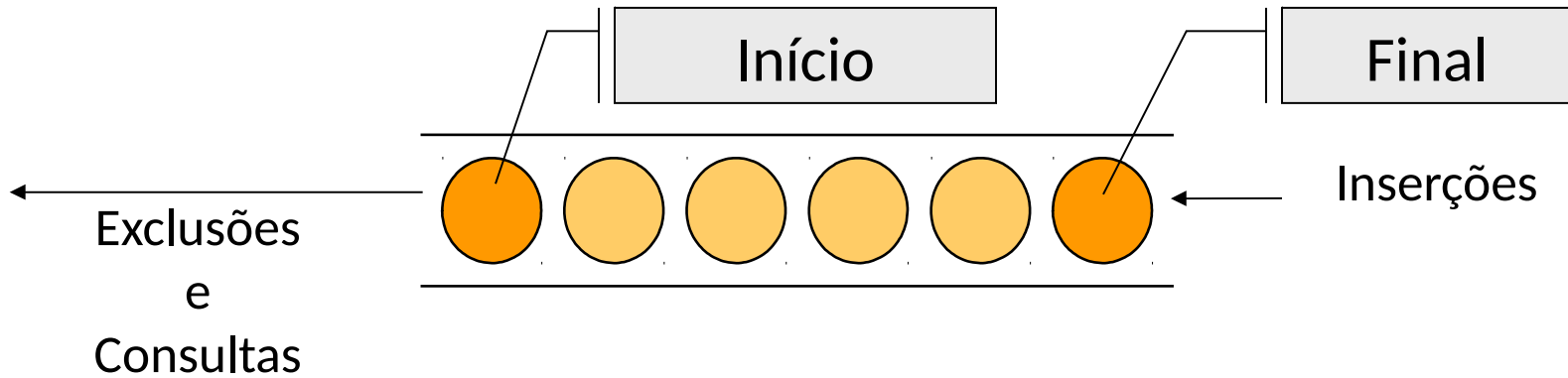


Filas

Prof Nádia Félix e Hebert Coelho
Instituto de Informática
UFG

*Vários slides foram adaptados de Nina Edelwais e Renata Galante
Estrutura de Dados – Série de Livros Didáticos - Informática - UFRGS*



Operações válidas:

- Criar uma fila vazia
- Inserir um nodo no final da fila
- Excluir o nodo do início da fila
- Consultar
- Destruir a fila

Filas

- A inserção de um item é feita de um lado da fila, enquanto a retirada é feita do outro.
 - Primeiro a entrar, primeiro a sair: **FIFO (First in First Out)**
 - **Restrições quanto à manipulação:** inserções sempre no final, remoções sempre do início da fila;
- Exemplo:
 - Gerenciamento de documentos enviados para a impressora

Tipos de Filas

- Basicamente existem dois tipos de implementações para uma fila:
 - Alocação estática com acesso sequencial
 - Alocação dinâmica com acesso encadeado

Filas

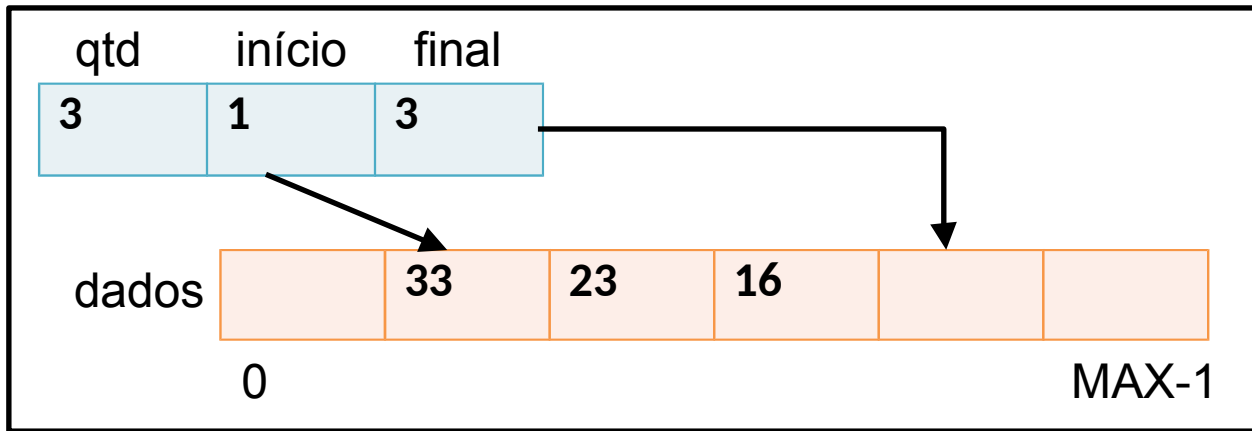
Filas implementadas
por alocação estática

Fila sequencial estática

- Definida utilizando alocação estática e acesso sequencial dos elementos
- Definida utilizando um array
- Implementação em módulos, temos que o usuário tem acesso a apenas um ponteiro do tipo fila
 - Isso impede o usuário de saber como foi implementada a fila e limita o seu acesso a apenas às funções que manipulam o início e o final da fila.

Fila Sequencial

Fila *fi



A principal vantagem de se utilizar um array na definição de uma fila estática é a facilidade de criar e destruir a fila.

Já a sua principal desvantagem é a necessidade de definir previamente o tamanho do array e, conseqüentemente da fila.

