MCTA028-15: Programação Estruturada

Aula 13: Filas
(Primeira Parte)

Wagner Tanaka Botelho
wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com
Universidade Federal do ABC (UFABC)
Centro de Matemática, Computação e Cognição (CMCC)

Introdução

Introdução

- O conceito de fila é bastante comum para as pessoas:
  - Somos obrigados a enfrentar uma fila sempre que vamos ao banco, cinema, etc.
- Na Computação:
  - Uma fila é um conjunto finito de itens (de um mesmo tipo) esperando por um serviço ou processamento;
  - Um exemplo é o gerenciamento de documentos enviados para a impressora.
- Uma fila, assim como uma lista, é uma sequência de elementos:
  - Entretanto, diferente das listas, os itens de uma fila obedecem a uma ORDEM DE ENTRADA e SAÌDA:
    - Um item só pode ser RETIRADO da fila depois que **TODOS** os itens à frente também forem retirados.

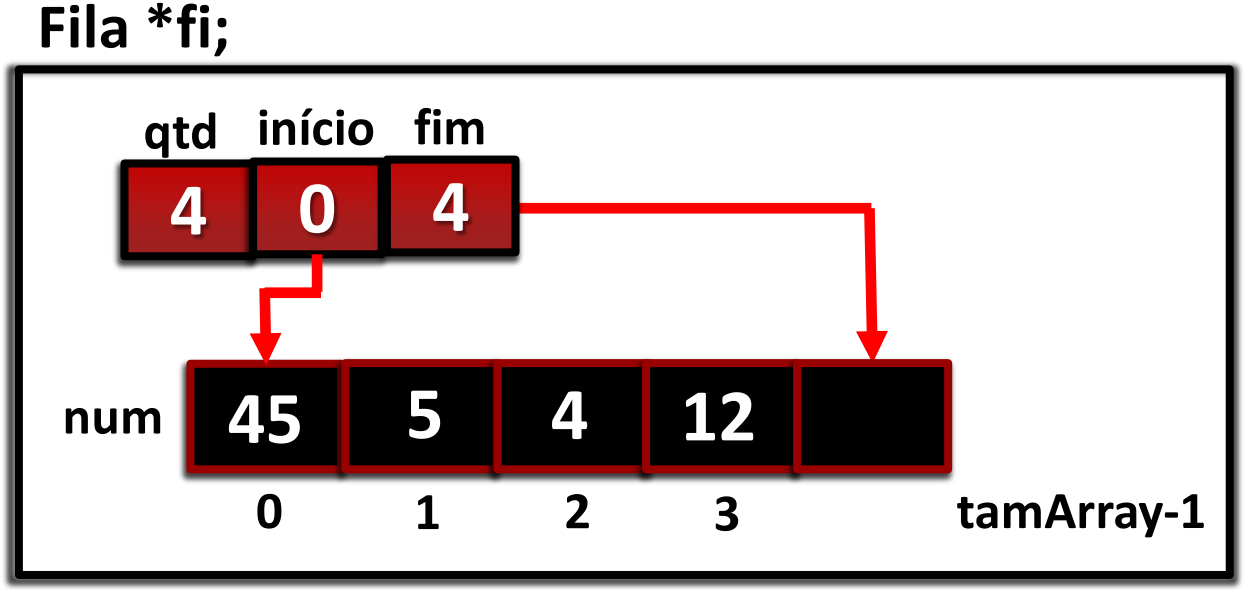
Introdução

- A INSERÇÃO de um item é feita de UM LADO da fila, enquanto a RETIRADA é feita do OUTRO lado:
  - Se quisermos ACESSAR determinado elemento da fila, devemos REMOVER todos os que estiverem à FRENTE dele:
    - Por este motivo, as filas são conhecidas como ESTRUTURAS do tipo **PRIMEIRO A ENTRAR**, **PRIMEIRO A SAIR** ou *First In First Out* (FIFO).

Fila Sequencial Estática

Fila Sequencial Estática

- É uma fila utilizando alocação estática e acesso sequencial dos elementos;
- É o tipo mais simples de fila possível;
- É implementada usando um *array* e **TRÊS** campos adicionais para guardar a QUANTIDADE (qtd) de elementos inseridos, o INÍCIO (inicio) e FINAL (fim) da fila.



Definindo o Tipo

*Ex_01.c

```c
#include<stdio.h>
#include<stdlib.h>

#define tamArray 5

struct fila{
    int qtd;
    int inicio;
    int fim;
    int num[tamArray];
};

typedef struct fila Fila;
```

Definindo uma constante (tamanho do *array*).

