4. Fila

"A sutileza do pensamento consiste em
descobrir a semelhança das coisas diferentes
e a diferença das coisas semelhantes."
Charles de Montesquieu

Uma fila é um conjunto ordenado de itens a partir do qual se podem eliminar itens numa extremidade - início da fila - e no qual se podem inserir itens na outra extremidade - final da fila.

Ela é uma *prima* próxima da pilha, pois os itens são inseridos e removidos de acordo com o princípio de *que o primeiro que entra é o primeiro que sai - first in, first out* (FIFO).

O conceito de fila existe no mundo real, vide exemplos como filas de banco, pedágios, restaurantes etc. As operações básicas de uma fila são:

- *insert* ou *enqueue* - insere itens numa fila (ao final).
- *remove* ou *dequeue* - retira itens de uma fila (primeiro item).
- *empty* - verifica se a fila está vazia.
- *size* - retorna o tamanho da fila.
- *front* - retorna o próximo item da fila sem retirar o mesmo da fila.

A operação **insert** ou **enqueue** sempre pode ser executada, uma vez que teoricamente uma fila não tem limite. A operação **remove** ou **dequeue** só pode ser aplicado se a fila não estiver vazia, causando um erro de *underflow* ou fila vazia se esta operação for realizada nesta situação.

Tomando a fila da figura 4.1 como exemplo, o item a apresenta a fila no seu estado inicial e é executado o conjunto de operações:

- *dequeue()* - Retorna o item A (a fila resultante é representada pelo item B)
- *enqueue(F)* - O item F é armazenado ao final da fila (a fila resultante é representada pelo item C)
- *dequeue()* - Retirado o item B da fila
- *enqueue(G)* - Colocado o item G ao final da fila (item D)

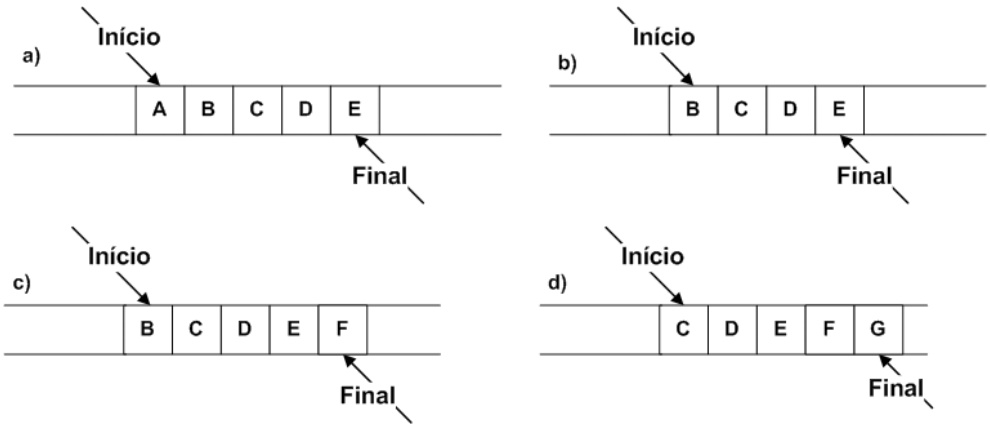

Figura 4.1: Operações numa fila

4.1 Representação de Filas com Pseudo-códigos

As operações em fila podem ser representados com os seguintes blocos de código: a operação **enqueue** (4.1), **dequeue** (4.2), **empty** (4.3), **size** (4.4) e **front** (4.5). Estes códigos podem ser adaptados para qualquer linguagem de programação [9].

Algoritmo 4.1: Inclusão de dados na fila (ENQUEUE(Q,x))

```
Input: A variável que contém a fila (Q) e o elemento a ser colocado na fila (x)
Output: Sem retorno
1  begin
2      Q[ fim de Q]⟵ x
3      if final de Q = comprimento de Q then
4          fim de Q⟵ 1
5          return
6      else
7          Q[ fim de Q]⟵ fim de Q + 1
8          return
9      endif
10 end
```

Algoritmo 4.2: Retirada de dados na fila (DEQUEUE(Q))

```
Input: A variável que contém a fila (Q)
Output: O elemento representado por na fila Q
1  begin
2      x ⟵ Q[ inicio de Q]
3      if início de Q = comprimento de Q then
4          Inicio de Q⟵ 1
5      else
6          Inicio de Q⟵ Inicio de Q + 1
7      endif
8      return x
9  end
```