Definindo o tipo de fila sequencial estática

- Definir o tipo de dado que será armazenado
- Ponteiro para a estrutura que define a fila
- Conjunto de funções visíveis
- FilaEstatica.h
- FilaEstatica.c

FilaEstatica.h

- O tamanho max do array utilizado na fila, representado pela constante MAX
- O tipo de dado que será armazenado na fila, **struct aluno**
- As funções disponíveis para se trabalhar com essa fila em especial

FilaEstatica.c

- As chamadas às bibliotecas necessárias à implementação da fila
- A definição do tipo que descreve o funcionamento da fila, **struct fila**
- As implementações das funções definidas no arquivo FilaEstatica.h

Note que

- O nosso tipo fila possui quatro campos:
 - início
 - final
 - qtd
 - array do tipo struct aluno, que é o tipo de dado a ser armazenado

FilaEstatica.h

```
1 //Arquivo FilaEstatica.h
2 #define MAX 100
3 struct aluno{
4     int matricula;
5     char nome[30];
6     float n1,n2,n3;
7 };
8
9 typedef struct fila Fila;
10
11 Fila* cria_Fila();
12 void libera_Fila(Fila* fi);
13 int consulta_Fila(Fila* fi, struct aluno *al);
14 int insere_Fila(Fila* fi, struct aluno al);
15 int remove_Fila(Fila* fi);
16 int tamanho_Fila(Fila* fi);
17 int Fila_vazia(Fila* fi);
18 int Fila_cheia(Fila* fi);
19 void imprime_Fila(Fila* fi);
```

FilaEstatica.c

```
#include <stdio.h>
#include <stdlib.h>
#include "FilaEstatica.h" //inclui os Protótipos

//Definição do tipo Fila
struct fila{
    int inicio, final, qtd;
    struct aluno dados[MAX];
};
```

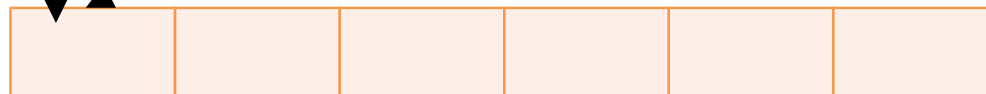
Criando uma fila

```
Fila* cria_Fila(){
    Fila *fi;
    fi = (Fila*) malloc(sizeof(struct fila));
    if(fi != NULL){
        fi->inicio = 0;
        fi->final = 0;
        fi->qtd = 0;
    }
    return fi;
}
```

Fila *fi

| qtd | início | final |
|-----|--------|-------|
| 0 | 0 | 0 |

dados



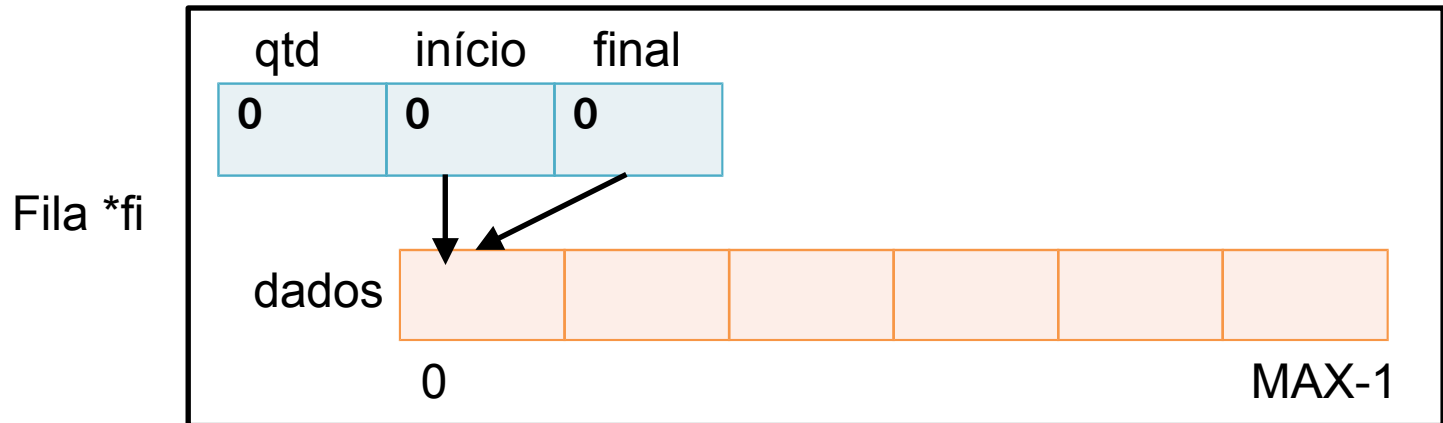
Destruindo a Fila

A small icon of a code editor window with a horizontal line and a vertical line, indicating a cursor position.

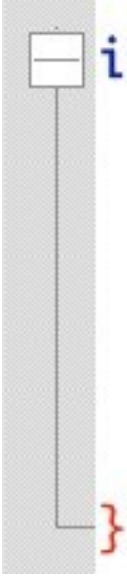
```
void libera_Fila(Fila* fi){  
    free(fi);  
}
```

Tamanho da Fila

```
int tamanho_Fila(Fila* fi){  
    if(fi == NULL)  
        return -1;  
    return fi->qtd;  
}
```

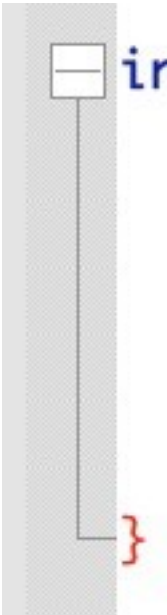


Fila Cheia



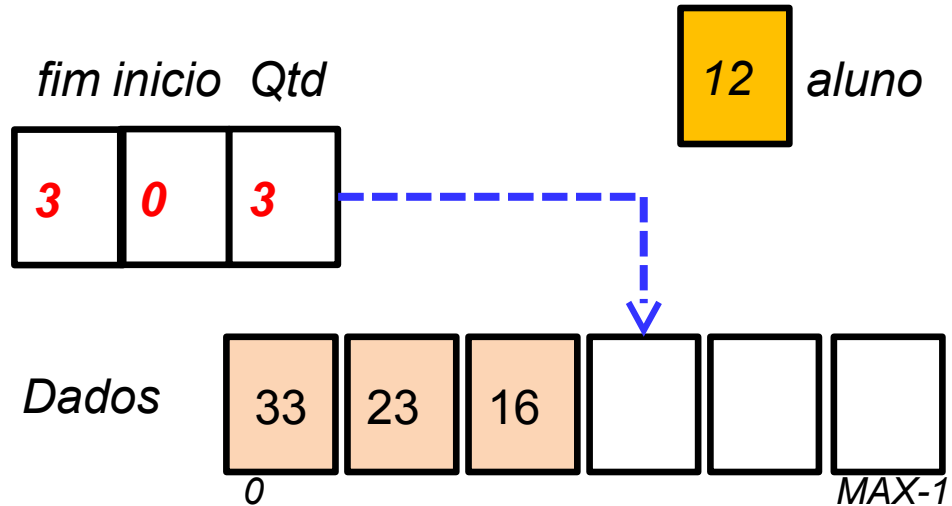
```
int Fila_cheia(Fila* fi){  
    if(fi == NULL)  
        return -1;  
    if (fi->qtd == MAX)  
        return 1;  
    else  
        return 0;  
}
```

