O QUE SÃO FILAS?

É uma lista encadeada especializada que atua em uma estratégia muito bem definida chamada FIFO.

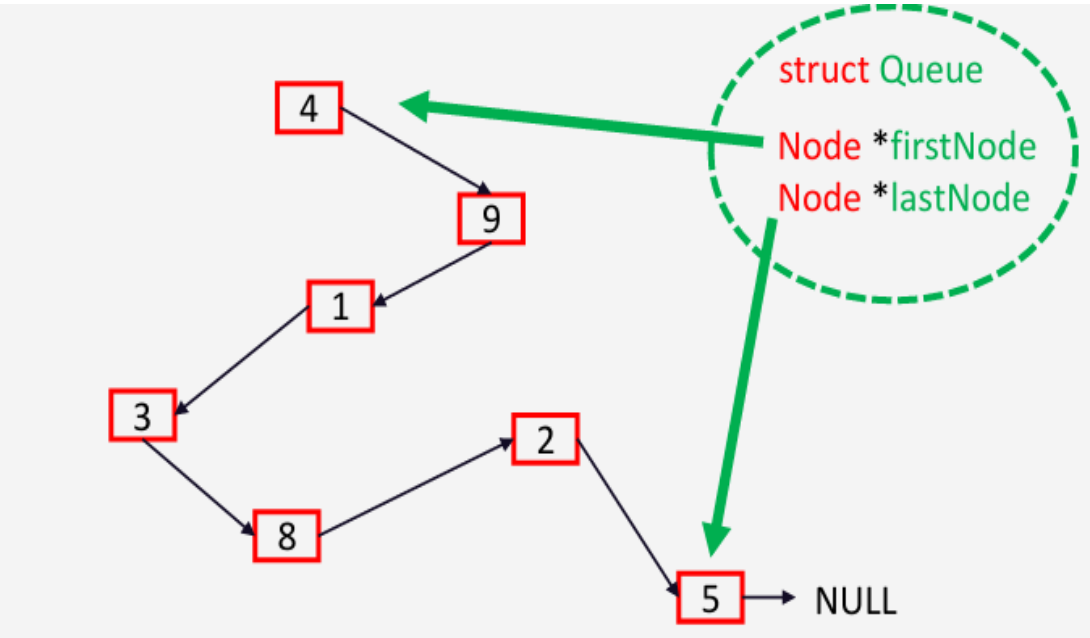
- FIFO–First In, First Out ou seja: “o primeiro que entra é o primeiro que sai”
- Obedece o critério de chegada
- A remoção ocorre no início da fila
- A inserção ocorre no final da fila

Temos que guardar um ponteiro para o primeiro e último nó da lista encadeada.

```c
// Definição da estrutura da Fila (Queue)
struct Queue
{
    struct Node* first; // Ponteiro para o primeiro elemento (cabeça) da fila (onde a remoção ocorre - DEQUEUE).
    struct Node* last; // Ponteiro para o último elemento (cauda) da fila (onde a inserção ocorre - ENQUEUE).
};
```

INSERT E REMOVE

Pelo fato de a fila ter o comportamento FIFO, os métodos de inserir e remover já trazem em si que:



INSERÇÃO - OCORRE NO FINAL

Insere um novo nó no FIM da fila, seguindo o princípio FIFO.

Recebe o ponteiro da fila e o valor a ser inserido e retorna a fila atualizada.

Se a fila está vazia o ponteiro ‘last’ aponta para NULL, neste caso o primeiro elemento será o início e o final, logo ‘first’ e ‘last’ apontam para o mesmo nó.

```
else
{
    // Se a fila não está vazia:

    //ANTES DA INSERÇÃO: [10] -> NULL

    // O ponteiro 'next' e 'last' é atualizado para apontar para o novo nó.
    fila->last->next = new_node;
    // O ponteiro 'last' da fila é atualizado para apontar para o novo nó (o novo final da fila).
    fila->last = new_node;

    //APÓS A INSERÇÃO: [10] -> [20] -> NULL
}
return fila; // Retorna o ponteiro para a fila atualizada.
```

REMOÇÃO - OCORRE NO INÍCIO

Se a fila estiver vazia, retorna a fila sem fazer nada.

```
// Cria um ponteiro temporário para o nó que será removido (o primeiro).
struct Node* temp = q->first;

//[10] -> [20] -> NULL
// O ponteiro 'first' da fila é movido para o próximo nó na lista.
q->first = q->first->next;
//-> [20] -> NULL

// Verifica se a fila ficou vazia após a remoção.
if (q->first == NULL)
{
    // Se ficou vazia, o ponteiro 'last' também deve ser ajustado para NULL.
    q->last = NULL;
}

// Libera a memória alocada para o nó removido (evita vazamento de memória).
free(temp);

return q; // Retorna o ponteiro para a fila atualizada.
```

VERIFICAR SE A LISTA ESTÁ VAZIA

→ SE O PONTEIRO DO INÍCIO(FIRST) APONTAR PARA NULL, SIGNIFICA QUE NÃO HÁ.

LIBERAR MEMORIA ALOCADA

→ PERCORRER TODOS OS NÓS E LIBERAR CADA UM (free(no), ALÉM DE LIBERAR A PRÓPRIA ESTRUTURA DA FILA free(nome_fila).
→ CRIAR UM PONTEIRO AUXILIAR QUE INICIE PELO PRIMEIRO NÓ

```
struct Node* aux = nome_dado->first; // Começa pelo primeiro nó
```

PERCORRER A LISTA ENCADEADA

→ GUARDAR O PONTEIRO UM QUE SERÁ LIBERADO(EM UM TEMP)
→ MOVER O PONTEIRO PARA O PRÓXIMO(NEXT) NÓ.
→ LIBERA A MEMÓRIA DO NÓ SALVO EM TEMP
→ LIBERA A MEMORIA ALOCADA PARA A ESTRUTURA.

IMPRIMIR OS ELEMENTOS DA FILA

→ COMEÇA A PERCORRER PELO PRIMEIRO NÓ
→ PERCORRA ENQUANTO HOUVER UM NÓ (O PONTEIRO NAO SEJA NULO)
→ IMPRIME O VALOR DO NÓ ATUAL
→ MOVE PARA O PRÓXIMO NÓ