Algoritmo 4.3: Verificação se a fila está vazia (função EMPTY(Q))

```
Input: A variável que contém a fila (Q)
Output: Verdadeiro ou falso
1  begin
2      if Inicio de Q = Fim de Q then
3          return true
4      else
5          return false
6      endif
7  end
```

Algoritmo 4.4: Tamanho da fila (função SIZE(Q))

```
   Input: A variável que contém a fila (Q)
   Output: Quantidade de itens da fila
1  begin
2  |  return fim de Q
3  end
```

Algoritmo 4.5: Próximo elemento da fila (função FRONT(Q))

```
   Input: A variável que contém a fila (Q)
   Output: Próximo elemento da fila (mas sem retirar da fila)
1  begin
2  |  x ⟵ Q[ inicio de Q ]
3  |  return x
4  end
```

4.2 Filas em C

A exemplo do que ocorre com estrutura em pilha, antes de programar a solução de um problema que usa uma fila, é necessário determinar como representar uma fila usando as estruturas de dados existentes na linguagem de programação. Novamente na linguagem C podemos usar um vetor. Mas a fila é uma estrutura dinâmica e pode crescer infinitamente, enquanto que um vetor na linguagem C tem um tamanho fixo. Contudo, pode-se definir este vetor com um tamanho suficientemente grande para conter a fila. No programa 4.1 é apresentado um exemplo para manipulação de filas.

Programa 4.1: Exemplo de manipulação de fila em C

```
   /* programa_fila_01.c */
   #include <stdio.h>
3  #include <stdlib.h>

   #define TAMANHO_MAXIMO 100

   struct queue
8  {
       int itens[TAMANHO_MAXIMO];
       int front,rear;
   };

13 int empty(struct queue * pq)
```

```
{
    /* se o início da fila for igual ao final da fila, a fila está vazia */
    if( pq->front == pq->rear )
    {
        return 1;
    }
    return 0;
}
void enqueue(struct queue * pq, int x)
{
    if( pq->rear + 1 >= TAMANHO_MAXIMO )
    {
        printf("\nEstouro da capacidade da fila");
        exit(1);
    }
    pq->itens[ pq->rear++ ] = x;
    return;
}

int size(struct queue * pq)
{
    return (pq->rear + 1);
}

int front(struct queue * pq)
{
    /* o primeiro elemento sempre está no início do vetor */
    return pq->itens[0];
}

int dequeue(struct queue * pq)
{
    int x, i;
    if( empty(pq) )
    {
        printf("\nFila vazia");
        exit(1);
    }
    /* Salva o primeiro elemento e refaz o arranjo dos itens, puxando o segundo elemento
       para o primeiro, o terceiro para o segundo e assim sucessivamente. */
    x = pq->itens[0];
    for( i=0; i < pq->rear; i++)
    {
        pq->itens[i] = pq->itens[i+1];
    }
    pq->rear--;
    return x;
}
```

```
int main(void)
{
    struct queue q;
    q.front = 0; q.rear = 0;
    enqueue(&q,1);
    enqueue(&q,2);
    enqueue(&q,3);
    enqueue(&q,4);

    printf("\nFila vazia %d", empty(&q));
    printf("\nTamanho da fila %d", size(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));

    printf("\nFila vazia %d", empty(&q));

    printf("\n");
}
```

No programa 4.1, o vetor foi definido para comportar apenas 100 elementos, caso fosse inserido um 101º elemento, haveria o estouro da pilha mesmo após várias operações de **dequeue**. Para resolver este problema, na operação **dequeue** foi implementada uma técnica de redistribuição dos elementos na fila, de tal forma que nunca se chegue a estourar a fila caso haja várias operações de inserção ou remoção (exceto se realmente houver 100 elementos da fila e houve uma tentativa de inserção de um novo elemento). O programa 4.2 é o trecho que implementa a técnica comentada:

Programa 4.2: Reajuste da fila

```
x = pq->itens[0];
for( i=0; i < pq->rear; i++)
{
    pq->itens[i] = pq->itens[i+1];
}
pq->rear--;
```

Esta técnica é ineficiente, pois cada eliminação da fila envolve deslocar cada elemento restante na fila. Se uma fila contiver 1000 ou 2000 elementos, cada elemento retirado da fila provocará o deslocamento de todos os demais elementos. A operação de remoção de um item na fila deveria logicamente trabalhar somen-

te com aquele elemento, permanecendo os demais elementos em suas posições originais.