Fila Vazia



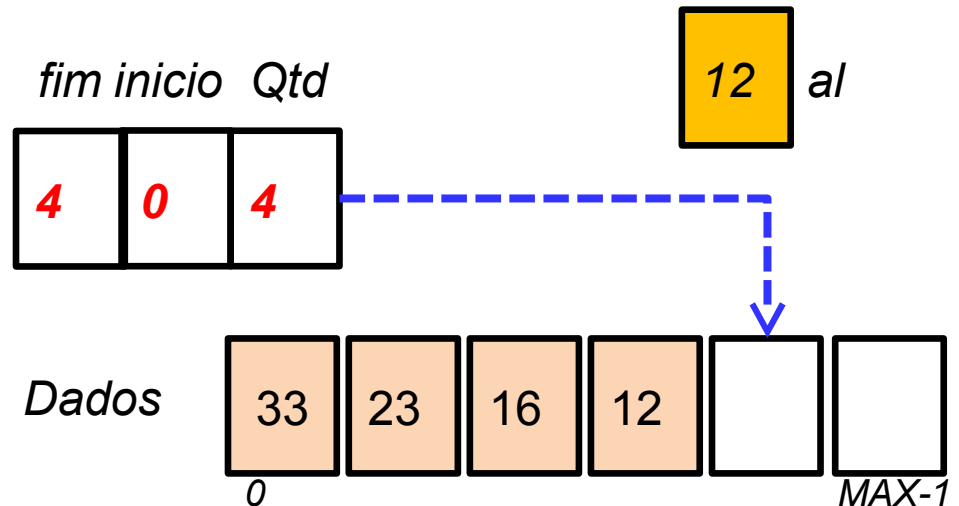
```
int Fila_vazia(Fila* fi){  
    if(fi == NULL)  
        return -1;  
    if (fi->qtd == 0)  
        return 1;  
    else  
        return 0;  
}
```

Inserindo um elemento na fila



Inserindo um elemento na fila

```
int insere_Fila(Fila* fi, struct aluno al){  
    if(fi == NULL)  
        return 0;  
    if(Fila_cheia(fi))  
        return 0;  
    fi->dados[fi->final] = al;  
    fi->final = (fi->final+1)%MAX;  
    fi->qtd++;  
    return 1;  
}
```



Inserindo um elemento na fila

```
int insere_Fila(Fila* fi, struct aluno al){  
    if(fi == NULL)  
        return 0;  
    if(Fila_cheia(fi))  
        return 0;  
    fi->dados[fi->final] = al;  
    fi->final = (fi->final+1)%MAX;  
    fi->qtd++;  
    return 1;  
}
```

| <i>fim</i> | <i>inicio</i> | <i>Qtd</i> |
|--------------|---------------|------------|
| MAX-1 | 1 | 3 |

| |
|----|
| 12 |
|----|

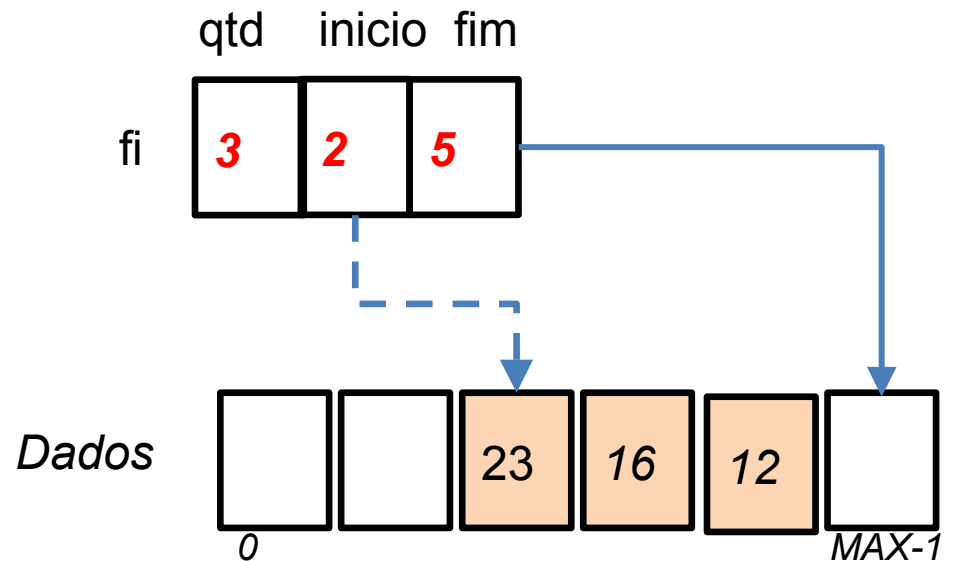
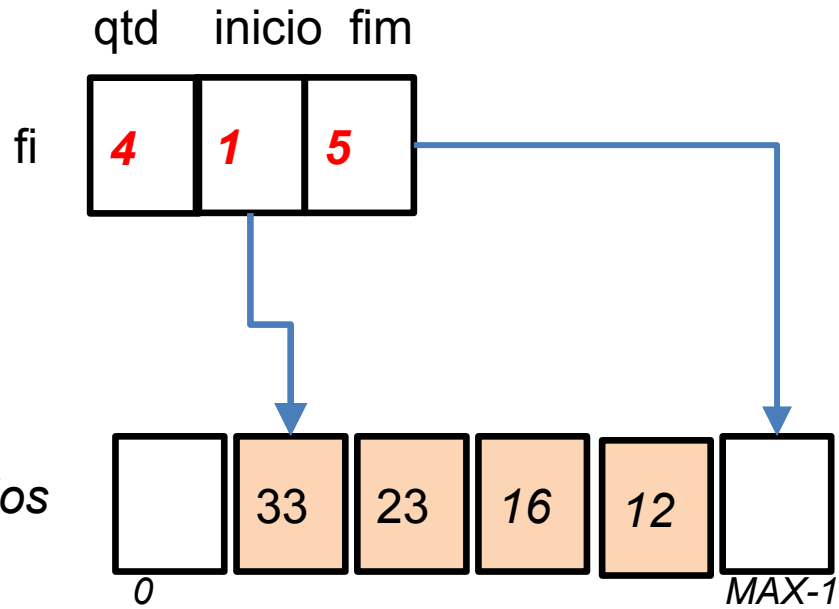
al

