FILAS & PILHAS

Prof. André Backes | @progdescomplicada

FILAS

Fila | Definição

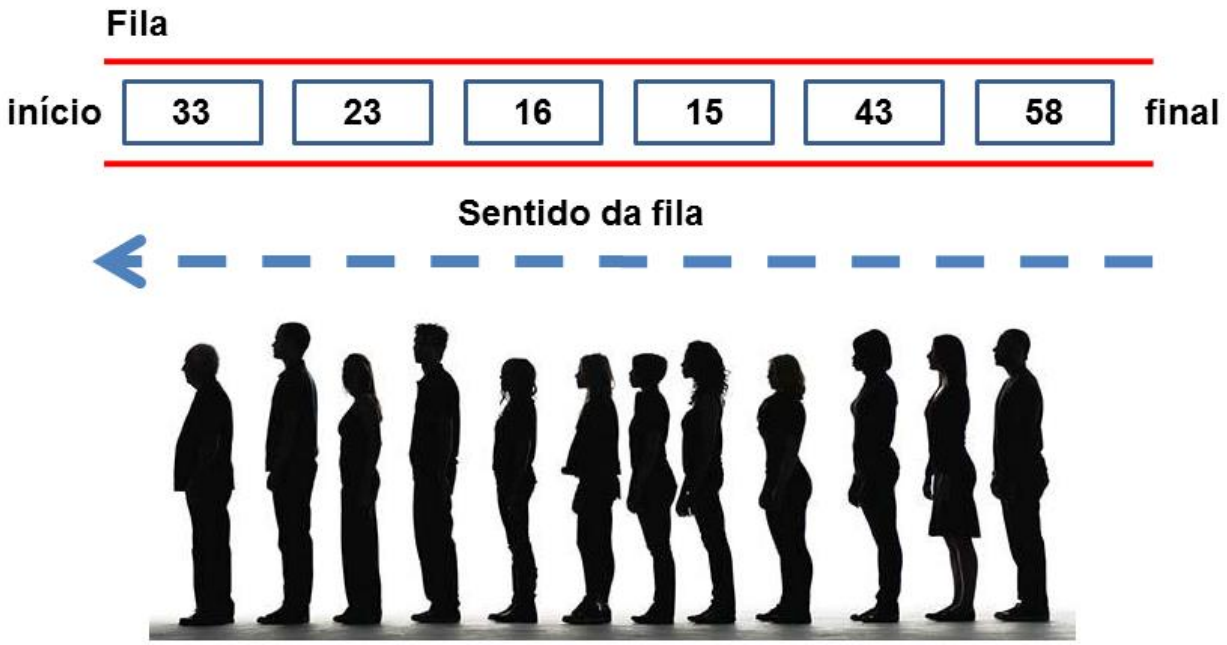
- Conceito muito comum para as pessoas
  - Afinal, somos obrigados a enfrentar uma fila sempre que vamos ao banco, ao cinema, etc

- Na computação, uma fila é um conjunto finito de itens esperando por um serviço ou processamento.
  - Exemplo: gerenciamento de documentos enviados para a impressora

- Trata-se de uma controle de fluxo muito comum na computação

Fila | Definição

- As filas são implementadas e se comportam parecido com as listas. Apenas obedecem uma ordem de entrada e saída
  - A inserção e remoção são realizadas sempre em extremidades distintas
  - São estruturas do tipo **FIFO** (*First In First Out - primeiro a entrar, primeiro a sair*”)



Fila | Definição

- Existem duas implementações principais para uma fila
- Fila estática
  - Os elementos são armazenados de forma consecutiva na memória (array)
  - É necessário definir o número máximo de elementos da fila

- Fila dinâmica
  - O espaço de memória é alocado em tempo de execução
  - A fila cresce e diminui com o tempo
  - Cada elemento da fila armazena o endereço de memória do próximo

Fila | Aplicações

- Filas de impressão
- Processamento de tarefas em sistemas operacionais
- Filas de pacotes em redes
- Armazenamento em buffer
  - Em transmissões ao vivo, um buffer de fila é usado para armazenar temporariamente os dados que chegam para serem reproduzidos na ordem correta

Fila Estática | TAD

- Utiliza um array para armazenar os elementos
  - Vantagem: fácil de criar e destruir
  - Desvantagem: necessidade de definir previamente o tamanho da fila