Definindo o tipo fila com quatro campos:

+ qtd (int): indica quantidade de elementos inseridos na fila;

+ inicio (int): posição no *array* do elemento que está no início da fila;

+ fim (int): posição no *array* que é o final da fila. Posição disponível para inserir um novo elemento;

+ *array* num do tipo int: tipo de dado a ser armazenado na fila.

Redefinindo a *struct* para encurtar o comando.

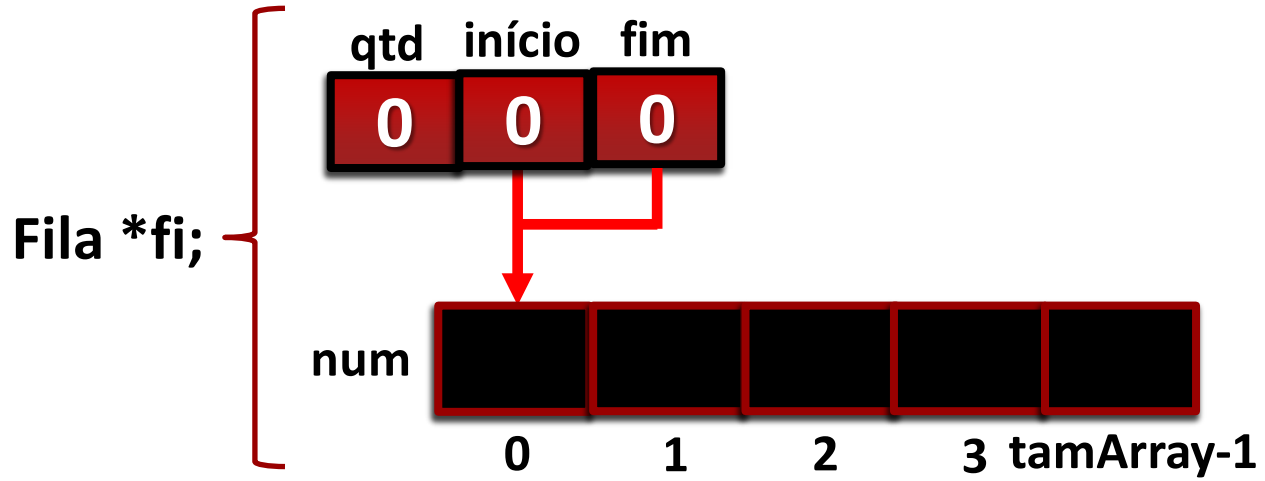
Criando a Fila

```
15  Fila* cria_Fila(){
16      Fila *fi = NULL;
17
18      fi = (Fila *) malloc (sizeof(Fila));
19
20      if(fi != NULL){
21          fi->inicio = 0;
22          fi->qtd = 0;
23          fi->fim = 0;
24      }
25      return fi;
26  }
```

Ponteiro para estrutura fila.

Alocando a área de memória para a fila.

Inicializando as variáveis com zero.



*fi	
qtd	0
inicio	0
fim	0
num	
[0]	-1163005939
[1]	-1163005939
[2]	-1163005939
[3]	-1163005939
[4]	-1163005939

Liberando a Fila

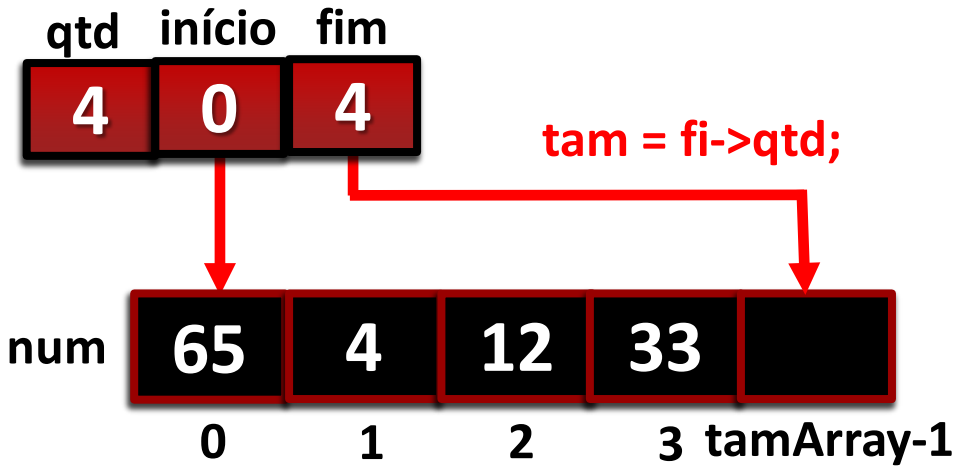
```
29  void libera_Fila(Fila *fi){
30      free(fi);
31  }
```

Tamanho da Fila

```
33  int tamanho_Fila(Fila *fi){
34      int tam = 0;
35
36      if(fi == NULL){
37          return -1;
38      }
39      else{
40          tam = fi->qtd;
41
42          return tam;
43      }
44  }
```

Se for verdade, algum problema aconteceu na criação da fila.