A operação de resto da divisão é usada para simular uma fila circular. Assim, ao chegar à posição MAX (que não existe no array), assim o final será colocado na posição ZERO

Dados

| | | | | | |
|---|----|----|----|------|-------|
| | 23 | 16 | 18 | | |
| 0 | | | | | MAX-1 |

Removendo um elemento da fila



Removendo um elemento da fila

```
int remove_Fila(Fila* fi){  
    if(fi == NULL || Fila_vazia(fi))  
        return 0;  
    fi->inicio = (fi->inicio+1)%MAX;  
    fi->qtd--;  
    return 1;  
}
```

Consultando o elemento no início da fila

- Em uma lista, pode-se acessar e recuperar as informações contidas em qualquer um dos seus elementos
- Já na fila, podemos acessar as informações apenas no elemento no **início** da fila.

Consultando o elemento no início da fila

```
int consulta_Fila(Fila* fi, struct aluno *al){  
    if(fi == NULL || Fila_vazia(fi))  
        return 0;  
    *al = fi->dados[fi->inicio];  
    return 1;  
}
```

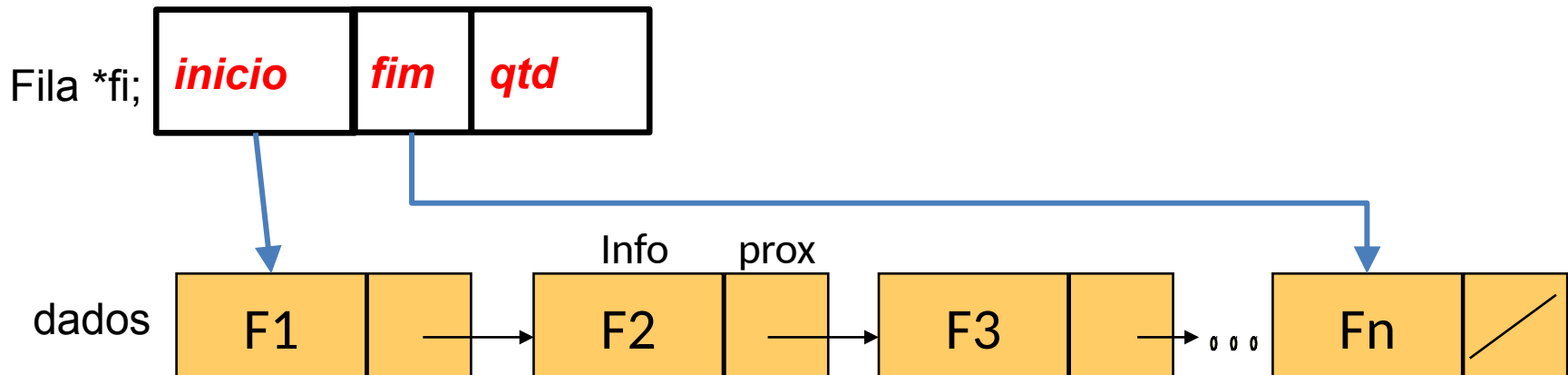
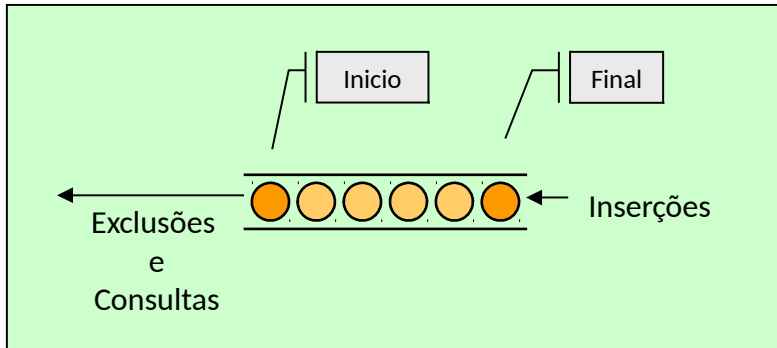
Análise da complexidade

- Operações de Inserção, remoção e consulta envolvem apenas a manipulação de alguns índices, independentemente do número de elementos na fila.
 - **Complexidade das operações é $O(1)$**

Filas

Filas implementadas por encadeamento

Filas implementadas por encadeamento



Para acessar o último nodo, é necessário percorrer toda a fila a partir do primeiro nodo

Operações sobre Filas implementadas por encadeamento com descritor

- Criar uma fila vazia
- Inserir um nodo no final da fila
- Excluir o nodo do início da fila
- Consultar / modificar nodo do início da fila
- Destruir a fila

Definindo o tipo fila dinâmica encadeada

- FilaDin.h
 - Tipo de dado que será armazenado na fila, **struct aluno**
 - struct fila
 - Os protótipos das funções
- FilaDin.c
 - As chamadas às bibliotecas necessárias
 - A definição do tipo que descreve cada elemento da fila
 - A definição do nó descritor
 - A implementação das funções

FilaDin.h

```
1 //Arquivo FilaDin.h
2 struct aluno{
3     int matricula;
4     char nome[30];
5     float n1,n2,n3;
6 };
7
8 typedef struct fila Fila;
9
10 Fila* cria_Fila();
11 void libera_Fila(Fila* fi);
12 int consulta_Fila(Fila* fi, struct aluno *al);
13 int insere_Fila(Fila* fi, struct aluno al);
14 int remove_Fila(Fila* fi);
15 int tamanho_Fila(Fila* fi);
16 int Fila_vazia(Fila* fi);
17 int Fila_cheia(Fila* fi);
18 void imprime_Fila(Fila* fi);
```