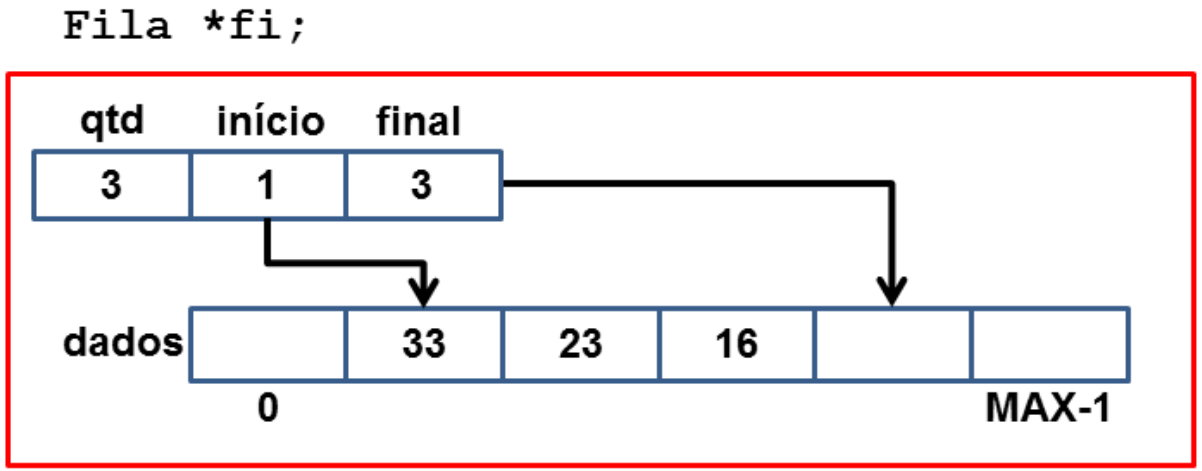
```c
//Definição do tipo Fila
#define MAX 100

struct fila{
    int inicio, final, qtd;
    struct aluno dados[MAX];
};

typedef struct fila Fila;
```

Fila Estática | TAD

- A fila é mantida usando três campos
  - início
  - final
  - quantidade de elementos na fila



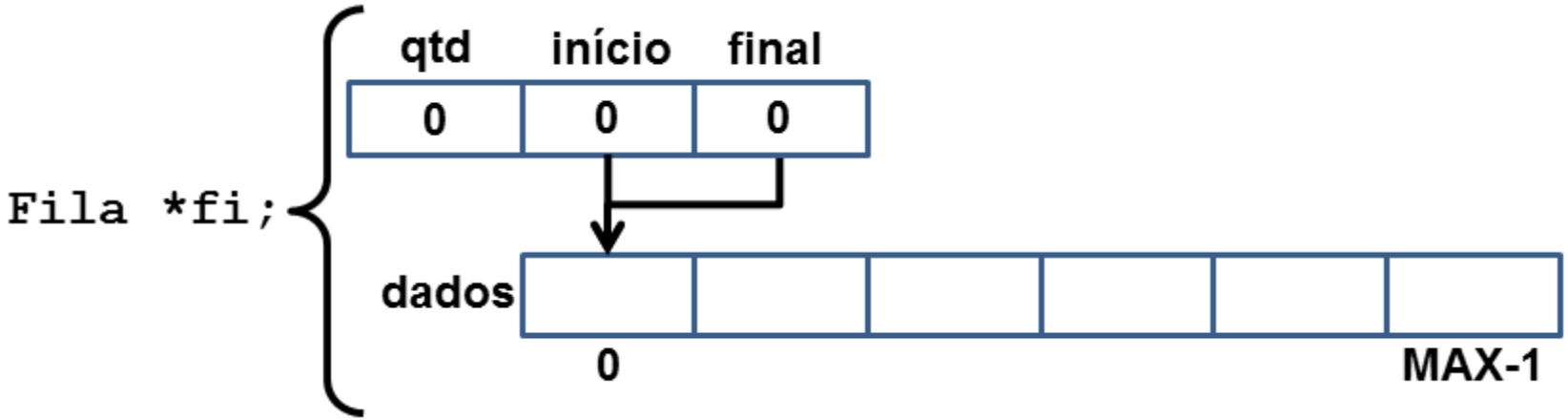
Fila Estática | Criação e liberação

- Criação
  - Aloca uma área de memória para a fila
  - Corresponde a memória necessária para armazenar a estrutura da fila
- Liberação
  - Basta liberar a memória alocada para a estrutura 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;
}