A solução para o problema é definir o vetor como um círculo, em vez de uma linha reta. Neste caso, os elementos são inseridos como numa fila reta, e a remoção de um elemento da fila não altera os demais elementos da fila. Com o conceito de fila circular, ao chegar ao final da fila, o ponteiro de controle da fila vai imediatamente para o início da fila novamente (se este estiver vago). As seguintes operações exemplificam a explicação (acompanhar o desenvolvimento da fila na figura 4.2), sendo o caso 1 o estado inicial da fila:

1. Estado inicial
2. *enqueue(D)* - O item D é armazenado ao final da fila
3. *enqueue(E)* - O item D é armazenado ao final da fila
4. *dequeue()* - Retirado o item A da fila
5. • *enqueue(F)* - O item F é armazenado ao final da fila
   • *enqueue(G)* - O item G é armazenado ao final da fila
6. *dequeue()* - Retirado o item B da fila
7. *enqueue(H)* - O item H é armazenado ao final da fila. Neste momento, o ponteiro da fila chegou ao final do vetor que contém a implementação da fila.
8. • *dequeue()* - Retirado o item C da fila
   • *enqueue(I)* - O item I é armazenado ao final da fila (mas no início do vetor)
9. *enqueue(K)* - O item K é armazenado ao final da fila (mas na segunda posição do vetor)

O programa 4.3 mostra a declaração da estrutura para uma fila circular.

Programa 4.3: Declaração de estrutura circular

```
#define TAMANHO_MAXIMO 100

struct queue
{
    int itens[TAMANHO_MAXIMO];
    int front,rear;
};
struct queue q;
q.front = q.rear = -1
```

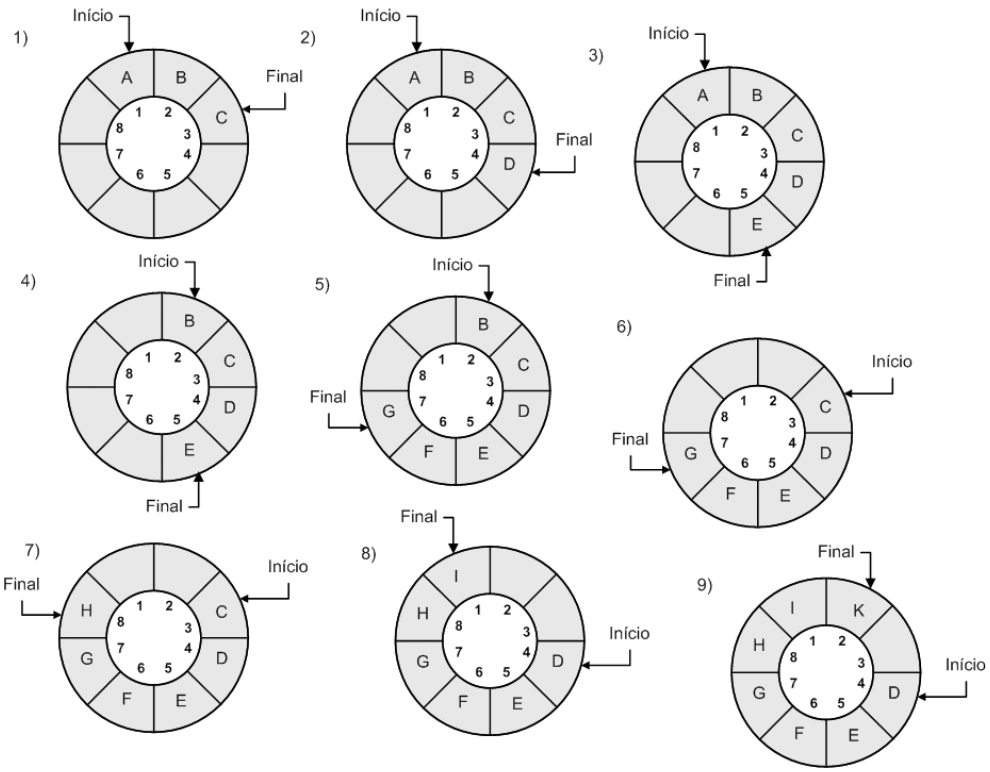

Figura 4.2: Operações numa fila circular

Desta forma, as funções de manipulação de fila (**empty**, **enqueue**, **dequeue**, **size** e **front**) devem sofrer modificações para refletir a nova condição de fila circular (programa 4.4):