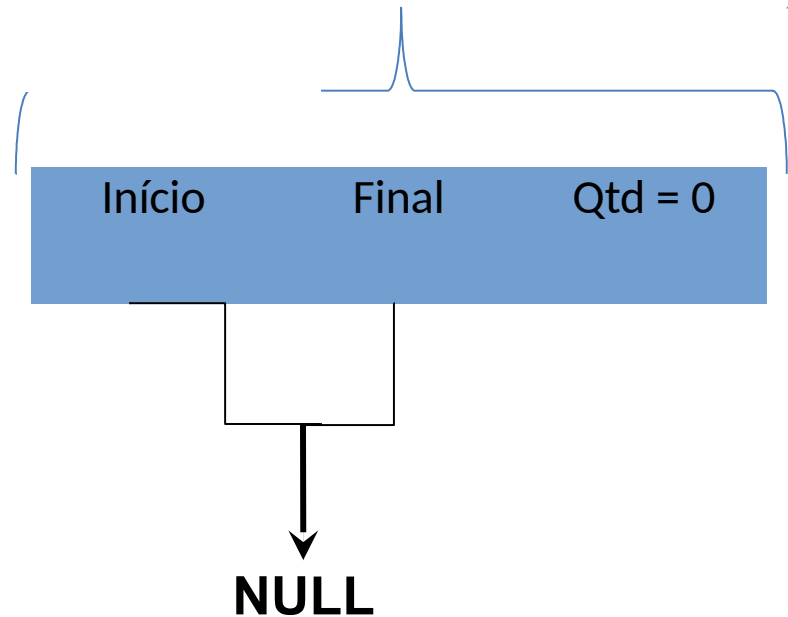
FilaDin.c

```
#include <stdlib.h>
#include "FilaDin.h" //inclui os Protótipos
//Definição do tipo Fila
- struct elemento{
    struct aluno dados;
    struct elemento *prox;
};
typedef struct elemento Elem;
//Definição do Nó Descritor da Fila
- struct fila{
    struct elemento *inicio;
    struct elemento *final;
    int qtd;
};
```

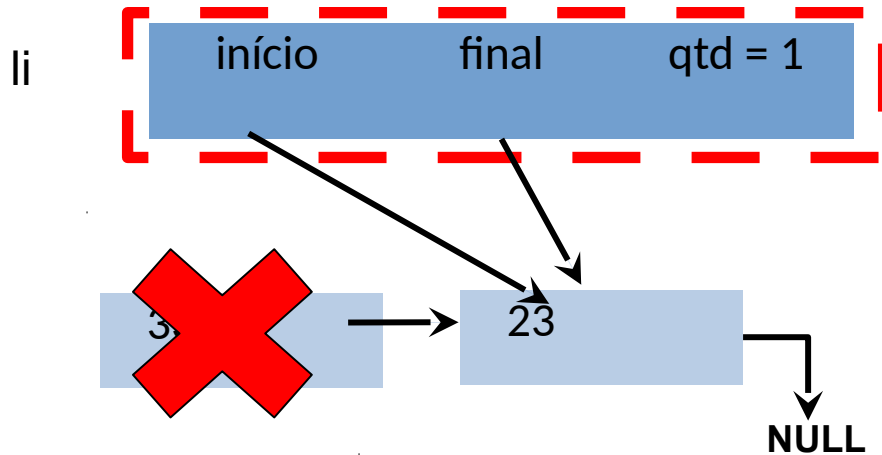
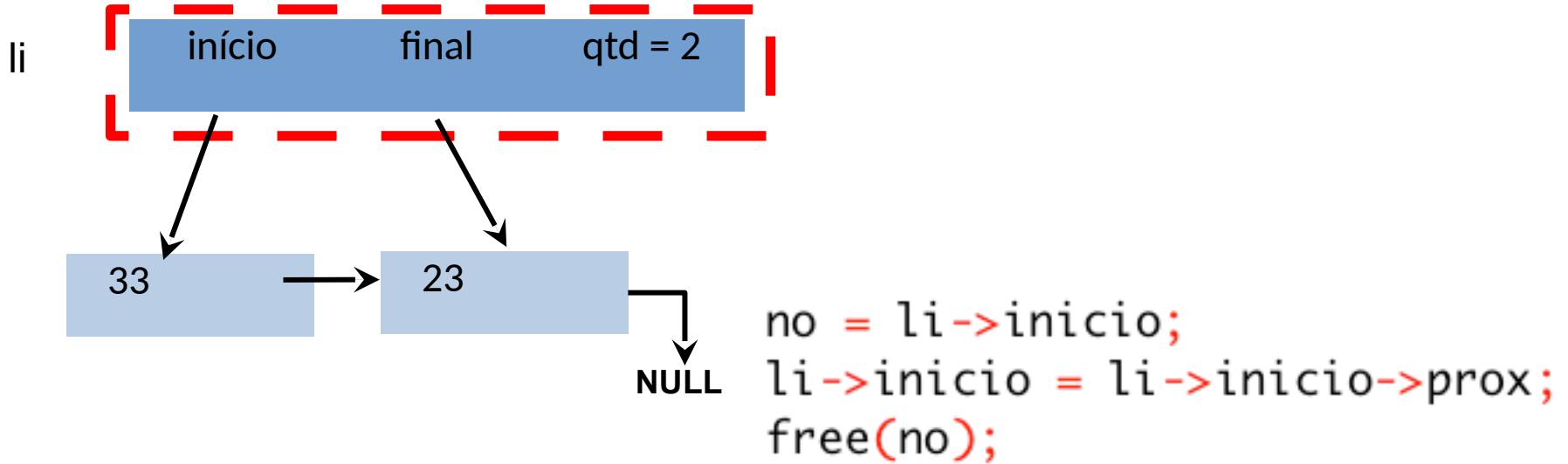

Criando uma fila

```
Fila* cria_Fila(){  
    Fila* fi = (Fila*) malloc(sizeof(Fila));  
    if(fi != NULL){  
        fi->final = NULL;  
        fi->inicio = NULL;  
        fi->qtd = 0;  
    }  
    return fi;  
}
```