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

Fila Estática | Criação



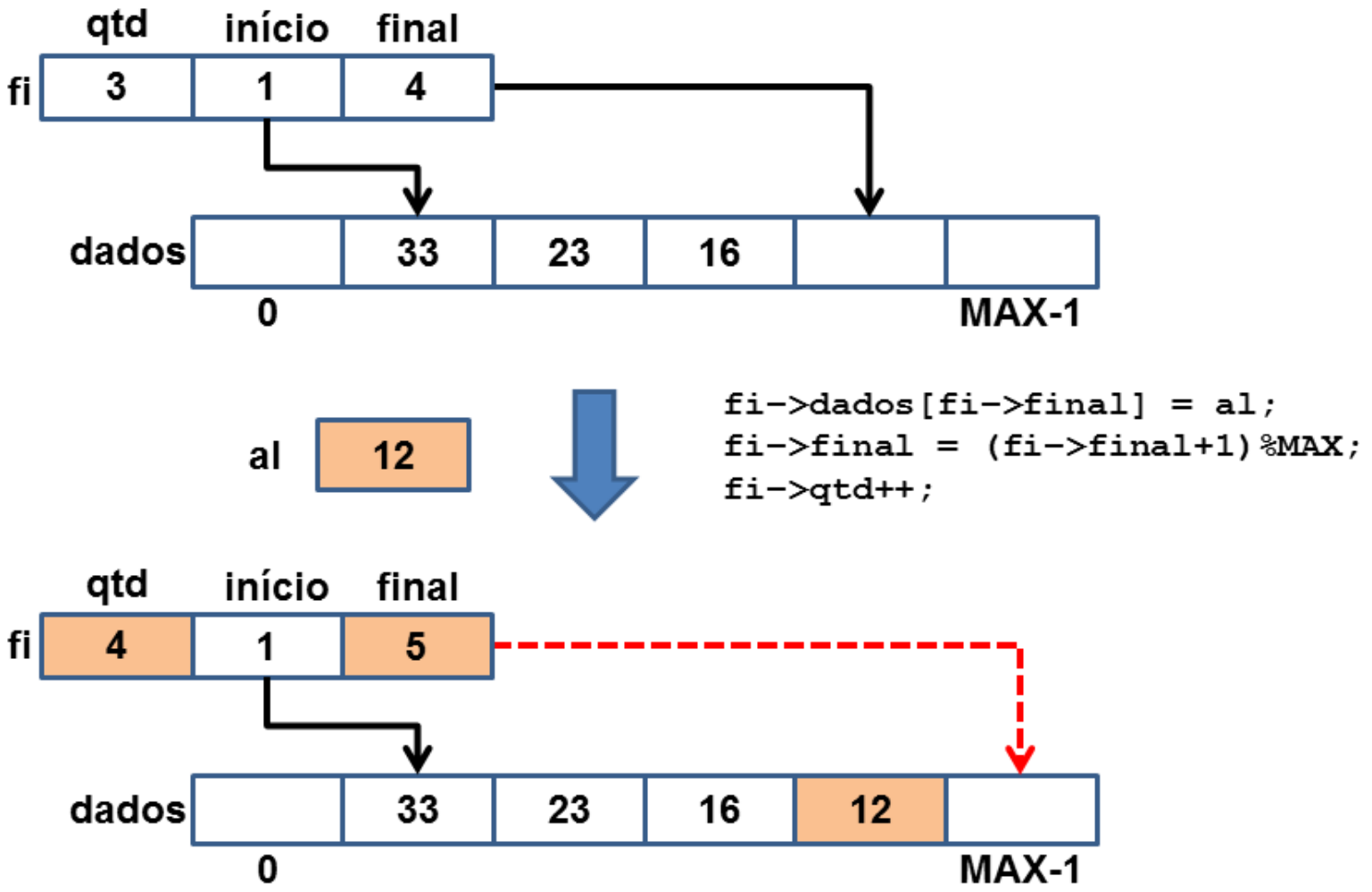
Fila Estática | Inserção

- Tarefa semelhante a inserção no final de uma Lista Sequencial Estática

- Precisamos verificar
  - se a fila existe
  - se a fila está cheia
- E só depois
  - copiar os dados
  - incrementar a quantidade

