação e o TAD Fila devem estar em arquivos separados.
- A aplicação (e outros módulos) deve manipular o TAD Fila exclusivamente através dos Operadores Primitivos: Empilha, Desempilha, Vazia etc. Aumente o volume da televisão apenas pelo botão de volume.
- Inclua no código do TAD Fila somente ações pertinentes ao armazenamento e recuperação das informações sobre a Cobra (quadradinhos coloridos que a compõem); faça o possível para deixar o TAD Fila o mais independente possível dos demais módulos; na medida do possível, não inclua no TAD Fila ações referentes à interface, por exemplo.
- Procure fazer a interface do modo mais independente possível dos demais componentes de software.

Exercício 3.16 Avançar o Projeto *Snake* — propor uma arquitetura de software

Proponha uma arquitetura de software para o seu projeto *Snake*, identificando os principais módulos do sistema e o relacionamento entre eles. Reveja a Figura 2.18, que trata da arquitetura do jogo *FreeCell*, e adapte conforme necessário para o seu novo projeto. Sugestão para uso acadêmico: desenvolva o projeto em grupo. Promova uma discussão entre o grupo, defina a arquitetura do software e uma divisão de trabalho entre os componentes do grupo.

Exercício 3.17 Avançar o Projeto *Snake* — definindo o tipo do Elemento da Cobra e o tipo de Comida da Cobra

Defina como você representará o Elemento da Cobra. Ele armazena, em essência, uma cor. Defina um conjunto de cores que você gostaria de utilizar (amarelo, vermelho, azul, verde, cinza etc.) e um modo de representar essas cores em seu programa. Lembre-se de que a Comida da Cobra será do mesmo tipo do Elemento da Cobra. Em algum momento, você terá que comparar a cor da Comida da Cobra com a cor da Cabeça da Cobra, ou seja, a cor do primeiro elemento da Fila que representa a Cobra. Não se preocupe com a interface em si ao definir o tipo do Elemento da Cobra. Procure implementar a interface como um módulo independente, o tanto quanto possível.

Exercício 3.18 Avançar o Projeto Snake — implemente uma Fila de Elementos Coloridos

Defina e implemente uma Fila de Elementos Coloridos. Os elementos da Fila devem ser do tipo definido no Exercício 3.18. Faça um programa para testar sua Fila sem se preocupar com a qualidade da interface neste momento.

Exercício 3.19 Avançar o Projeto Snake — defina as regras, escolha um nome e inicie o desenvolvimento do seu Snake

Defina as regras da sua variação do *Snake*. Altere as regras em algum aspecto. Dê personalidade própria ao seu *Snake*; escolha para o seu jogo um nome representativo e interessante. Escreva as regras da sua versão do *Snake* e coloque em um arquivo-texto que possa ser utilizado na documentação do jogo ou em uma opção de Ajuda/Como Jogar. Sugestão para uso acadêmico: desenvolva o Projeto *Snake* em grupo. Tome as principais decisões em conjunto e divida o trabalho entre os componentes do grupo, cada qual ficando responsável por parte das atividades.

Exercício 3.20 Avançar o Projeto Snake — projeto da interface



Avance um pouco mais no estudo da biblioteca gráfica que você escolheu para implementar uma interface visual e intuitiva para sua versão do *Snake*. Você pode iniciar seu estudo pelo Tutorial de Programação Gráfica disponível nos Materiais Complementares. Fique livre para estudar outros materiais e para adotar outras ferramentas gráficas para o seu projeto. Projete a interface do modo mais independente possível dos demais módulos. Se estiver trabalhando em grupo, um dos membros pode se responsabilizar por estudar e desenvolver os aspectos referentes à interface e depois ajudar os demais componentes nesse aprendizado. Você pode optar também por implementar primeiramente uma interface bem simples, textual, para testar os demais componentes do jogo e depois substituir por uma interface mais sofisticada.

Comece a desenvolver o seu *Snake* agora!

Faça um *Snake* legal! Um *Snake* com a sua cara! Distribua para os amigos! Aprender a programar pode ser divertido!



Consulte nos Materiais Complementares

Tutorial de Programação Gráfica

Banco de Jogos: Adaptações do *Snake*

Banco de Jogos: Aplicações de Filas



<http://www.elsevier.com.br/edcomjogos>

Exercícios de fixação

Exercício 3.21 O que é uma Fila?

Exercício 3.22 Para que serve uma Fila? Em que tipo de situações uma Fila pode ser utilizada?

Exercício 3.23 Qual a diferença entre uma Pilha e uma Fila?

Exercício 3.24 Compare as implementações de Fila com realocação de elementos e em vetor circular. Na sua opinião, qual é mais interessante e por quê?

