

4

FILAS

Este capítulo explica a estrutura de dados fila, descrevendo as principais operações que esse tipo de dados suporta, e mostra como implementá-la com alocação dinâmica sequencial.

4.1 Fundamentos

Fila é uma lista em que as *inserções* são feitas num extremo, denominado *final*, e as *remoções* são feitas no extremo oposto, denominado *início*.

Quando um novo item é inserido numa fila, ele é colocado em seu final e, em qualquer instante, apenas o item no início da fila pode ser removido. Devido a essa política de acesso, os itens de uma fila são removidos na *mesma ordem* em que foram inseridos, ou seja, o primeiro a entrar é o primeiro a sair (Figura 4.1). Por isso, as filas também são denominadas listas FIFO (*First-In/First-Out*).

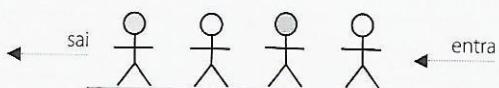


Figura 4.1 | Uma fila de pessoas: a primeira que entra é a primeira que sai.

A principal propriedade de uma fila é a sua capacidade de *manter a ordem* de uma sequência. Essa propriedade é útil em várias aplicações em computação.



Por exemplo, em um sistema operacional, cada solicitação de impressão de documento feita pelo usuário é inserida no final de uma fila de impressão. Então, quando a impressora fica livre, o gerenciador de impressão atende à próxima solicitação de impressão, removendo-a do início dessa fila. Assim, as solicitações de impressão são atendidas na mesma ordem em que elas são feitas.

Uma fila também é usada num sistema operacional para gerenciar a entrada de dados via teclado. À medida que as teclas são pressionadas pelo usuário, os caracteres correspondentes são inseridos numa área de memória chamada *buffer* de teclado. Então, quando um caractere é lido por um programa, por exemplo, com a função `getchar()`, declarada em `stdio.h`, o primeiro caractere inserido no *buffer* de teclado é removido e devolvido como resposta. Assim, os caracteres são processados na mesma ordem em que são digitados pelo usuário.

4.2 Operações em filas

Uma fila F suporta as seguintes operações:

- `fila(m)`: cria e devolve uma fila vazia F, com capacidade máxima m.
- `vaziaf(F)`: devolve 1 (*verdade*) se F está vazia; senão, devolve 0 (*falso*).
- `cheiaf(F)`: devolve 1 (*verdade*) se F está cheia; senão, devolve 0 (*falso*).
- `enfileira(x, F)`: insere o item x no final da fila F.
- `desenfileira(F)`: remove e devolve o item existente no início da fila F.
- `destroif(&F)`: destrói a fila F.

A Figura 4.2 mostra efeitos e resultados dessas operações numa fila F, denotada por uma lista com início no extremo esquerdo e final no extremo direito.

Operação	Fila F	Resultado
<code>F = fila(3)</code>	[]	-
<code>vaziaf(F)</code>	[]	1
<code>cheiaf(F)</code>	[]	0
<code>enfileira(1, F)</code>	[1]	-
<code>enfileira(2, F)</code>	[1, 2]	-
<code>enfileira(3, F)</code>	[1, 2, 3]	-
<code>vaziaf(F)</code>	[1, 2, 3]	0
<code>cheiaf(F)</code>	[1, 2, 3]	1
<code>desenfileira(F)</code>	[2, 3]	1
<code>desenfileira(F)</code>	[3]	2
<code>enfileira(4, F)</code>	[3, 4]	-
<code>enfileira(desenfileira(F), F)</code>	[4, 3]	-
<code>vaziaf(F)</code>	[4, 3]	0
<code>cheiaf(F)</code>	[4, 3]	0
<code>destroif(&F)</code>	inexistente	-

Figura 4.2 | Resultados e efeitos das operações em fila.

4.2.1 Cadeia palíndroma

Para exemplificar o uso de filas em programação, vamos criar um programa que verifica se uma cadeia é palíndroma. Uma cadeia é *palíndroma* se ela é igual à sua inversa (ignorando-se os espaços). Por exemplo, "ovo", "subi no onibus" "a sacada da casa" e "anotaram a data da maratona" são cadeias palíndromas.