Programa 4.4: Manipulação de fila circular em C

```
/* programa_fila_02.c */
#include <stdio.h>
#include <stdlib.h>

#define TAMANHO_MAXIMO 10

struct queue
{
int itens[TAMANHO_MAXIMO];
int front,rear;
};
```

```
int empty(struct queue * pq)
{
    if( pq->front == pq->rear )
    {
        return 1;
    }
    return 0;
}

void enqueue(struct queue * pq, int x)
{
    /* Inversão das posições dos ponteiros. Se o final do vetor já foi
    alcançado, então retorna-se ao início do vetor */
if( pq->rear == TAMANHO_MAXIMO-1)
{
    pq->rear = 0;
}
else
{
    pq->rear++;
}
if( pq->rear == pq->front )
{
    printf("\nEstouro da fila");
    exit(1);
}
    pq->itens[pq->rear] = x;
    return;
}

int size(struct queue * pq)
{
    /* se o final da fila ainda não alcançou o final do vetor... */
    if( pq->front <= pq->rear)
    {
        return pq->rear - pq->front; /* ... então o tamanho da fila é o final
                                        da fila menos o início da fila... */
}


    /* ... se não, quer dizer que o ponteiro de final da fila já alcançou o final do vetor
       e foi reposicionado para o início do vetor, então o tamanho da fila é a quantidade
       de itens que ainda restam até chegar ao final do vetor somando itens que estão
       no início do vetor */
    return pq->rear + pq->front;
}

int front(struct queue * pq)
{
```

```
    return pq->itens[pq->front+1];
}

int dequeue(struct queue * pq)
{
    int x, i;
    if( empty(pq) )
    {
        printf("\nFila vazia");
        exit(1);
    }

    /* Inversão das posições dos ponteiros. Se o final do vetor já foi alcançado,
      então retorna-se ao início do vetor */
    if( pq->front == TAMANHO_MAXIMO - 1)
    {
        pq->front = 0;
    }
    else
    {
        pq->front++;
    }
    return (pq->itens[ pq->front ]);
}


int main(void)
{
    struct queue q;
    q.front = -1;
    q.rear = - 1;
    enqueue(&q,1);
    enqueue(&q,2);
    enqueue(&q,3);
    enqueue(&q,4);


    printf("\nTamanho da fila %d", size(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));
    enqueue(&q,5);
    enqueue(&q,6);
    enqueue(&q,7);
    enqueue(&q,8);
```

```
    enqueue(&q,9);
    printf("\nTamanho da fila %d", size(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));

    enqueue(&q,10);
    enqueue(&q,11);
    enqueue(&q,12);
    enqueue(&q,13);
    enqueue(&q,14);
    enqueue(&q,15);
    enqueue(&q,16);
    enqueue(&q,17);
    enqueue(&q,18);
    printf("\nTamanho da fila %d", size(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));

    printf("\nFila vazia %d", empty(&q));

    enqueue(&q,20);
    enqueue(&q,21);
    enqueue(&q,22);
    enqueue(&q,23);
    enqueue(&q,24);
    enqueue(&q,25);
    printf("\nTamanho da fila %d", size(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nProximo da fila %d", front(&q));
    printf("\nTirando da fila %d", dequeue(&q));
```

```
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));
    printf("\nTirando da fila %d", dequeue(&q));
    printf("\nTamanho da fila %d", size(&q));

    printf("\nFila vazia %d", empty(&q));


    printf("\n");
    return 0;
}
```

4.3 Exercícios

1. Se um vetor armazenando uma fila não é considerado circular, o texto sugere que cada operação **dequeue** deve deslocar para baixo todo elemento restante de uma fila. Um método alternativo é adiar o deslocamento até que ***rear*** seja igual ao último índice do vetor. Quando essa situação ocorre e faz-se uma tentativa de inserir um elemento na fila, a fila inteira é deslocada para baixo, de modo que o primeiro elemento da fila fique na posição 0 do vetor. Quais são as vantagens desse método sobre um deslocamento em cada operação **dequeue**? Quais as desvantagens? Reescreva as rotinas **dequeue**, **queue** e **size** usando esse método.

2. Faça um programa para controlar uma fila de pilhas.