

# Estrutura de Dados

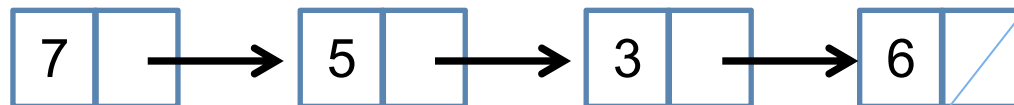
## **Estrutura Fila**

**(Implementação Dinâmica/Encadeada)**

**Prof. Luiz Gustavo Almeida Martins**

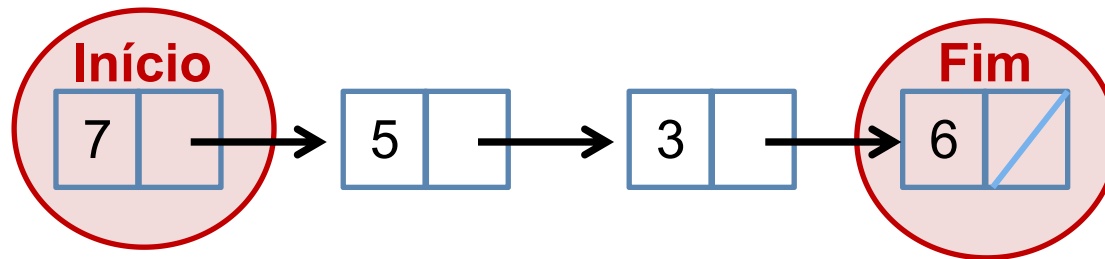
# Introdução

- Aloca e libera cada elemento individualmente nas operações de inserção e remoção
  - Utiliza **alocação dinâmica**



# Introdução

- Aloca e libera cada elemento individualmente nas operações de inserção e remoção
  - Utiliza **alocação dinâmica**
- **Problema:** eficiência depende do **acesso rápido e direto** às extremidades da fila
  - Inserção precisa conhecer o **final da fila**
  - Remoção precisa conhecer o **início da fila**



# Formas de Implementação

- Existem **2 soluções**:
  - Abordagem 1: encadeamento simples
  - Abordagem 2: encadeamento circular

# Fila com Encadeamento Simples

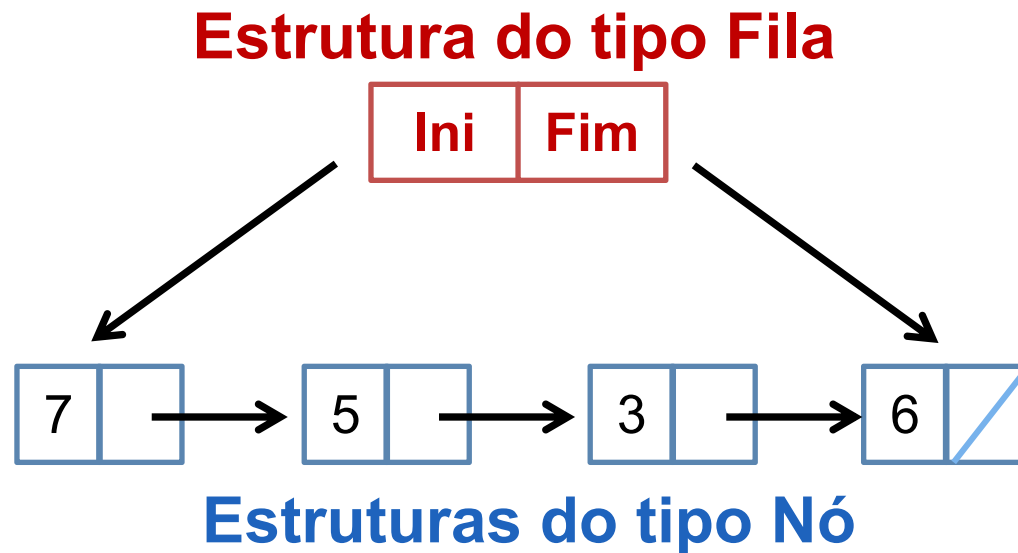
- **Forma de representação:**
  - **Estrutura nó** não tem alteração
    - Campo *info* para armazenar o **valor do elemento**
    - Campo *prox* para apontar o próximo nó (**sucessor**)