Para obter a cadeia *direta*, sem os espaços, basta percorrer a cadeia original, da esquerda para a direita, inserindo numa *fila* cada letra encontrada. Então, quando essas letras forem removidas da fila, elas formarão uma cadeia na mesma ordem da cadeia original. Analogamente, para obter a cadeia *inversa*, basta percorrer a cadeia original, da esquerda para a direita, inserindo as letras numa *pilha*. Então, quando as letras forem removidas da pilha, elas formarão uma cadeia na ordem inversa da cadeia original. Portanto, comparando-se as letras removidas da fila e da pilha, podemos determinar se a cadeia original é ou não palíndroma. Essa ideia é ilustrada na Figura 4.3.

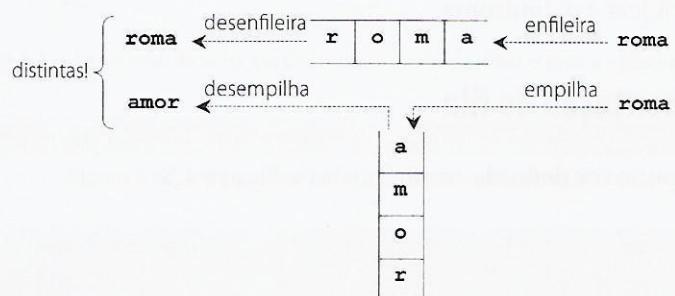


Figura 4.3 | Comparaçāo entre uma cadeia e sua inversa.

O programa que verifica se uma cadeia digitada pelo usuário é palíndroma é apresentado na Figura 4.4.

```
// palindroma.c - verifica se uma cadeia é palindroma

#include <stdio.h>
#include <ctype.h>
#include "../ed/pilha.h" // pilha de char
#include "../ed/fila.h" // fila de char

int main(void) {
    char s[256];
    Fila F = fila(256);
    Pilha P = pilha(256);
    printf("\nFrase? ");
    gets(s);
    for(int i=0; s[i]; i++)
        if( isalpha(s[i]) ) {
            enfileira(s[i], F);
            empilha(s[i], P);
        }
}
```

```
while( !vaziaf(F) && desenfileira(F)==desempilha(P) );
if( vaziaf(F) ) puts("A frase e palindroma");
else puts("A frase nao e palindroma");
destroif(&F);
destroip(&P);
return 0;
}
```

Figura 4.4 | Programa para verificação de cadeias palíndromas.

Nesse programa, `#include "...`*ed/fila.h*" inclui a implementação do tipo `Fila`, que será desenvolvida na próxima seção. A lógica do programa consiste essencialmente de duas repetições: a primeira delas percorre a cadeia, inserindo suas letras numa fila e numa pilha (a função `isalpha()`, declarada em `ctype.h`, é usada para reconhecer as letras); a segunda confere se as letras removidas da fila são iguais às aquelas removidas da pilha (a repetição termina quando a fila fica vazia ou quando uma incorrespondência é encontrada). No fim, se a fila estiver vazia, então nenhuma incorrespondência foi encontrada e a cadeia é palíndroma.

4.3 Implementação de fila

Em C, uma fila pode ser definida como mostra a Figura 4.5.

Figura 4.5 | Definição da estrutura de fila.

Nessa figura, a primeira linha define o tipo `Itemf` como `char`, indicando que os itens da fila são caracteres. As demais linhas definem `Fila` como um tipo de ponteiro que aponta uma estrutura (`struct fila`) com cinco campos: `max`, que indica a *capacidade* máxima da fila; `total`, que indica o *total* de itens guardados na fila; `inicio`, que indica a posição *inicial* da fila; `final`, que indica a posição *final* da fila e `item`, que aponta um vetor dinâmico que guarda os *itens* da fila. Por exemplo, a Figura 4.6 mostra uma fila criada a partir dessas definições.

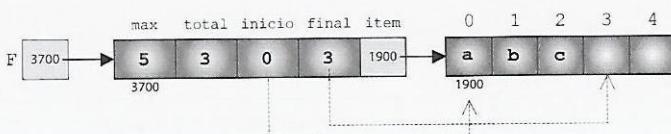


Figura 4.6 | Ponteiro para uma estrutura de fila, que armazena os itens a, b e c.

Os campos de uma fila apontada por um ponteiro F são $F->max$, $F->total$, $F->inicio$, $F->final$ e $F->item$. Esses campos nunca devem ser acessados diretamente por um programa que *usa* a fila. Toda manipulação de fila deve ser feita *exclusivamente* pelas funções que implementam as operações em fila.