Exercício 3.25 Quando você propôs soluções para os Exercícios 3.1 (junção de duas Filas) e 3.2 (troca dos elementos entre duas Filas), nem sabia como o TAD Fila seria implementado. Logo, você teve que propor uma solução que manipulasse a Fila exclusivamente por seus Operadores Primitivos. Suponha agora que você conhece a implementação da Fila (vetor circular ou vetor com realocação de elementos). Que estratégia você escolheria para implementar os Exercícios 3.1 e 3.2? Você manteria a estratégia de manipular a Fila exclusivamente por seus Operadores Primitivos? Ou adotaria uma solução dependente da implementação? Justifique sua escolha.

Exercício 3.26 Na versão do *Snake* descrita no Desafio 2, no decorrer do jogo a Cobra comerá um quadradinho colorido e você precisará saber se esse quadradinho colorido é da cor da cabeça da Cobra ou não. Para isso, precisará saber a cor da cabeça da Cobra. Você tem duas alternativas para implementar uma operação que consulta o valor do primeiro elemento de uma Fila: pelo botão do volume (acionando as Operações Primitivas da Fila) ou abrindo a televisão (solução dependente da implementação). Qual dessas duas alternativas você considera mais interessante para o seu projeto e por quê?

Soluções para alguns dos exercícios

Exercício 3.3 Esquema da implementação de uma Fila

Veja a [Figura 3.7](#) (solução com realocação de elementos – compare com a Figura 2.11 – Pilha); e veja a [Figura 3.10](#) (solução sem realocação de elementos).

Exercício 3.6 Operação Retira da Fila

Retira (parâmetro por referência F do tipo Fila, parâmetro por referência X do tipo Char, parâmetro por referência DeuCerto do tipo Boolean) {

/* Caso a Fila F não estiver vazia, retira o primeiro elemento da Fila e retorna o seu valor no parâmetro X. Se a Fila F estiver vazia, o parâmetro DeuCerto retorna Falso */

Se (Vazia(F)==Falso)

Então DeuCerto = Falso; // se a Fila F estiver vazia, não podemos Retirar

Senão { // Retira elemento da Fila F e retorna o valor em X

DeuCerto = Verdadeiro; // a operação deu certo..

F.NroElementos = F.NroElementos - 1; // diminui o número de elementos

X = F.Elementos[F.Primeiro] = X; // X recebe valor do Primeiro elemento da Fila F

// avançando o apontador Primeiro. Atenção: o vetor é circular

Se (F.Primeiro == Tamanho) // se Primeiro estiver na última posição do vetor

Então F.Primeiro = 1; // primeira posição pode ser 0 - linguagem C

Senão F.Primeiro = F.Primeiro + 1; // avança para a próxima posição)

}

} // fim da operação Retira



Exercício 3.7 Operação Cria a Fila

```
Cria (parâmetro por referência F do tipo Fila) {  
    /* Cria a Fila F, inicializando a Fila como vazia - sem nenhum elemento. */  
    F.Primeiro = 1;  
    F.Final = 1;  
    F.NroElementos = 0;  
} // fim da operação Cria
```

Exercício 3.8 Operação para testar se a Fila está vazia

```
Boolean Vazia (parâmetro por referência F do tipo Fila) {  
    /* Retorna Verdadeiro se a Fila F estiver vazia - sem nenhum elemento; retorna Falso caso  
    contrário */  
    Se (F.NroElementos == 0)  
        Então Retorne Verdadeiro;  
    Senão Retorne Falso;  
} // fim da operação Vazia
```

Exercício 3.9 Operação para testar se a Fila está cheia

```
Boolean Cheia (parâmetro por referência F do tipo Fila) {  
    /* Retorna Verdadeiro se a Fila F estiver Cheia, ou seja, se na estrutura de armazenamento não  
    couber mais nenhum elemento; retorna Falso caso contrário */  
    Se (F.Total == Tamanho) // ou Tamanho -1, se vetor começar em zero  
        Então Retorne Verdadeiro;  
    Senão Retorne Falso;  
} // fim da operação Cheia
```

Exercício 3.12 Mesmo número de elementos?

Sugestão: calcule o número de elementos de cada Fila através da solução ao Exercício 3.11. Depois é só comparar o tamanho das duas filas.

Exercício 3.13 As Filas são iguais?

Sugestão: veja a solução proposta para o Exercício 2.4, que verifica se duas Pilhas são iguais.

Referências e leitura adicional

- Celes, W.; Cerqueira, R, Rangel, J. L. *Introdução a estruturas de dados*. Rio de Janeiro: Elsevier, 2004.
p. 171- 175.
Pereira, S. L. *Estruturas de dados fundamentais: conceitos e aplicações*. São Paulo: Érica, 1996. p. 64- 66.