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

Fila Estática | Inserção



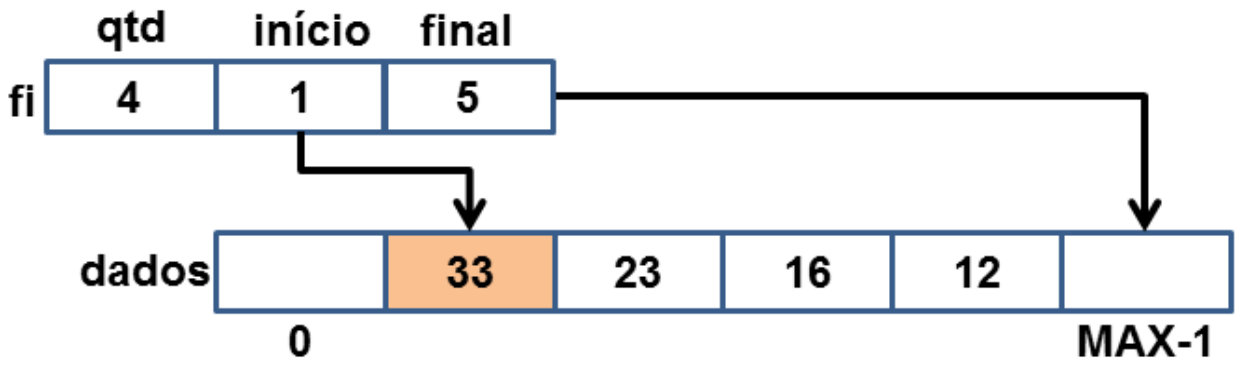
Fila Estática | Remoção

- Tarefa semelhante a remoção do final de uma Lista Sequencial Estática

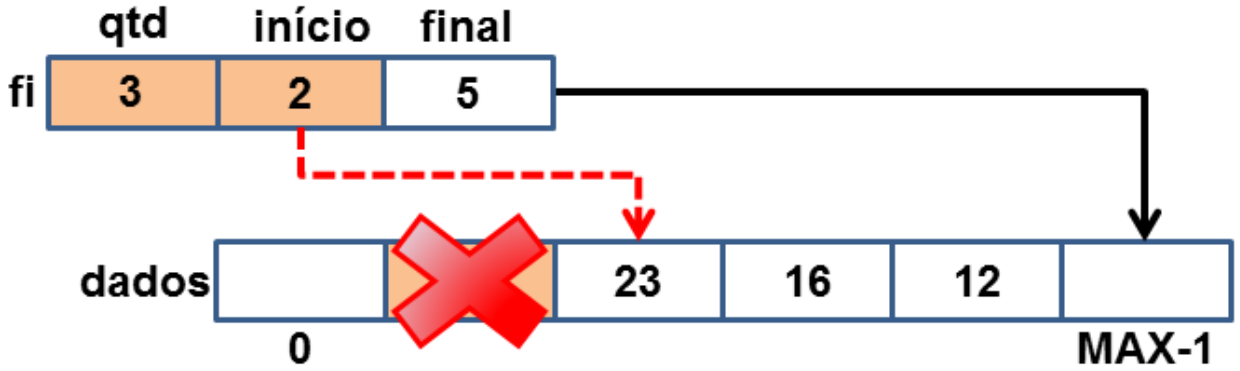
- Precisamos verificar
  - se a fila existe
  - se a fila não está vazia
- E só depois
  - Mover o início
  - Diminuir a quantidade

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

Fila Estática | Remoção



```
fi->inicio = (fi->inicio+1)%MAX;
fi->qtd--;
```



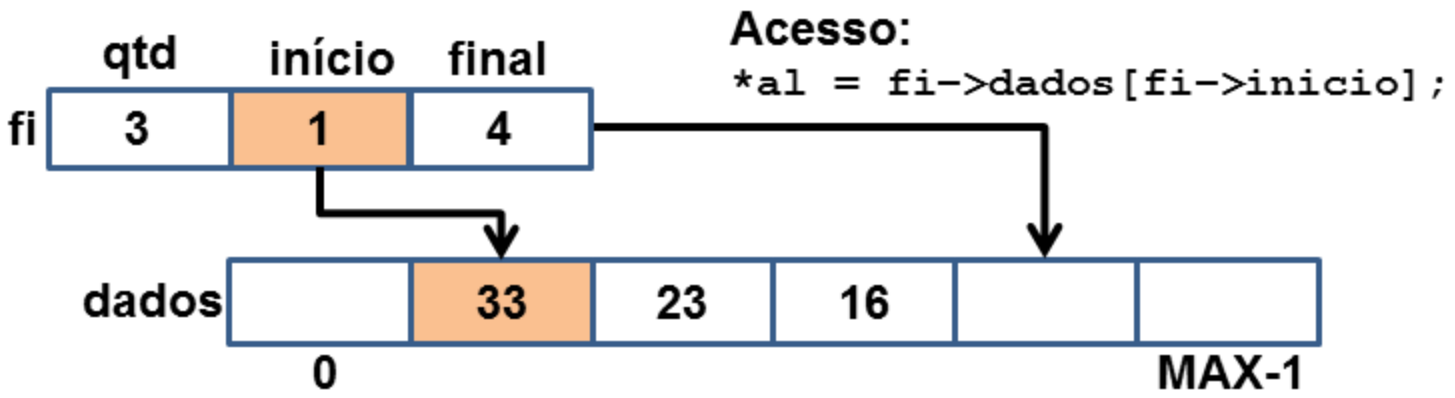
Fila Estática | Acesso

- Podemos acessar apenas o elemento no início da fila

- Precisamos verificar
  - se a fila existe
  - se a fila não está vazia
- E só depois
  - Acessar os dados do início