Para melhor aproveitamento de espaço no vetor $F->item$, vamos simular que esse vetor é *circular*, como na Figura 4.7. Então, quando um índice ($F->inicio$ ou $F->final$) indicando a última posição desse vetor for *avançado*, ele voltará a indicar a primeira posição (assim, posições desocupadas por itens removidos da fila poderão ser reusadas para a inserção de novos itens). A operação que *avança* índices de forma circular é definida na Figura 4.8.

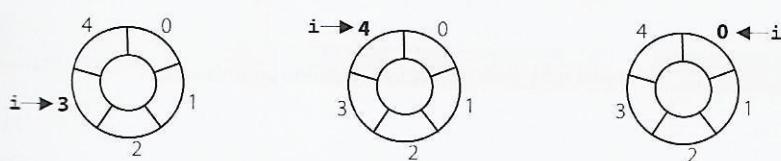


Figura 4.7 | Simulação de vetor circular: após a última posição, o índice volta à primeira.

```
#define avanca(i) (i = (i+1) % F->max)
```

Figura 4.8 | Operação de avanço de índice num vetor circular.

4.3.1 Criação de fila

A função para criação de fila é definida na Figura 4.9.

```
Fila fila(int m) {
    Fila F = malloc(sizeof(struct fila));
    F->max    = m;
    F->total   = 0;
    F->inicio  = 0;
    F->final   = 0;
    F->item    = malloc(m*sizeof(Itemf));
    return F;
```

Figura 4.9 | Função para criação de fila.

Quando chamada, a função `fila()` executa os seguintes passos:

- Chama a função `malloc()` para alocar a área de memória onde a estrutura de fila será criada, cujo tamanho em bytes é `sizeof(struct fila)`. Caso haja memória suficiente, a função `malloc()` aloca o espaço solicitado e devolve o seu endereço como resposta; caso contrário, ela devolve `NULL`. O endereço devolvido pela função `malloc()` é atribuído ao ponteiro `F`.
- Acessa o campo `max` apontado por `F` e atribui a ele o valor `m`.

- Acessa os campos `total`, `inicio` e `final`, apontados por `F`, e atribui a todos eles o valor 0.
 - Acessa o campo `item` apontado por `F` e atribui a ele o endereço de um vetor dinâmico, com capacidade para armazenar `n` valores do tipo `Itemf`.
 - Devolve como resposta o endereço da estrutura de fila que foi criada.
- Por exemplo, a fila na Figura 4.10 pode ser criada da seguinte forma:

```
Fila F = fila(5);
```

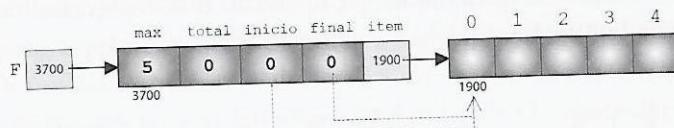


Figura 4.10 | Um ponteiro apontando uma fila vazia.

4.3.2 Teste em fila

Numa fila apontada por um ponteiro `F`, o campo `F->total` indica o total de itens na fila. Quando esse campo tem valor 0, a fila está vazia (Figura 4.10); por outro lado, quando ele tem valor `F->max`, a fila está cheia (Figura 4.11). Observe que, quando a posição 4 do vetor é ocupada, o índice `F->final` avança *circularmente*, retornando à posição 0. De fato, sempre que a fila está vazia ou cheia, os índices `F->inicio` e `F->final` têm o mesmo valor (não necessariamente 0). Porém, com `F->total`, é possível distinguir claramente uma fila vazia de uma fila cheia.

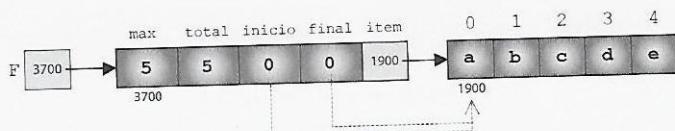


Figura 4.11 | Um ponteiro apontando uma fila cheia.

As funções para teste de fila vazia e teste de fila cheia são definidas, respectivamente, nas Figuras 4.12 e 4.13.

```
int vaziaf(Fila F) {
    return ( F->total == 0 );
}
```

Figura 4.12 | Função para teste de fila vazia.

```
int cheiaf(Fila F) {
    return ( F->total == F->max );
}
```

Figura 4.13 | Função para teste de fila cheia.



deles o
dinâ-

4.3.3 Inserção em fila

Para inserir um item numa fila, primeiro temos que verificar se há espaço. Caso a fila esteja cheia, a função de inserção causa um erro de *fila cheia* e a execução do programa é abortada. Caso contrário, o item deve ser inserido no final da fila. Para isso, basta guardar o item em $F \rightarrow item[F \rightarrow final]$, avançar *circularmente* o índice $F \rightarrow final$ e incrementar o campo $F \rightarrow total$ (Figura 4.14).