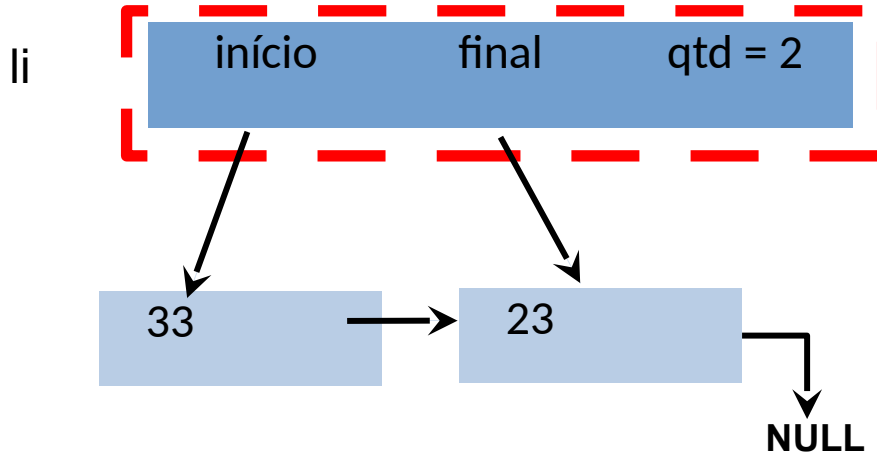
FILA *fi;



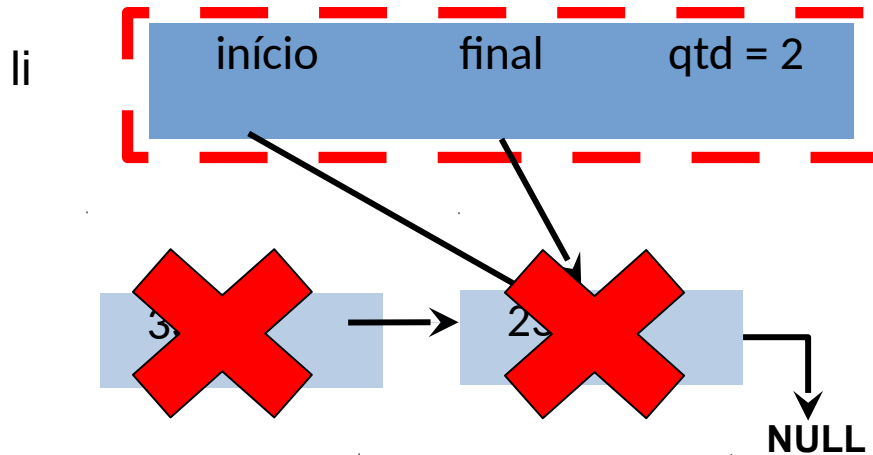
Destruindo uma fila



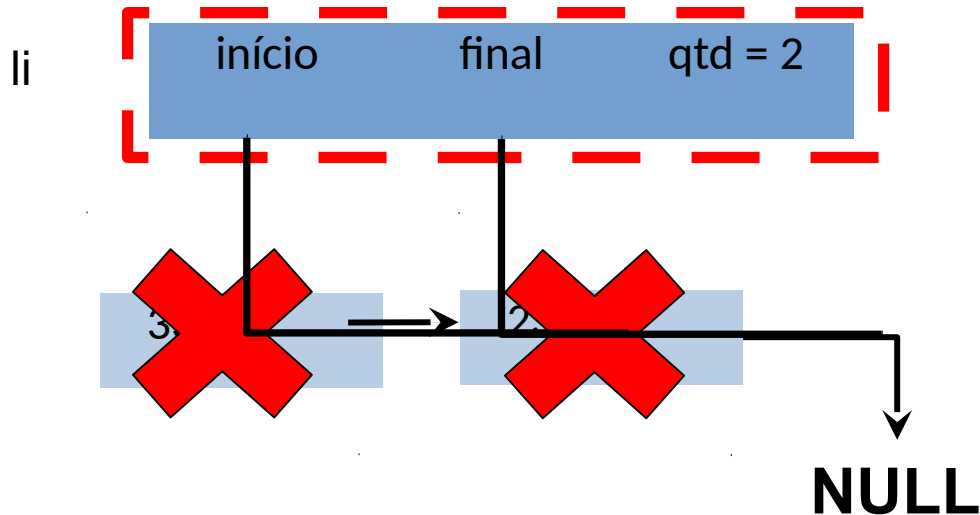
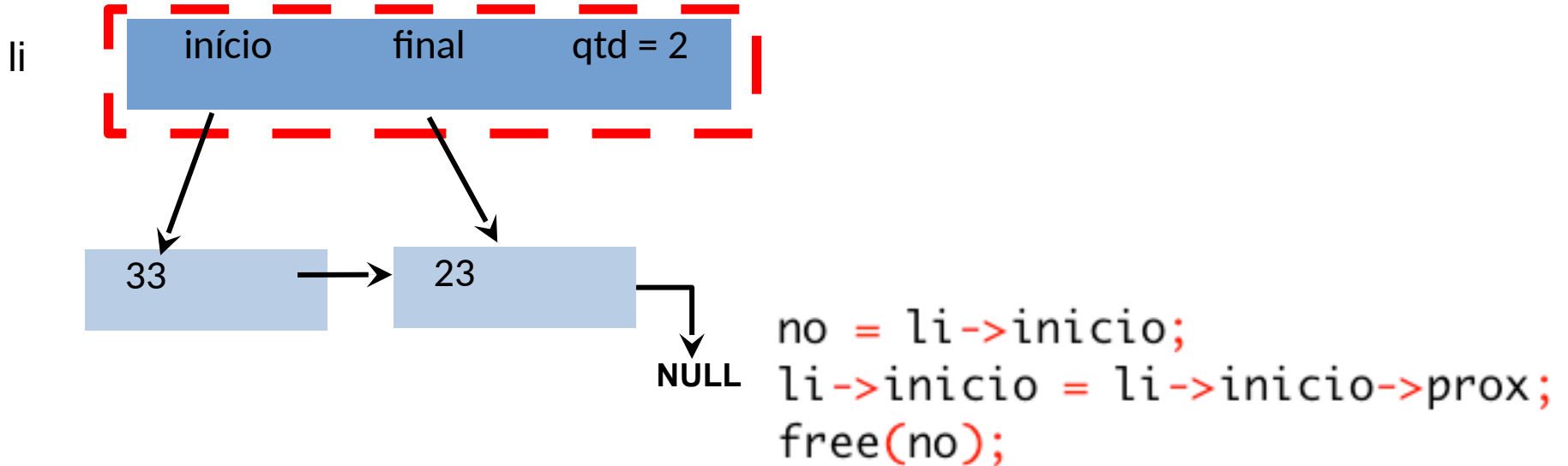
Destruindo uma fila




```
no = li->início;  
li->início = li->início->prox;  
free(no);
```