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

Fila Estática | Acesso



Fila Dinâmica | TAD

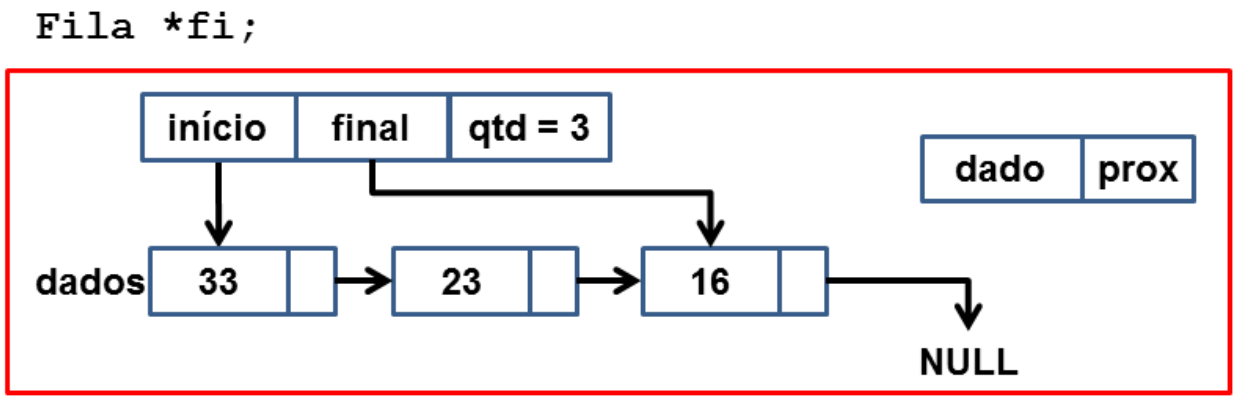
- Cada elemento é alocado e liberado conforme a necessidade
  - Vantagem: melhor uso dos recursos de memória
  - Desvantagem: necessidade de percorrer toda a fila para destruí-la

```c
//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;
};
typedef struct fila Fila;
```

Fila Dinâmica | TAD

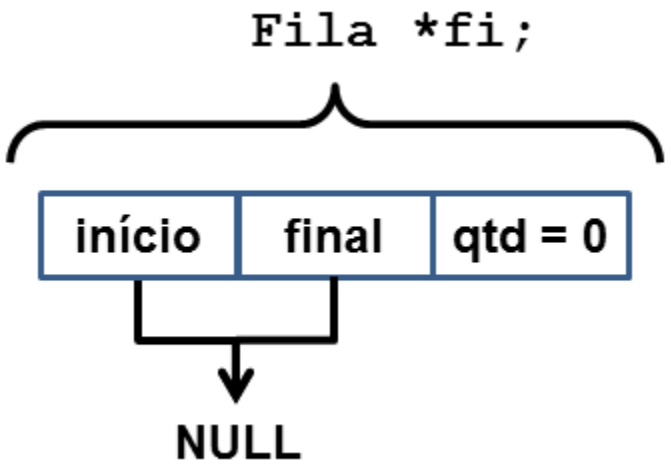
- Utiliza um nó descritor para guardar
  - início
  - final
  - quantidade de elementos na fila



Fila Dinâmica | Criação

- Aloca uma área de memória para a estrutura fila
- Inicializa os ponteiro para NULL e quantidade com zero (fila vazia)

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



Fila Dinâmica | Destruindo

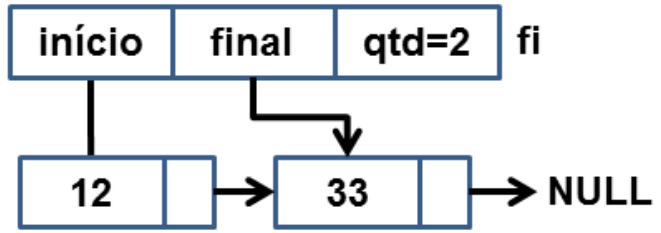
- Para liberar uma fila dinâmica é preciso percorrer toda a fila liberando a memória alocada para cada elemento inserido

- Ao final, liberamos a memória da fila em si

```
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);
    }
}
```