`tam` recebe o valor de `qtd`, ou seja, o tamanho da fila.



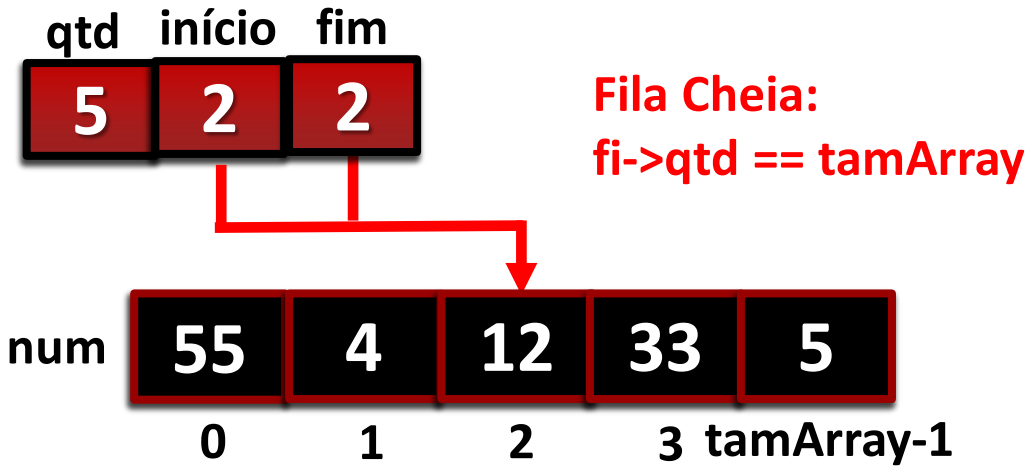
Fila Cheia

```
45  int fila_Cheia(Fila *fi){
46      if(fi==NULL){
47          return -1;
48      }
49      else{
50          if(fi->qtd == tamArray){
51              return 1;
52          }
53          else{
54              return 0;
55          }
56      }
57  }
```

Se ocorreu algum problema na criação da fila, o valor retornado será -1.

A variável **qtd** também é utilizada para saber se a fila está cheia. Basta verificar se **qtd =** ao tamanho do *array* (**tamArray**).

Se a fila não estiver cheia, o valor retornado será 0.



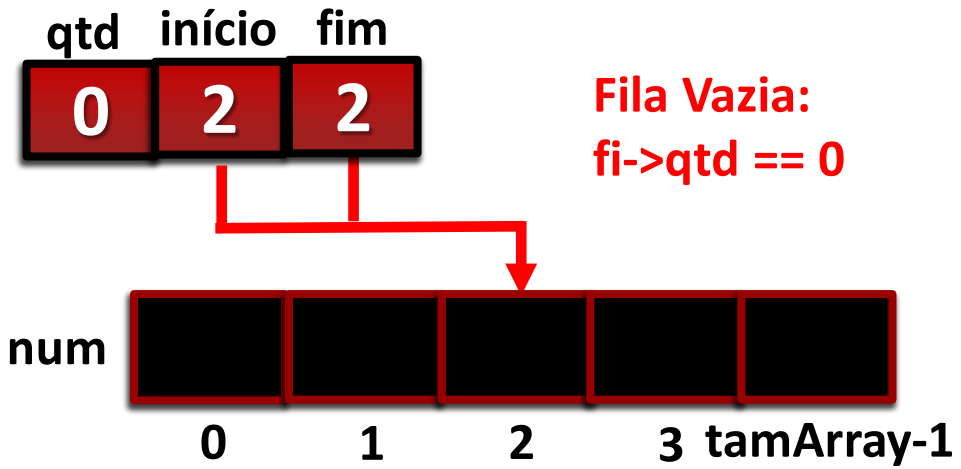
Fila Vazia

```
59  int fila_Vazia(Fila *fi){
60      if(fi == NULL){
61          return -1;
62      }
63      else{
64          if(fi->qtd == 0){
65              return 1;
66          }
67          else{
68              return 0;
69          }
70      }
71  }
```

Se ocorreu algum problema na criação da fila, o valor retornado será -1.

A variável **qtd** também é utilizada para saber se a fila está vazia. Basta verificar se **`qtd = 0`**.

Se a fila não estiver vazia, o valor retornado será 0.