Destruindo uma fila



Libera_fil




```
void libera_Fila(Fila* fi){
    if(fi != NULL){
        Elem* no;
        while(fi->inicio != NULL){
            no = fi->inicio;
            fi->inicio = fi->inicio->prox;
            free(no);
        }
        free(fi);
    }
}
```

Tamanho da fila

```
int tamanho_Fila(Fila* fi){  
    if(fi == NULL)  
        return 0;  
    return fi->qtd;  
}
```

Fila cheia

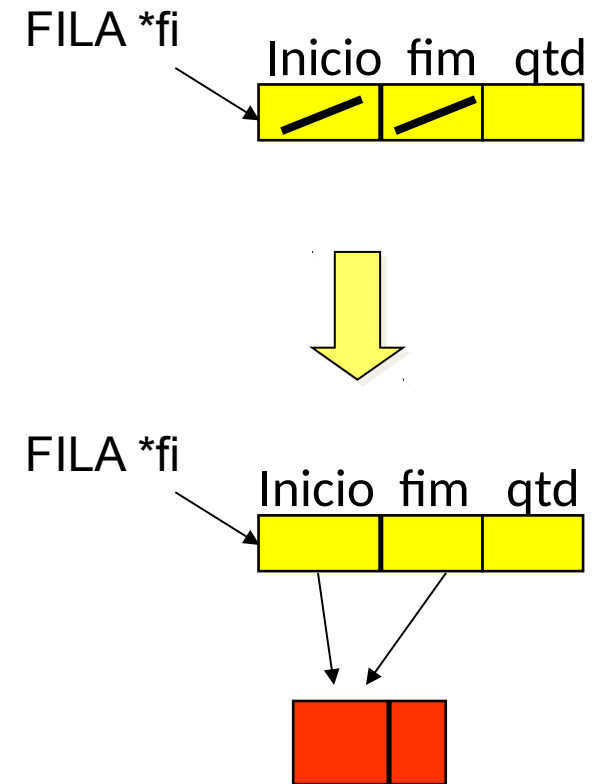
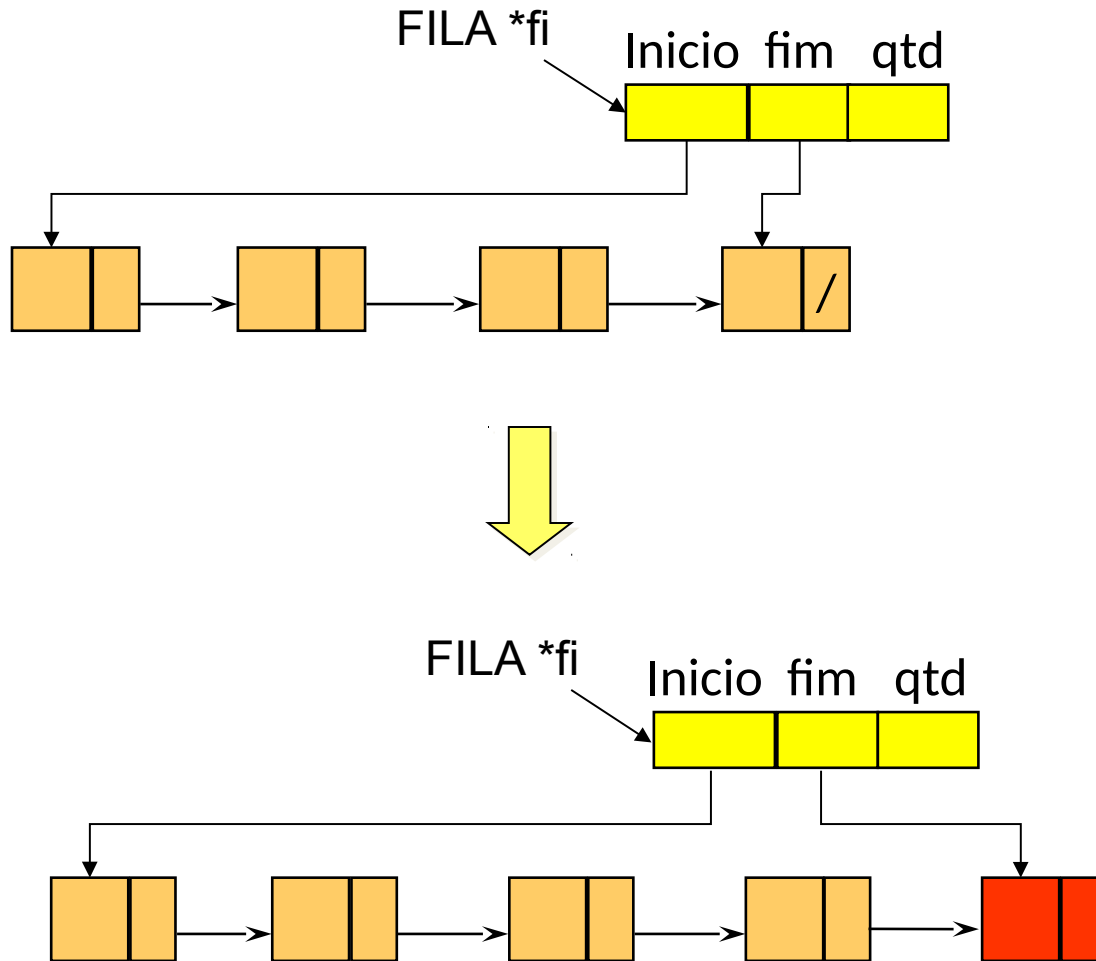


```
int Fila_cheia(Fila* fi){  
    return 0;  
}
```

Fila Vazia

```
int Fila_vazia(Fila* fi){  
    if(fi == NULL)  
        return 1;  
    if(fi->inicio == NULL)  
        return 1;  
    return 0;  
}
```