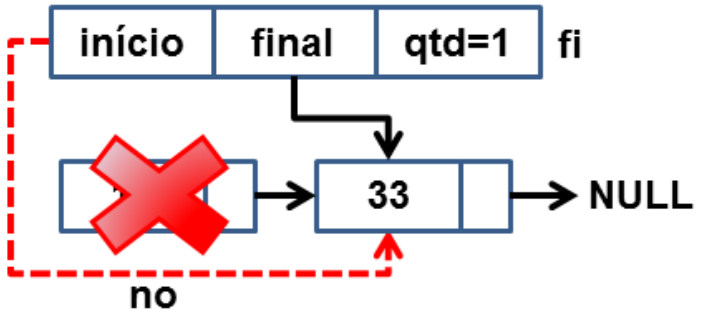
Fila Dinâmica | Destruindo

Fila inicial



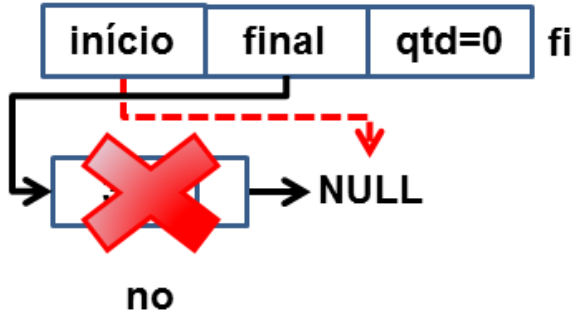
Passo 1:

```
no = fi->inicio;
fi->inicio = fi->inicio->prox;
free(no);
```



