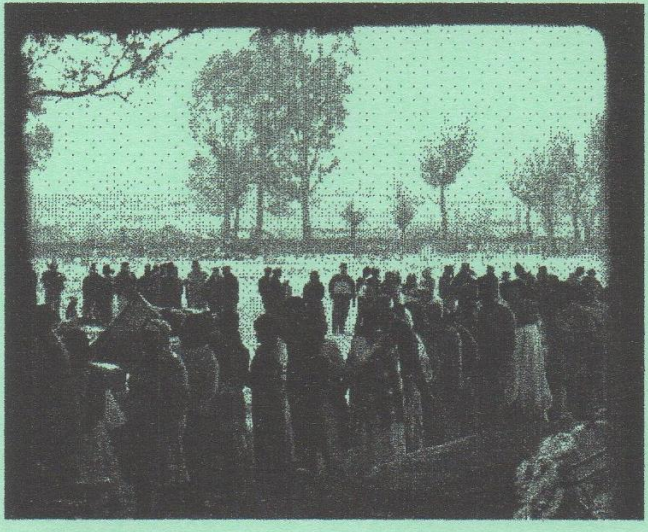
UMA MINI-ZINE SOBRE

FILA

(QUEUE)



@andrekishimoto

-2023-

FILA (QUEUE)

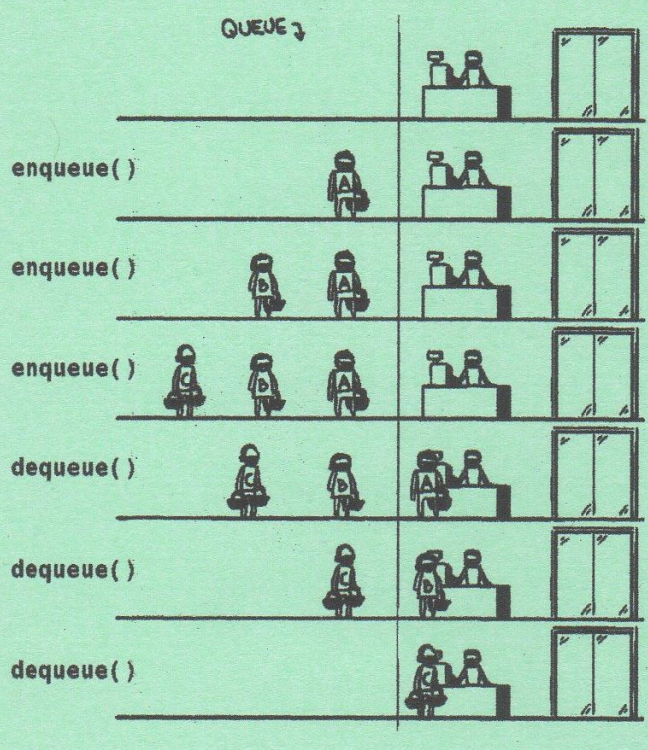
A fila (***queue***) é uma estrutura de dados que define como os dados são acessados, aplicando uma regra que segue o princípio **FIFO**: *First-In, First-Out* (o primeiro a entrar é o primeiro a sair). Embora muito incomum, poderíamos usar o termo **LILO**: *Last-In, Last-Out* (o último a entrar é o último a sair).

Essa estrutura segue o mesmo conceito de fila que temos na vida real: filas no supermercado, no banco, em show, no banheiro, etc. A famosa "ordem de chegada" (sem contar as filas prioritárias). E, obviamente, estamos desconsiderando os casos das pessoas que furam fila e atrapalham a vida de todos...

As operações fundamentais de uma fila são:
» **`enqueue()`**: um elemento inserido na fila sempre será colocado no final da fila;
» **`dequeue()`**: um elemento removido da fila sempre é o primeiro elemento da fila (o que está há mais tempo na fila);
» **`front()`**: a pessoa que usa a estrutura fila só consegue saber quem é o primeiro elemento da fila (operação também conhecida como **`first()`** e/ou **`peek()`**).

Outras operações que facilitam o uso de uma fila:
» **`create()`**: cria e retorna uma fila vazia.
» **`size()`**: retorna a capacidade da fila.
» **`count()`**: retorna a quantidade de elementos na fila.

» **`isEmpty()`**: retorna se a fila está vazia ou não.
» **`isFull()`**: retorna se a fila está cheia ou não.
» **`clear()`**: esvazia a fila (remove todos os elementos da fila).



IMPLEMENTAÇÃO DA FILA

Podemos criar filas estáticas (sequenciais) e dinâmicas (encadeadas). Essa zine apresenta apenas a versão sequencial, cuja implementação usa um *array* `<type> data[]` de tamanho fixo para armazenar os dados inseridos em uma fila (`<type>` é o tipo da fila; ex. `int`).

O **tamanho do *array* define a capacidade da fila** e pode ser definido em tempo de compilação (alocação estática) ou em tempo de execução (alocação dinâmica). Para uma capacidade definida em tempo de compilação, é recomendável criar uma constante inteira ao invés de usar "números mágicos" no código – por exemplo, `const int QUEUE_CAPACITY=128;` ou o equivalente na linguagem de programação de sua escolha.