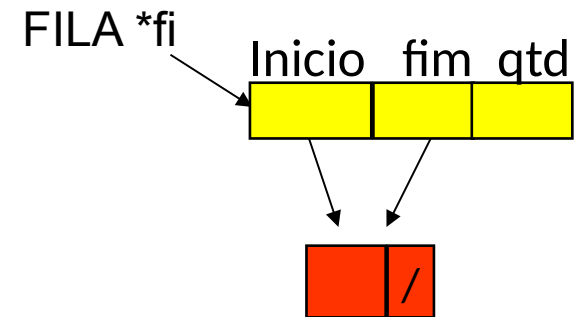
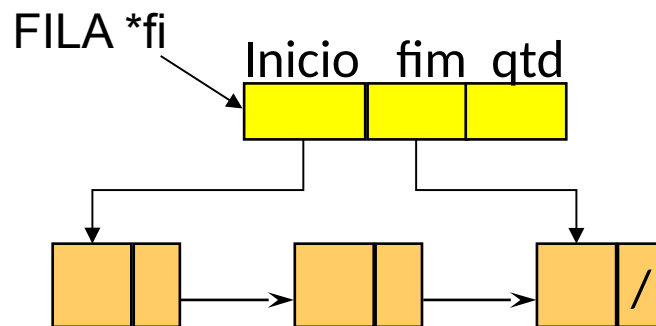
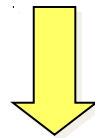
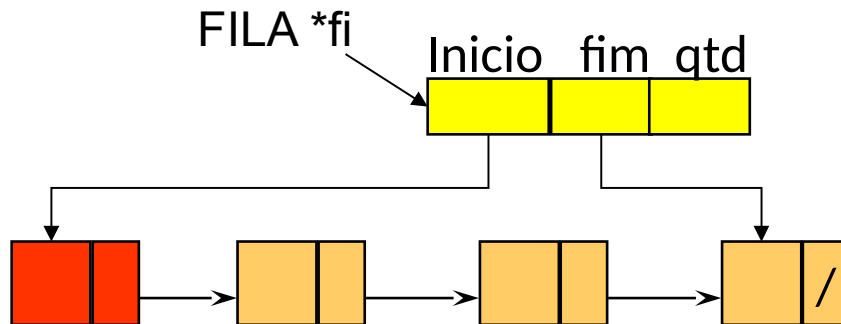

Inserção de um nodo na fila encadeada



Inserção de um nodo na fila encadeada

```
int insere_Fila(Fila* fi, struct aluno al){  
    if(fi == NULL)  
        return 0;  
    Elem *no = (Elem*) malloc(sizeof(Elem));  
    if(no == NULL)  
        return 0;  
    no->dados = al;  
    no->prox = NULL;  
    if(fi->final == NULL)//fila vazia  
        fi->inicio = no;  
    else  
        fi->final->prox = no;  
    fi->final = no;  
    fi->qtd++;  
    return 1;  
}
```

Remoção de um nodo de fila encadeada

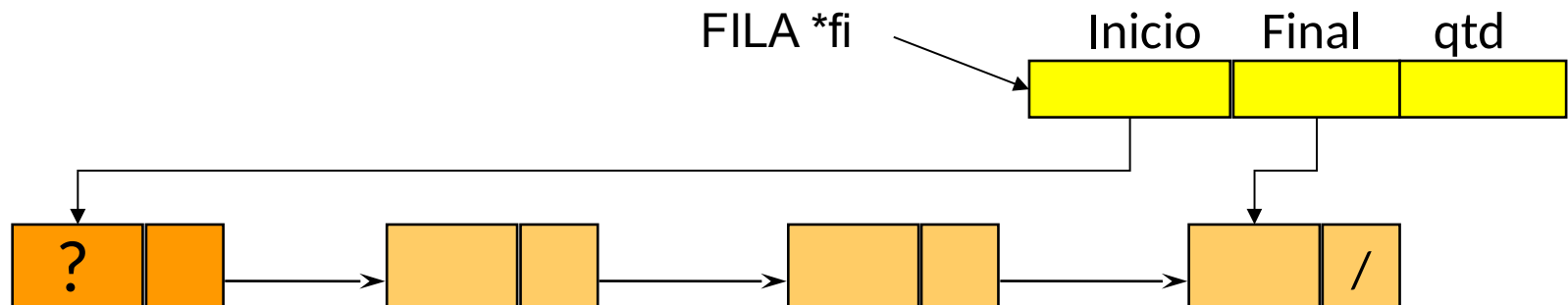


Removendo um elemento da fila

```
int remove_Fila(Fila* fi){  
    if(fi == NULL)  
        return 0;  
    if(fi->inicio == NULL)//fila vazia  
        return 0;  
    Elem *no = fi->inicio;  
    fi->inicio = fi->inicio->prox;  
    if(fi->inicio == NULL)//fila ficou vazia  
        fi->final = NULL;  
    free(no);  
    fi->qtd--;  
    return 1;  
}
```

Consultando um nodo de fila encadeada

- Só o nodo do início da fila pode ser acessado
- Acessado diretamente pelo endereço no descritor



Consulta Fila

```
int consulta_Fila(Fila* fi, struct aluno *al){  
    if(fi == NULL)  
        return 0;  
    if(fi->inicio == NULL)//fila vazia  
        return 0;  
    *al = fi->inicio->dados;  
    return 1;  
}
```

Análise de complexidade

- Inserção, remoção e consulta envolvem apenas a manipulação de ponteiros, independentemente do número de elementos na fila.
 - A complexidade das operações $O(1)$

Exercício

- Crie uma função que imprima os elementos de uma fila

Exercício

- Crie uma função que imprima os elementos de uma fila

Exercício

- Crie uma função que imprima os elementos de uma fila

```
void imprime_Fila(Fila* fi){
    if(fi == NULL)
        return;
    Elem* no = fi->inicio;
    while(no != NULL){
        printf("Matricula: %d\n", no->dados.matricula);
        printf("Nome: %s\n", no->dados.nome);
        printf("Notas: %f %f %f\n", no->dados.n1,
                                           no->dados.n2,
                                           no->dados.n3);
        printf("-----\n");
        no = no->prox;
    }
}
```