# Fila com Encadeamento Simples

- **Forma de representação:**
  - **Estrutura nó** não tem alteração
    - Campo *info* para armazenar o **valor do elemento**
    - Campo *prox* para apontar o próximo nó (**sucessor**)
  - **Fila** representada como uma **estrutura composta por 2 ponteiros**:
    - Um ponteiro endereça o **1º nó da fila**
    - Outro ponteiro endereça o **último nó da fila**

# Fila com Encadeamento Simples

- Forma de representação:



# Fila com Encadeamento Simples

- Implementação em C (**fila de inteiros**):

**fila.c**

```
struct no {  
    int info;  
    struct no * prox;  
};
```

```
struct fila {  
    struct no * ini;  
    struct no * fim;  
};
```



# Fila com Encadeamento Simples

- Implementação em C (**fila de inteiros**):

**fila.c**

```
struct no {  
    int info;  
    struct no * prox;  
};
```

```
struct fila {  
    struct no * ini;  
    struct no * fim;  
};
```

**fila.h**

```
typedef struct fila * Fila;
```

# Fila com Encadeamento Simples

- Implementação em C (**fila de inteiros**):

**fila.c**

```
struct no {  
    int info;  
    struct no * prox;  
};
```

```
struct fila {  
    struct no * ini;  
    struct no * fim;  
};
```

**fila.h**

```
typedef struct fila * Fila;
```

O uso do \* é opcional.

Foi mantido devido à praticidade na codificação das operações (**mantém o estilo do código**)

# Fila com Encadeamento Simples

- **Exemplo:**

Início:



**ini** = NULL

**fim** = NULL

# Fila com Encadeamento Simples

- **Exemplo:**

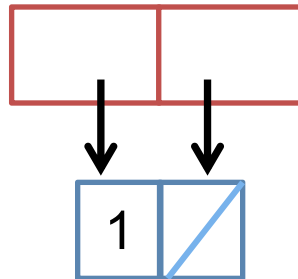
criação (início):



ini = NULL

fim = NULL

insere(1):



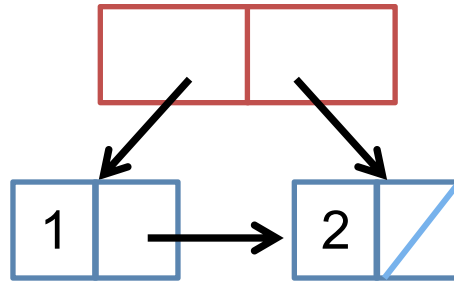
ini = end. do nó 1

fim = end. do nó 1

# Fila com Encadeamento Simples

- **Exemplo:**

insere(2):



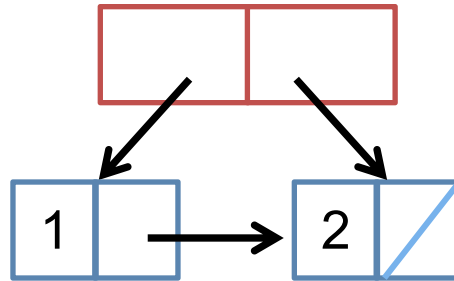
ini = end. do nó 1

**fim** = end. do nó 2

# Fila com Encadeamento Simples

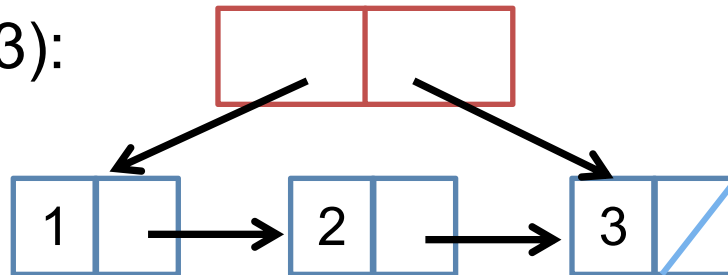
- Exemplo:

insere(2):



ini = end. do nó 1  
fim = end. do nó 2

insere(3):