Inserir um Elemento

```
73  int insere_Fila(Fila *fi, int n){
74      if(fi == NULL){
75          return 0;
76      }
77      else{
78          if(fi->qtd == tamArray){
79              return 0;
80          }
81          else{
82              fi->num[fi->fim] = n;
83              fi->fim = fi->fim + 1;
84              fi->qtd++;
85              return 1;
86          }
87      }
88  }
```

Se ocorreu algum problema na criação da fila, o valor retornado será 0.

A INSERÇÃO NA FILA É SEMPRE NO FINAL!!!

Se a fila estiver cheia, o valor retornado será 0.

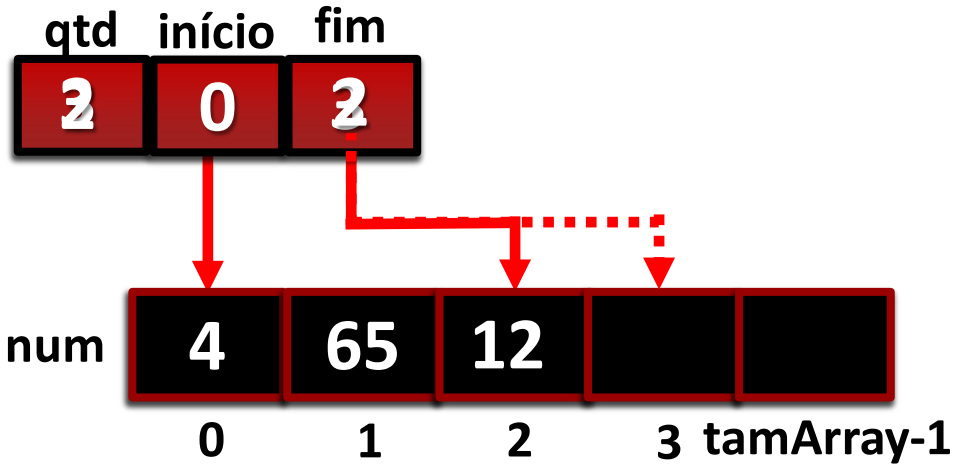
+ n é copiado para a posição apontada para a variável fim (final da fila);
+ o fim é incrementado para indicar a próxima posição vaga na fila;
+ qtd: quantidade de elementos é incrementada.

insere_Fila (pFila, 12);

fi->num[2]=12

fi->fim = 3

fi->qtd = 3



Remover um Elemento

```
90  int remove_Fila(Fila *fi){
91      if(fi == NULL || fi->qtd == 0){
92          return 0;
93      }
94      else{
95          fi->inicio = fi->inicio+1;
96          fi->qtd--;
97
98          return 1;
99      }
100 }
```

Verifica se a fila foi criada e se não tem elementos a serem retirados.

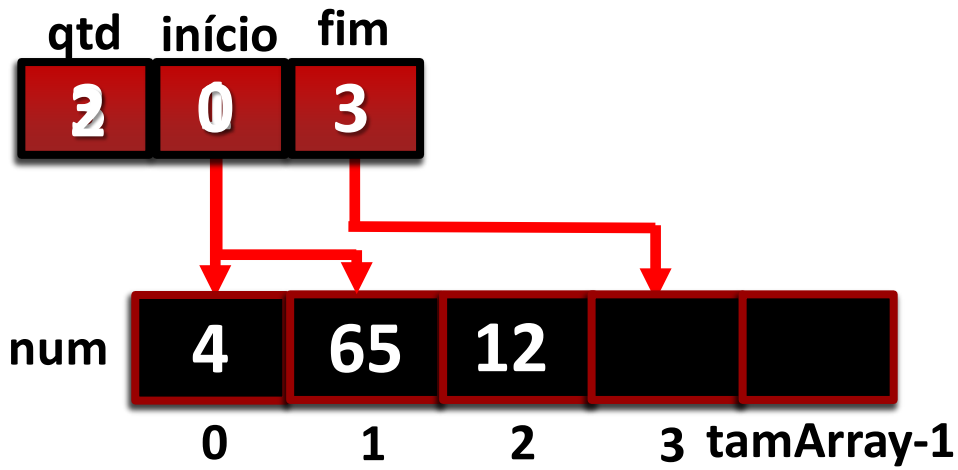
Como a remoção é feita no início da fila, basta incrementar em uma unidade o seu valor.

Decrementando a quantidade de elementos na fila.

remove_Fila (pFila);

fi->inicio = 1

fi->qtd-- = 2



O início da fila fica com um número. Entretanto, a posição é considerada NÃO ocupada por elementos na fila.