Passo 2:

```
no = fi->inicio;
fi->inicio = fi->inicio->prox;
free(no);
```

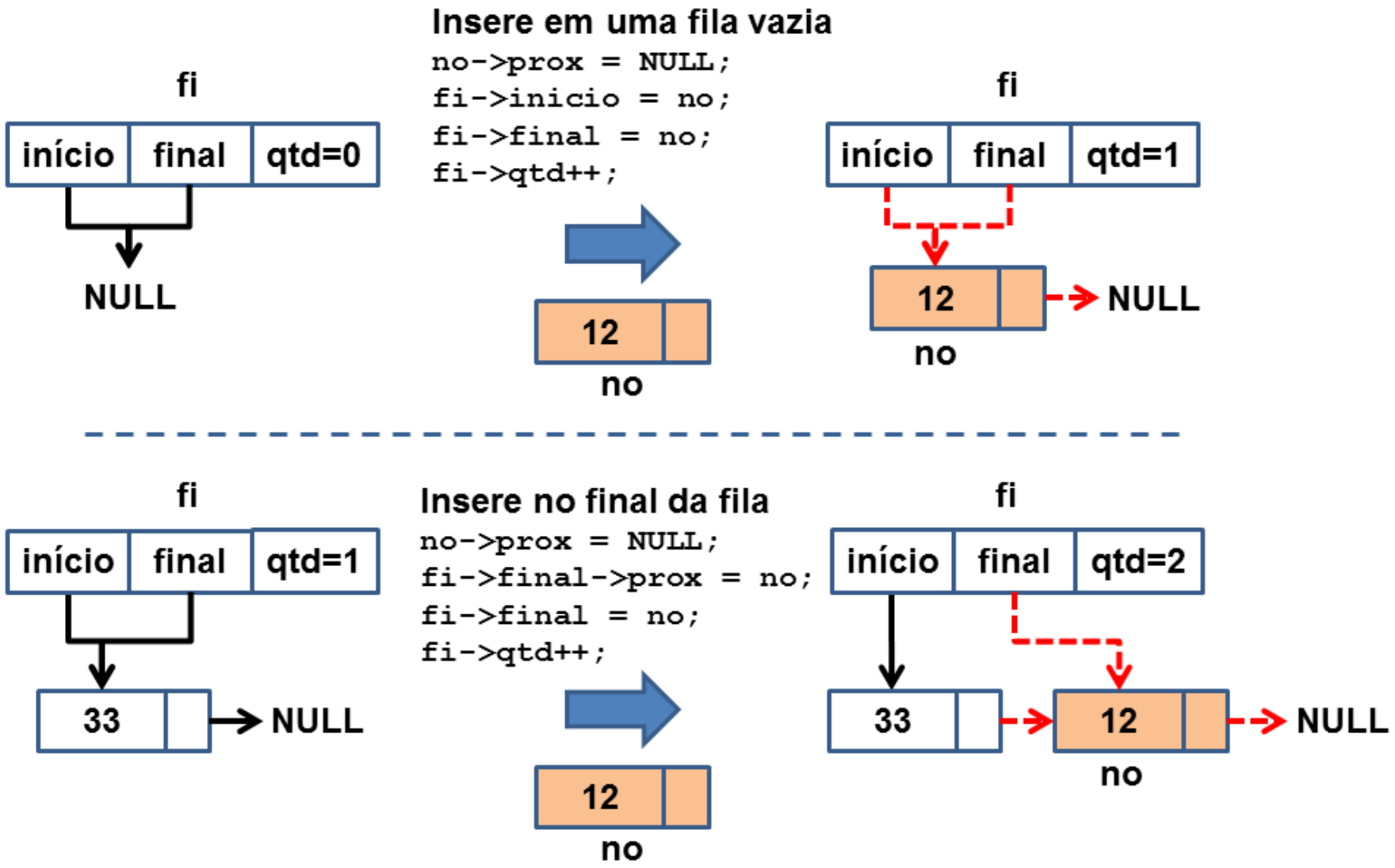


Fila Dinâmica | Inserção

- Tarefa simples, envolve alocar espaço para o novo elemento e ajustar alguns ponteiros
- Primeiro, verificamos se a fila existe
- Em seguida
  - alocar memória para o novo nó
  - copiar os dados
  - ajustar ponteiros
  - incrementar a quantidade

```
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;
}
```

Fila Dinâmica | Inserção



Fila Dinâmica | Remoção

- Envolve liberar a memória do elemento removido e ajustar alguns ponteiros

- Primeiro, verificamos se
  - a fila existe
  - a fila possui elementos

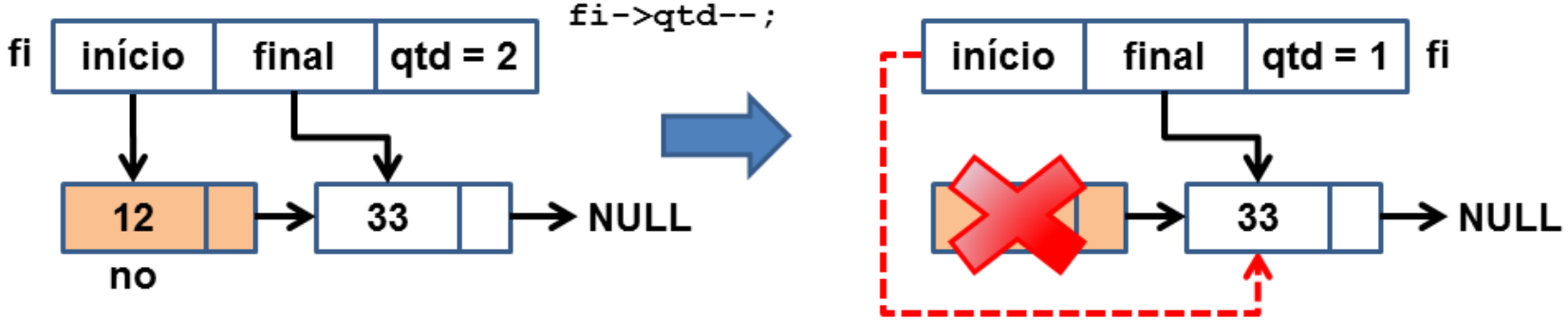
- E só depois
  - ajustar ponteiros
  - liberar a memória do nó
  - diminuir a quantidade

```
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;
}
```

Fila Dinâmica | Remoção

Remove do final da fila:

```
fi->inicio = fi->inicio->prox;
free(no);
fi->qtd--;
```



Remove e a fila fica vazia:

```
fi->inicio = fi->inicio->prox;
fi->final = NULL;
free(no);
fi->qtd--;
```

Fila Dinâmica | Acesso

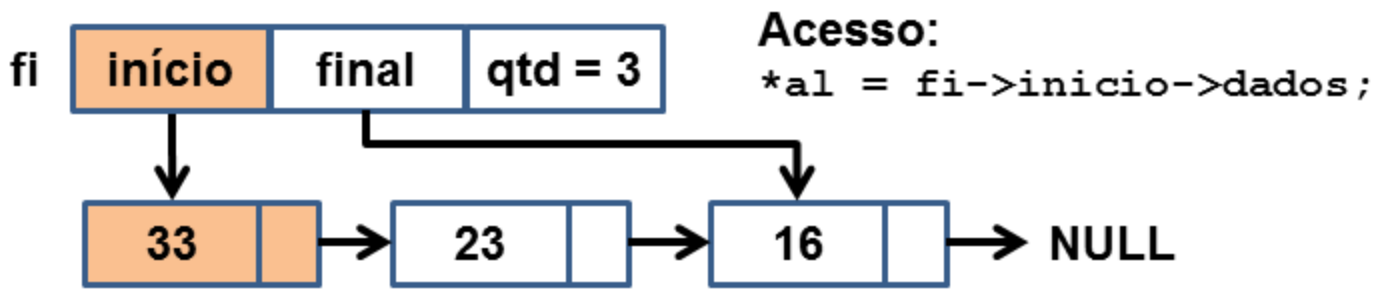
- Podemos acessar apenas o elemento no início da fila

- Precisamos verificar
  - se a fila existe
  - se a fila não está vazia

- E só depois
  - Acessar os dados do início

```
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;
}
```