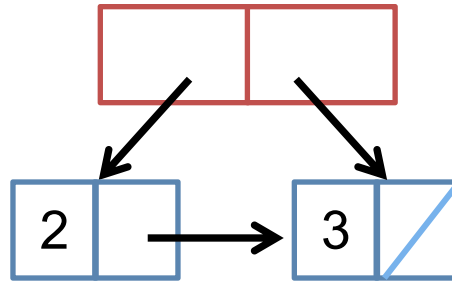
ini = end. do nó 1  
**fim** = end. do nó 3

# Fila com Encadeamento Simples

- **Exemplo:**

remove(&x):

$x = 1$



$ini$  = end. do nó 2

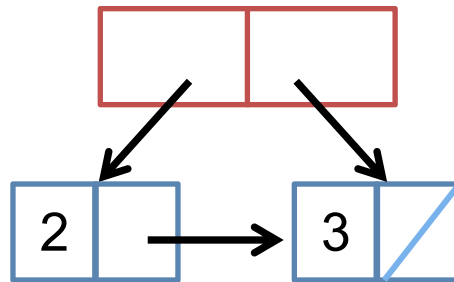
$fim$  = end. do nó 3

# Fila com Encadeamento Simples

- Exemplo:

remove(&x):

$x = 1$

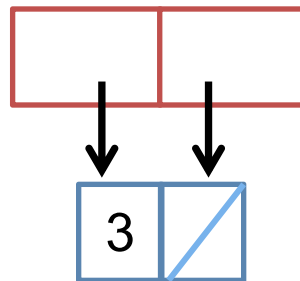


$ini = \text{end. do nó 2}$

$fim = \text{end. do nó 3}$

remove(&x):

$x = 2$



$ini = \text{end. do nó 3}$

$fim = \text{end. do nó 3}$

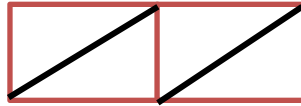


# Fila com Encadeamento Simples

- **Exemplo:**

remove(&x):

$x = 3$



$ini = \text{NULL}$

$fim = \text{NULL}$

**FILA** VOLTA AO ESTADO DE **VAZIA**

# Fila com Encadeamento Simples

- Operação **cria\_fila()**:
  - Aloca uma estrutura fila

# Fila com Encadeamento Simples

- Operação **cria\_fila()**:
  - Aloca uma estrutura fila
  - *Coloca a fila no estado de vazia*
    - Inicializa os campos *ini* e *fim* com **NULL**

# Fila com Encadeamento Simples

- Operação **cria\_fila()**:
  - Aloca uma estrutura fila
  - *Coloca a fila no estado de vazia*
    - Inicializa os campos *ini* e *fim* com **NULL**

```
Fila cria_fila() {  
    Fila f;  
    f = (Fila) malloc(sizeof(struct fila));  
    if (f != NULL) {  
        f->ini = NULL;  
        f->fim = NULL;  
    }  
    return f;  
}
```

# Fila com Encadeamento Simples

- Operação **cria\_fila()**:
  - Aloca uma estrutura fila
  - *Coloca a fila no estado de vazia*
    - Inicializa os campos *ini* e *fim* com **NULL**

```
Fila cria_fila() {  
    Fila f;  
    f = (Fila) malloc(sizeof(struct fila));  
    if (f != NULL) {  
        f->ini = NULL;  
        f->fim = NULL;  
    }  
    return f;  
}
```

**OBS:** Se **não adotar o \*** no **typedef** do **fila.h**, deve-se colocá-lo na frente das 2 ocorrências do tipo **Fila** (declaração do retorno da função e da variável **f**); e usar ponto (**f.**) ao invés da seta (**f->**) na inicialização dos campos **ini** e **fim**.