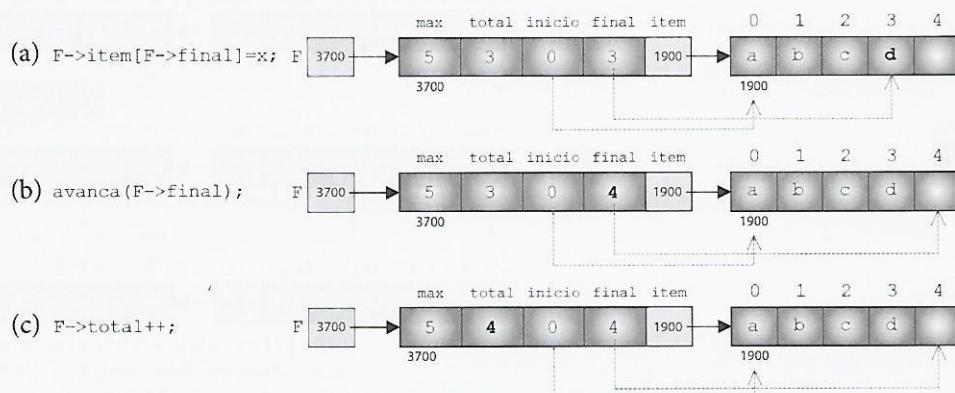


Figura 4.14 | Passos para inserir um item numa fila.

A função para inserção em fila é definida na Figura 4.15. Nessa função, a função `abort()`, declarada em `stdlib.h`, é usada para abortar a execução do programa.

```
void enfileira(Itemf x, Fila F) {
    if( cheiaf(F) ) { puts("fila cheia!"); abort(); }
    F->item[F->final] = x;
    avanca(F->final);
    F->total++;
}
```

Figura 4.15 | Função para inserção em fila.

4.3.4 Remoção em fila

A função para remoção em fila, definida na Figura 4.16, também é bem simples.

```
Itemf desenfileira(Fila F) {
    if( vaziaf(F) ) { puts("fila vazia!"); abort(); }
    Itemf x = F->item[F->inicio];
    avanca(F->inicio);
    F->total--;
    return x;
}
```

Figura 4.16 | Função para remoção em fila.



Para remover um item de uma fila, primeiro essa função verifica se a fila está vazia. Caso esteja, ocorre um erro de *fila vazia* e a execução do programa é abortada. Caso contrário, o item no início da fila deve ser removido. Para isso, a função copia o item *F->item[F->inicio]* numa variável auxiliar; avança *circularmente* o índice *F->inicio* e decrementa o campo *F->total*, como ilustrado na Figura 4.17. No fim, o valor da variável auxiliar é devolvido como resposta.

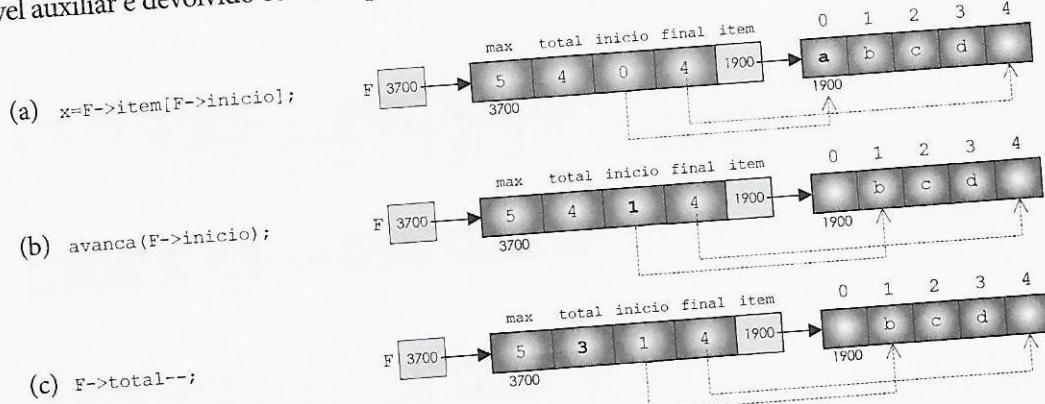


Figura 4.17 | Passos para remover um item de uma fila.