Fila Dinâmica Encadeada

Introdução

Introdução

- Fila dinâmica encadeada:
  - Definida utilizando alocação dinâmica;
  - Acesso encadeado dos elementos;
  - Cada elemento é alocado dinamicamente:
    - A medida que os dados são inseridos dentro da fila;
    - Memória é liberada a medida que é removido.
  - Possui uma estrutura contendo DOIS CAMPOS de informação:
    - **Dado:** utilizado para armazenar a informação inserida na fila;
    - **Prox:** ponteiro que indica o próximo elemento da fila.
  - Além da estrutura, a fila utiliza um nó descritor para guardar o INÍCIO, FINAL e a QUANTIDADE DE ELEMENTOS inseridos na fila.

Introdução

- A principal vantagem de se utilizar uma abordagem dinâmica e encadeada na definição da fila é a melhor utilização dos recursos de memória:
  - NÃO sendo mais necessário definir previamente o tamanho da fila.
- Entretanto, a principal desvantagem é a necessidade de percorrer TODA A FILA para destruí-la;
- A implementação de uma fila dinâmica encadeada é praticamente IGUAL à implementação de uma lista dinâmica encadeada:
  - A diferença é que uma fila possui uma regra para INSERÇAO e outra para REMOÇÃO.

Definindo o Tipo

Ex_02.c

```c
#include<stdio.h>
#include<stdlib.h>

struct elemento{
    int numero;
    struct elemento *prox;
};

typedef struct elemento Elem;

struct fila{
    struct elemento *inicio;
    struct elemento *fim;
    int qtd;
};

typedef struct fila Fila;
```

Definindo o tipo que descreve cada elemento da fila.
+ *prox: ponteiro para o próximo elemento da fila;
+ numero: tipo de dado (int) a ser armazenado na fila.

Redefinindo a *struct* para encurtar o comando.

Definindo o tipo que descreve o nó da fila.
+ *inicio: ponteiro que indica o primeiro elemento da fila;
+ *fim: ponteiro que indica o último elemento da fila;
+ qtd (int): armazena o número de elementos dentro da fila.

Redefinindo a *struct* para encurtar o comando.

Criando a Fila

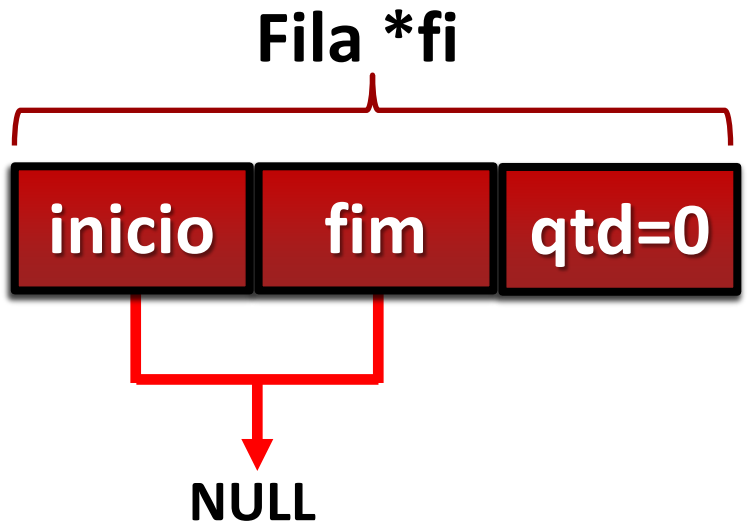
```
19  Fila* cria_Fila(){
20      Fila *fi = NULL;
21
22      fi = (Fila*) malloc(sizeof(Fila));
23
24      if(fi != NULL){
25          fi->inicio = NULL;
26          fi->fim = NULL;
27          fi->qtd = 0;
28      }
29      return fi;
30  }
```

Alocando uma área de memória para a fila (struct fila);

Aponta para o elemento que está no final da fila;

Aponta para o elemento que está no início da fila;

Indica a quantidade de elementos na fila. Recebe ZERO, pois nenhum elemento foi inserido na fila.