Um detalhe importante sobre o *array* interno da fila! Poderíamos usar o *array* de "maneira tradicional", isto é, o primeiro da fila sempre seria representado pelo elemento de índice zero. Funciona, mas a solução não é muito boa, pois teríamos que deslocar todos os elementos do *array* quando o primeiro elemento é removido da fila, resultando em uma operação de remoção $O(n)$, quando é possível fazer a remoção em tempo $O(1)$.

Como? Usando o *array* de maneira circular! Nesse caso, quando chegamos ao final do *array* (último índice), devemos voltar para o início do *array* – veja a última página da zine para uma ilustração do ***array* circular**.

Para auxiliar no uso do *array* circular, usamos três variáveis de controle:
» A quantidade de elementos na fila (**`int count`**).
» O índice do início/primeiro da fila (**`int first`**).
» O índice correto para inserir um elemento no fim da fila, o novo último elemento (**`int last`**).
(Na verdade, precisamos só do **`first`** e **`count`**, já que podemos calcular **`last = first + count`**.)

A atualização dessas variáveis de controle é feita de acordo com o tipo de operação que modifica a fila:
» Inserção na fila: **`last`** e **`count`** são incrementadas.
» Remoção da fila: **`count`** é decrementada e **`first`** é incrementada (veja a ilustração da última página).

Se, após o incremento, **`first`** e/ou **`last`** chegarem ao índice **`N`** (capacidade da fila), então devem voltar para o índice zero. Podemos fazer o incremento e correção dos valores dessas variáveis da seguinte forma:

```
» first = (first + 1) % N;
» last = (last + 1) % N;
```

operação módulo: resto da divisão (nº inteiros).

Generalizando, quando fazemos **`X = VALOR % N`**, mantemos **`X`** no intervalo **`0 <= X < N`**.

Como exemplo, se **`N=5`** e **`VALOR={0..12}`**, observe que **`X`** sempre será **`0`**, **`1`**, **`2`**, **`3`** ou **`4`**, já que: **`0%5=0`**; **`1%5=1`**; **`2%5=2`**; **`3%5=3`**; **`4%5=4`**; **`5%5=0`**; **`6%5=1`**; **`7%5=2`**; **`8%5=3`**; **`9%5=4`**; **`10%5=0`**; **`11%5=1`** e **`12%5=2`**.

A seguir, temos os pseudocódigos das operações fundamentais da fila.

ENQUEUE

```
void enqueue(<type> value) { // O(1)
  // TODO: Fila cheia?

  data[last] = value;
  last = (last + 1) % QUEUE_CAPACITY;
  ++count;
}
```

DEQUEUE

```
<type> dequeue() { // O(1)
  // TODO: Fila vazia?

  <type> front = data[first];
  // Reinicia o elemento da fila. Mude de
  // acordo com o tipo de dado e contexto.
  data[first] = null;
  first = (first + 1) % QUEUE_CAPACITY;
  --count;
  return front;
}
```

FRONT

```
<type> front() { // O(1)
  // TODO: Fila vazia?

  return data[first];
}
```

AGORA É A SUA VEZ!

1. Implemente as operações fundamentais da estrutura de dados fila na linguagem de programação que você está estudando e escreva um exemplo de código que insere e remove elementos da fila.

2. Implemente as operações extras `create()`, `size()`, `count()`, `isEmpty()`, `isFull()` e `clear()`. A operação `create()` pode ser o construtor da classe da fila, caso esteja usando uma linguagem orientada a objetos e baseada em classes.

3. Nos pseudocódigos, há comentários `TODO` ("a fazer"), indicando que devemos verificar os casos em que a fila está cheia ou vazia. Remova esses comentários e implemente as verificações para que a fila funcione corretamente.

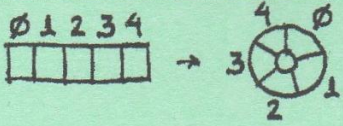
4. Qual é o ponto negativo da fila sequencial?

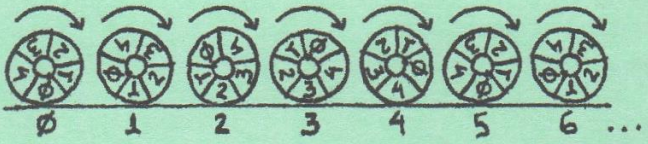
5. Pesquise e descreva algumas aplicações computacionais onde a estrutura fila é usada.

6. Explique o que é a estrutura de dados fila, mas com as suas próprias palavras. É um bom exercício para verificar se você entendeu o conceito de fila. Aproveite e consulte mais referências para se aprofundar no assunto!

Sugestão: Aproveite o verso da folha para escrever as suas respostas!

ARRAY CIRCULAR





	count	first	last	data[] 0 1 2 3 4
create( )	0	0	0	[ ][ ][ ][ ][ ]
enqueue(A)	1	0	1	[A][ ][ ][ ][ ]
enqueue(B)	2	0	2	[A][B][ ][ ][ ]
enqueue(C)	3	0	3	[A][B][C][ ][ ]
dequeue( )	2	1	3	[ ][B][C][ ][ ]
enqueue(D)	3	1	4	[ ][B][C][D][ ]
enqueue(E)	4	1	5→0	[ ][B][C][D][E]
dequeue( )	3	2	5→0	[ ][ ][C][D][E]
enqueue(F)	4	2	6→1	[F][ ][C][D][E]