# Fila com Encadeamento Simples

- Operação **fila\_vazia()**:
  - Verifica se a fila está no estado de vazia
    - Usa um dos campos (*ini* ou *fim*) e verifica se é **NULL**

```
int fila_vazia(Fila f) {  
    if (f->ini == NULL)  
        return 1;  
    else  
        return 0;  
}
```

# Fila com Encadeamento Simples

- Operação **fila\_vazia()**:
  - Verifica se a fila está no estado de vazia
    - Usa um dos campos (*ini* ou *fim*) e verifica se é **NULL**

```
int fila_vazia(Fila f) {  
    if (f->ini == NULL)  
        return 1;  
    else  
        return 0;  
}
```

- Não existe operação **fila\_cheia()**

# Fila com Encadeamento Simples

- Operação **insere\_fim()**:
  - Existem **2 cenários** possíveis:
    - Fila vazia
    - Fila com elementos



# Fila com Encadeamento Simples

- Operação **insere\_fim()**:
  - Existem **2 cenários** possíveis:
    - Fila vazia
    - Fila com elementos
  - Diferem em um único comando (operação)

# Fila com Encadeamento Simples

- Operação **insere\_fim()**:
  - Aloca uma novo nó
  - Preenche seus os campos:
    - Campo **info** recebe o valor do elemento
    - Campo **prox** recebe **NULL**
  - **Se fila vazia:**
    - Faz o **campo ini** da fila apontar para o novo nó
  - **Se fila com elementos:**
    - Faz o **último nó** da fila apontar para o novo nó
  - Faz o campo **fim** da fila apontar para o novo nó

# Fila com Encadeamento Simples

- Implementação em C:

```
int insere_fim(Fila f, int elem) {  
    struct no * N;  
    N = (struct no *) malloc(sizeof(struct no));  
    if (N == NULL)  
        return 0;  
    N->info = elem; // Preenche campo info  
    N->prox = NULL; // Preenche campo prox  
    if (fila_vazia(f) == 1)  
        f->ini = N; // Se fila vazia  
    else  
        (f->fim)->prox = N; // Se fila com elementos (NÃO vazia)  
    f->fim = N; // Campo fim aponta para N  
    return 1;  
}
```

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Existem **3 cenários** possíveis:
    - Fila vazia
    - Fila com um único elemento
    - Fila com mais de um elemento

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Existem **3 cenários** possíveis:
    - Fila vazia
    - Fila com um único elemento
    - Fila com mais de um elemento
  - Na fila vazia não existe nenhum elemento e a operação falha

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Existem **3 cenários** possíveis:
    - Fila vazia
    - Fila com um único elemento
    - Fila com mais de um elemento
  - Na fila vazia não existe nenhum elemento e a operação falha
  - Demais cenários diferem por um comando
    - Fila com um elemento **executa um comando extra**

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Cria um ponteiro *auxiliar* que recebe o endereço do 1º nó da fila (*aux = f->ini*)

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Cria um ponteiro *auxiliar* que recebe o endereço do 1º nó da fila ( $aux = f \rightarrow ini$ )
  - Faz campo *ini* da fila apontar o **sucessor** do nó endereçado por *aux* ( $f \rightarrow ini = aux \rightarrow prox$ )



# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Cria um ponteiro *auxiliar* que recebe o endereço do 1º nó da fila (*aux = f->ini*)
  - Faz campo *ini* da fila apontar o **sucessor** do nó endereçado por *aux* (*f->ini = aux->prox*)
  - Se fila com um único elemento:
    - Faz campo *fim* apontar para **NULL** (*f->fim = NULL*)

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Cria um ponteiro *auxiliar* que recebe o endereço do 1º nó da fila (*aux = f->ini*)
  - Faz campo *ini* da fila apontar o **sucessor** do nó endereçado por *aux* (*f->ini = aux->prox*)
  - Se fila com um único elemento:
    - Faz campo *fim* apontar para **NULL** (*f->fim = NULL*)
  - Retorna o valor do nó removido (*\*e = aux->info*)