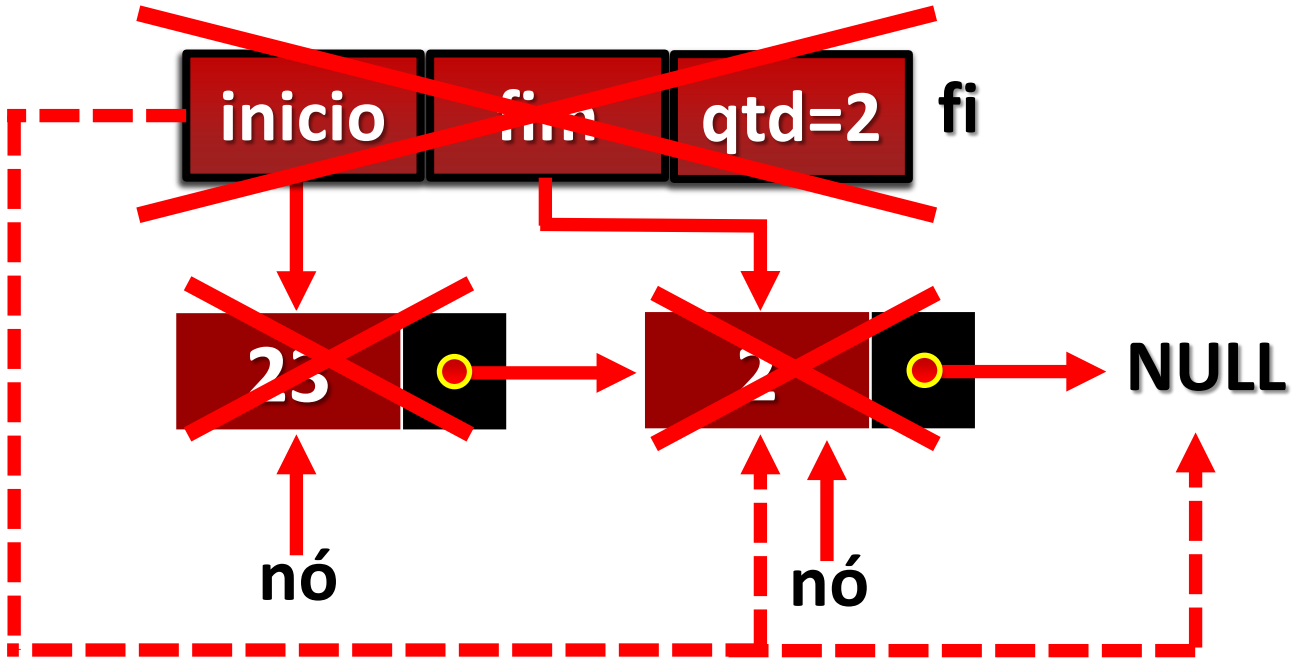


“Destruindo” a Fila

```
32  void libera_Fila(Fila *fi){
33      if(fi != NULL){
34          Elem *no = NULL;
35
36          while(fi->inicio != NULL){
37              no = fi->inicio;
38              fi->inicio = fi->inicio->prox;
39              free(no);
40          }
41          free(fi);
42      }
43  }
```

Se a fila foi criada com sucesso.

Cada elemento da fila é percorrido, enquanto o conteúdo do INÍCIO da fila for diferente de NULL.



Tamanho da Fila

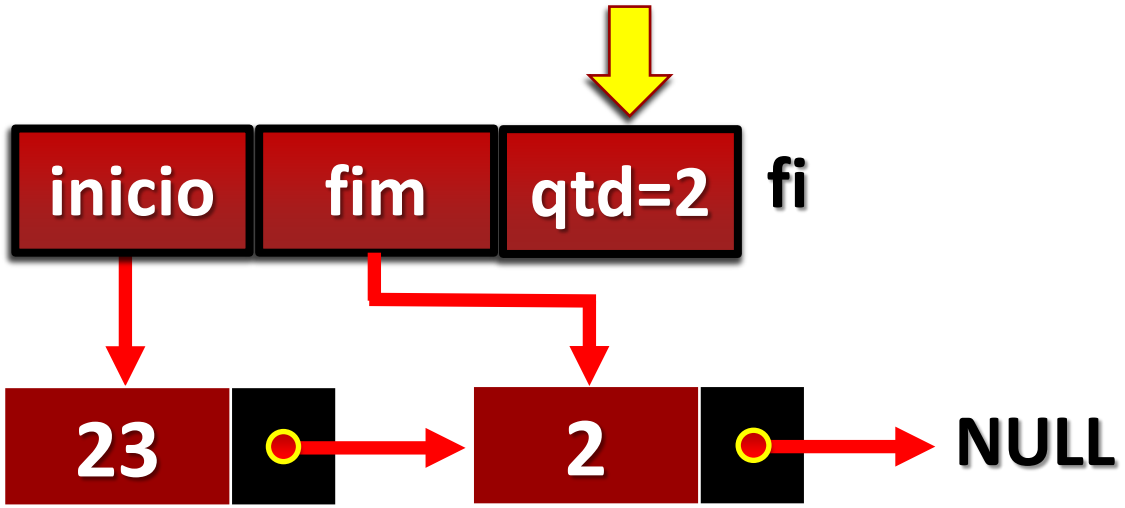
Tamanho da Fila

- Saber o tamanho de uma fila dinâmica encadeada é uma tarefa relativamente simples:
  - Basta ler o nó **qtd** que indica a quantidade de elementos inseridos na fila.

```
45  int tamanho_Fila(Fila *fi){
46      int tam = 0;
47
48      if(fi == NULL){
49          return 0;
50      }
51      else{
52          tam = fi->qtd;
53          return tam;
54      }
55  }
```

Se a fila NÃO foi criada com sucesso.