4.3.5 Destrução de fila

A função para destruição de fila é definida na Figura 4.18. Para destruir uma fila apontada por *F*, basta chamar `destroif(&F)`. Essa é a única função de fila cujo parâmetro *F* é passado por *referência* (isto é, *G* é um *ponteiro de ponteiro*).

```
void destroif(Fila *G) {
    free((*G)->item);
    free(*G);
    *G = NULL;
}
```

Figura 4.18 | Função para destruição de fila.

Quando a chamada `destroif(&F)` é feita, o endereço do ponteiro *F* é copiado para o ponteiro *G*, usado como parâmetro da função. Então, a notação **G* permite acessar o ponteiro *F* e, consequentemente, a notação *(*G)->item* permite acessar o campo *item* da estrutura apontada por *F*. Assim, quando a chamada `free((*G)->item)` é feita, o vetor *item* apontado por *F* é destruído. Depois, quando a chamada `free(*G)` é feita, a estrutura de fila apontada por *F* também é destruída. Finalmente, quando a atribuição **G=NULL* é feita, o ponteiro *F* passa a ter valor *NULL* (isto é, a fila que era apontada por ele não existe mais).

4.3.6 O ar

Daqui em
arquivo fil
C). Então,
ed/fila.h
fila.h ser
zam que fi

Exercíc

4.1 Simu

Fila
enfil
enfil
enfil
enfil
enfil
pri

4.2 Qu

#in
#in
int

4.3 O

U
co

4.4 O

a
c
t
f
c

4.3.6 O arquivo fila.h

Daqui em diante, assumimos que as definições de tipos e funções para filas estão no arquivo `fila.h`, na pasta `Pelles C Projects/ed` (veja mais detalhes no Apêndice C). Então, para usar o tipo `Fila` num programa, basta usar a diretiva `#include "../ed/fila.h"`. Assim, durante a compilação do programa, todas as definições no arquivo `fila.h` serão usadas automaticamente. As aspas em `#include "../ed/fila.h"` enfatizam que `fila.h` não é um arquivo padrão em C.

Exercícios

- 4.1** Simule a execução do código a seguir e indique a saída exibida em vídeo:

```
Fila F = fila(5);
enfileira(1,F);
enfileira(2,F);
enfileira(3,F);
enfileira(desenfileira(F),F);
enfileira(desenfileira(F),F);
printf("%d\n",desenfileira(F));
```

- 4.2** Qual a saída exibida pelo programa a seguir?

```
#include <stdio.h>
#include "../ed/fila.h" // fila de char
int main(void) {
    Fila F = fila(5);
    for(int i=0; i<=3; i++) enfileira('A'+i,F);
    while( !vaziaf(F) ) printf("%c\n",desenfileira(F));
    destroif(&F);
    return 0;
}
```

- 4.3** O programa da Figura 4.4 não reconhece "Amor a Roma" como uma cadeia palíndroma. Use a função `toupper()`, declarada em `ctype.h`, para resolver esse problema (essa função converte uma letra minúscula em maiúscula).

- 4.4** O programa a seguir simula o compartilhamento de uma CPU entre vários processos que aguardam numa fila para serem executados. Enquanto a fila não fica vazia, o primeiro processo na fila pode usar a CPU por certo período de tempo. Se nesse período o processo termina, ele é removido da fila; senão, ele volta para o final dela e o próximo processo na fila passa a usar a CPU. Nessa fila, um processo `p` que precisa de `t` unidades de tempo para concluir sua execução é representado por um número da forma `p*10+t` (sendo `p` e `t` dígitos). Analise o programa e indique a ordem de conclusão dos processos.



```
#include <stdio.h>
#include "../ed/fila.h" // fila de int
#define tempo 3           // tempo maximo de uso de CPU
int main(void) {
    Fila F = fila(5);
    enfileira(17,F);
    enfileira(25,F);
    enfileira(39,F);
    enfileira(46,F);
    while( !vaziaf(F) ) {
        int x = desenfileira(F);
        int p = x/10;
        int t = x%10;
        if( t>3 ) enfileira(p*10+(t-tempo),F);
        else printf("Processo %d concluido\n",p);
    }
    destroif(&F);
    return 0;
}
```