# Fila com Encadeamento Simples

- Operação **remove\_ini()**:
  - Cria um ponteiro *auxiliar* que recebe o endereço do 1º nó da fila (*aux = f->ini*)
  - Faz campo *ini* da fila apontar o **sucessor** do nó endereçado por *aux* (*f->ini = aux->prox*)
  - **Se fila com um único elemento:**
    - Faz campo *fim* apontar para **NULL** (*f->fim = NULL*)
  - Retorna o valor do nó removido (*\*e = aux->info*)
  - Libera a memória usada pelo nó removido

# Fila com Encadeamento Simples

- Implementação em C:

```
int remove_ini(Fila f, int *elem) {  
    if (fila_vazia(f) == 1)  
        return 0;  
    struct no *aux = f->ini; // Aux aponta para o 1º nó  
    *elem = aux->info; // Retorna valor do elemento  
  
    // Verifica se a fila tem um único nó  
    if (f->ini == f->fim)  
        f->fim = NULL;  
  
    f->ini = aux->prox; // Retira 1º nó da fila  
    free(aux); // Libera a memória alocada  
    return 1;  
}
```

# Fila com Encadeamento Circular

- Adota a **MESMA** forma de representação usada em **lista linear circular**

# Fila com Encadeamento Circular

- Adota a **MESMA** forma de representação usada em **lista linear circular**
  - **Estrutura nó** formada por:
    - Campo *info* para armazenar o **valor do elemento**
    - Campo *prox* para apontar o próximo nó (**sucessor**)

# Fila com Encadeamento Circular

- Adota a **MESMA** forma de representação usada em **lista linear circular**
  - **Estrutura nó** formada por:
    - Campo *info* para armazenar o **valor do elemento**
    - Campo *prox* para apontar o próximo nó (**sucessor**)
  - **Fila é um **ponteiro**** para o **último nó da fila**
    - **Último nó** aponta para o **1o nó** da fila

# Fila com Encadeamento Circular

- Implementação em C (**fila de inteiros**):

**fila.c**

```
struct no {  
    int info;  
    struct no * prox;  
};
```

**fila.h**

```
typedef struct no * Fila;
```



# Fila com Encadeamento Circular

- **Exemplos:**

Fila vazia:

**F → NULL**

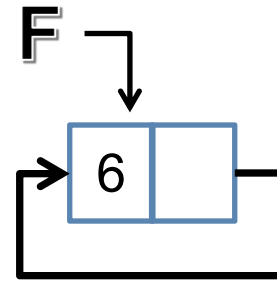
# Fila com Encadeamento Circular

- **Exemplos:**

Fila vazia:

**F** → **NULL**

Fila com um único elemento:



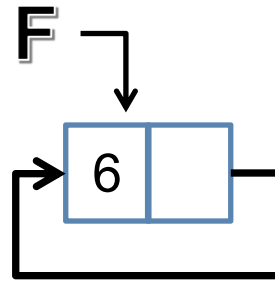
# Fila com Encadeamento Circular

- **Exemplos:**

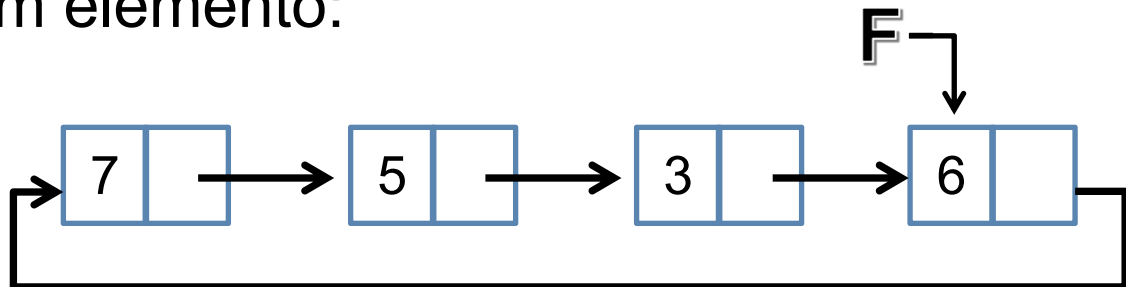
Fila vazia:

**F** → **NULL**

Fila com um único elemento:



Fila com + de um elemento:



# Fila com Encadeamento Circular

- Operações também são **IDÊNTICAS** àquelas apresentadas em lista circular

# Fila com Encadeamento Circular

- Operações também são **IDÊNTICAS** àquelas apresentadas em lista circular

```
Fila cria_fila() {  
    return NULL;  
}
```

# Fila com Encadeamento Circular

- Operações também são **IDÊNTICAS** àquelas apresentadas em lista circular

```
Fila cria_fila() {  
    return NULL;  
}
```

```
int fila_vazia (Fila f) {  
    if (f == NULL)  
        return 1;  
    else  
        return 0;  
}
```

# Fila com Encadeamento Circular

- Operação **insere\_fim()**:

```
int insere_fim (Fila *f, int elem) {  
    // Aloca um novo nó e preenche campo info  
    Lista N = (Fila) malloc(sizeof(struct no));  
    if (N == NULL) { return 0; } // Falha: nó não alocado  
    N->info = elem; // Insere o conteúdo (valor do elem)  
  
    // Trata fila vazia  
    if (fila_vazia(*f) == 1) {  
        N->prox = N; // Faz o novo nó apontar para ele mesmo  
        *f = N; // Faz a fila apontar para o novo nó (último nó)  
    }  
    ...  
}
```

# Fila com Encadeamento Circular

- Operação **insere\_fim()**:

```
int insere_fim (Fila *f, int elem) {
```

```
    ...
```

```
    // Trata fila com elementos (1 ou +)
```

```
    else {
```

```
        N->prox = (*f)->prox; // Faz o novo nó apontar o 1º nó
```

```
        (*f)->prox = N; // Faz o último nó apontar para o novo nó
```

```
        *f = N; // Faz a fila apontar para o novo nó (último nó)
```

```
    }
```

```
    return 1;
```

```
}
```



# Fila com Encadeamento Circular

- Operação **remove\_ini()**:

```
int remove_ini (Fila *f, int *elem) {  
    // Trata fila vazia  
    if (fila_vazia(*f) == 1)  
        return 0;  
    Fila aux = (*f)->prox; // Faz aux apontar para 1º nó  
    *elem = aux->info; // Retorna valor do nó a ser removido  
    if (*f == (*f)->prox) // Trata fila com 1 único nó  
        *f = NULL;  
    else // Trata fila com + de 1 elemento  
        (*f)->prox = aux->prox;  
    free(aux);  
    return 1;  
}
```

# Exercícios

1. Implementar, utilizando a implementação **dinâmica/encadeada com encadeamento simples**, o TAD fila de números inteiros. Essa implementação deve contemplar as operações básicas: *criar\_fila*, *fila\_vazia*, *fila\_cheia*, *insere\_fim* e *remove\_ini*. Além disso, desenvolva um programa aplicativo que permita ao usuário criar uma fila, inserir e remover elementos, e imprimir a fila.

*Teste este programa com a seguinte seqüência de operações:*

- *Cria fila*
- *Imprime fila*
- *Insere os elementos {4,8,-1,19,2,7,8,5,9,22,45};*
- *Imprime fila*
- *Remove elemento*
- *Imprime fila*
-

# Referências

- *Backes, André, Linguagem C Descomplicada, portal de vídeo-aulas, <https://programacaodescomplicada.wordpress.com/>, acessado em 09/03/2016.*
- *Celes, W., Cerqueira, R. e Rangel, J. L. Introdução a estruturas de dados. Ed. Campus Elsevier, 2004.*