Acessando o campo qtd da fila.



Fila Cheia

Fila Cheia

- Na fila dinâmica encadeada somente será considerada **CHEIA** quando **NÃO** tiver mais memória disponível para alocar novos elementos:
  - Apenas ocorrerá quando a chamada da função **`malloc()`** retornar **NULL**.

Fila Vazia

Lista Vazia

- Uma fila dinâmica encadeada é considerada **VAZIA** sempre que o conteúdo do seu "**INÍCIO**" apontar para a constante **NULL**.

```
57  int fila_Vazia(Fila *fi){
58      if(fi == NULL){
59          return -1;
60      }
61      else{
62          if(fi->inicio == NULL){
63              return 1;
64          }
65          else{
66              return 0;
67          }
68      }
69  }
```

Verifica se a fila foi criada com sucesso.

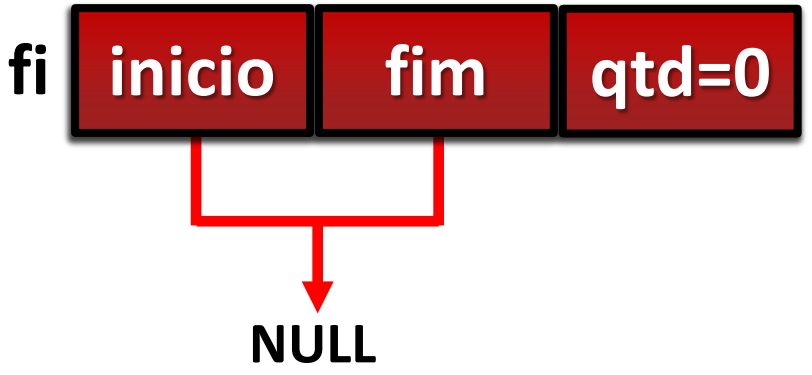
Acessa o conteúdo do início para comparar com NULL.

Se a fila estiver vazia.

Fila NÃO vazia.

Fila Vazia:

fi->inicio == NULL



Inserindo um Elemento na Fila

```
75      else{
76          Elem *no = NULL;
77          no = (Elem*) malloc(sizeof(Elem));
78
79          if(no == NULL){
80              return 0;
81          }
82          else{
83              no->numero = num;
84              no->prox = NULL;
85
86              if(fi->fim == NULL){
87                  fi->inicio = no;
88              }
89              else{
90                  fi->fim->prox = no;
91              }
92
93              fi->fim = no;
94              fi->qtd++;
95
96              return 1;
97          }
98      }
96          return 1;
97      }
98  }
99  }
```

criada com sucesso.

Se não foi possível alocar memória.

Copiando (para o no) o número recebido como parâmetro.

Como a inserção é no final da fila, o elemento a ser inserido obrigatoriamente deve apontar para a constante NULL.

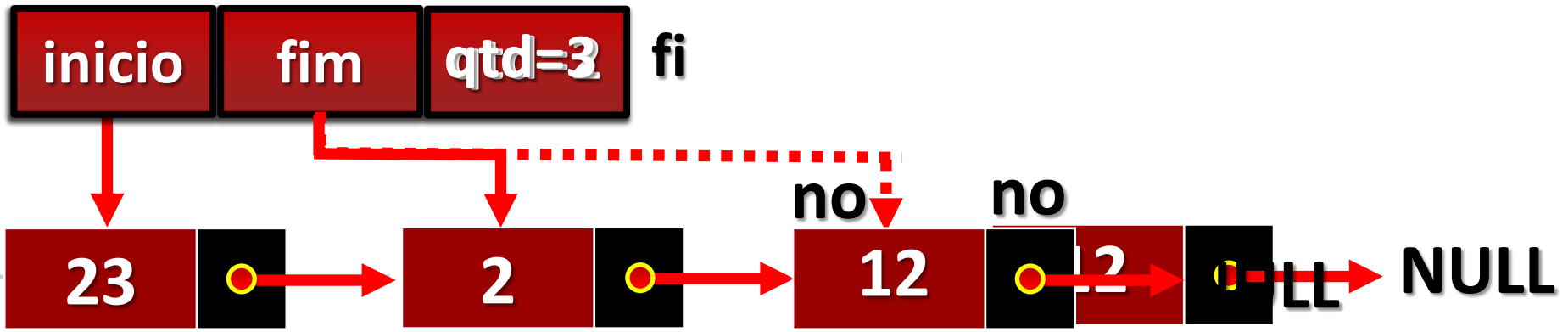
Se a fila estiver vazia.

Deve-se mudar o conteúdo do "início" da fila (fi->inicio) para que ele passe a ser o elemento no.

Elemento do final da fila (fim) deve apontar para o novo elemento.

Conteúdo do "final" (fim) da fila (fi->final) passa a ser o elemento no.

Incrementa a quantidade (qtd) de elementos da fila.



Removendo um Elemento da Fila

```
109         else{
110             Elem *no = NULL;
111
112             no = fi->inicio;
113             fi->inicio = fi->inicio->prox;
114             free(no);
115
116             if(fi->inicio == NULL){
117                 fi->fim = NULL;
118             }
119
120             fi->qtd--;
121             return 1;
122         }
```

criada com sucesso.

No aponta para o inicio da fila.

Inicio da fila aponta para o elemento seguinte.

estiver vazia.

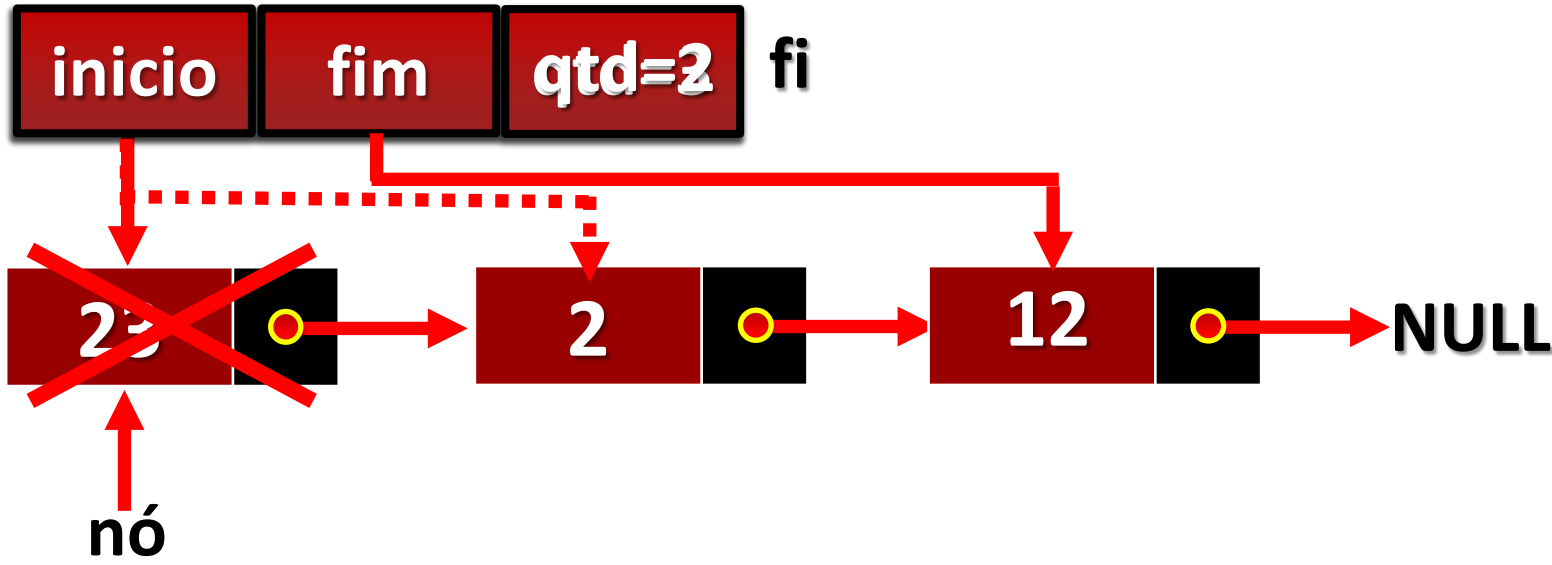
Liberando a memória (no).

Se após a remoção, a fila ficou vazia.

O fim deve ser apontado para NULL.

Decrementando a quantidade de elementos.

```
116         if(fi->inicio == NULL){
117             fi->fim = NULL;
118         }
119
120         fi->qtd--;
121         return 1;
122     }
123 }
124 }
```



Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011;
- BACKES, André. Estrutura de Dados Descomplicada em Linguagem C. GEN LTC. 2016.