Fila Dinâmica | Acesso



PILHAS

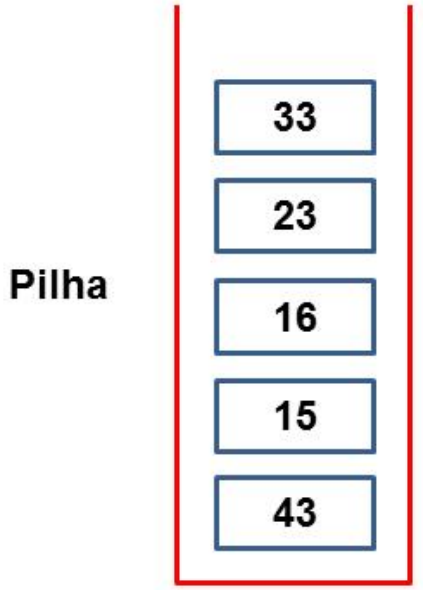
Pilha | Definição

- Conceito muito comum para as pessoas
  - Um conjunto finito de itens dispostos uns sobre os outros
  - Exemplo: a pilha de pratos esperando para ser lavada na pia

- Na computação, uma pilha é uma estrutura de dados linear utilizada para armazenar e organizar dados em um computador
  - Somente podemos inserir um novo item se colocarmos ele acima dos demais e somente poderemos remover o item que está no topo da pilha
  - Exemplo: desfazer/refazer em editores de texto ou softwares

Pilha | Definição

- As pilhas são implementadas e se comportam parecido com as listas
- Apenas obedecem uma ordem de entrada e saída
  - A inserção e remoção são realizadas sempre na mesma extremidade
  - São estruturas do tipo **LIFO** (*Last In First Out - último a entrar, primeiro a sair*)



Pilha | Definição

- Existem duas implementações principais para uma pilha
- Pilha estática
  - Os elementos são armazenados de forma consecutiva na memória (array)
  - É necessário definir o número máximo de elementos que a pilha irá possuir

- Pilha dinâmica
  - O espaço de memória é alocado em tempo de execução
  - A pilha cresce e diminui com o tempo
  - Cada elemento da pilha armazena o endereço de memória do próximo

Pilha | Aplicações

- Chamadas de função (recursão)
- Undo/Redo em editores de texto
- Navegação em navegadores
  - páginas anteriores e próximas
- Avaliação de expressões matemáticas:
  - avaliar expressões matemáticas em notação pós-fixa
- Algoritmos de busca em profundidade
- Conversão entre diferentes notações
  - conversão de expressões matemáticas, como de infixa para pós-fixa

Pilha Estática | TAD

- Utiliza um array para armazenar os elementos
  - Vantagem: fácil de criar e destruir
  - Desvantagem: necessidade de definir previamente o tamanho da fila

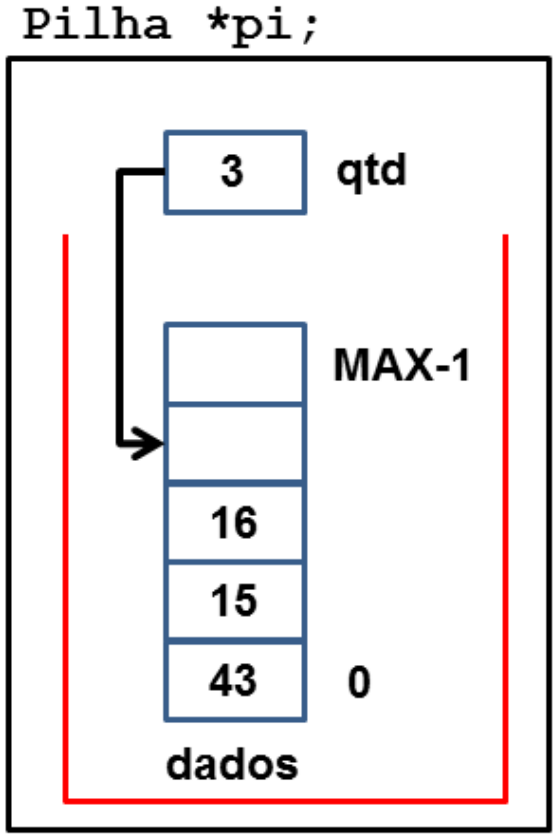
```c
//Definição do tipo Pilha
#define MAX 100

struct pilha{
    int qtd;
    struct aluno dados[MAX];
};

typedef struct pilha Pilha;
```

Pilha Estática | TAD

- Equivale a uma Lista Sequencial Estática
  - Diferente da lista, a pilha permite apenas um único tipo de inserção e remoção



Pilha Estática | Criação e liberação

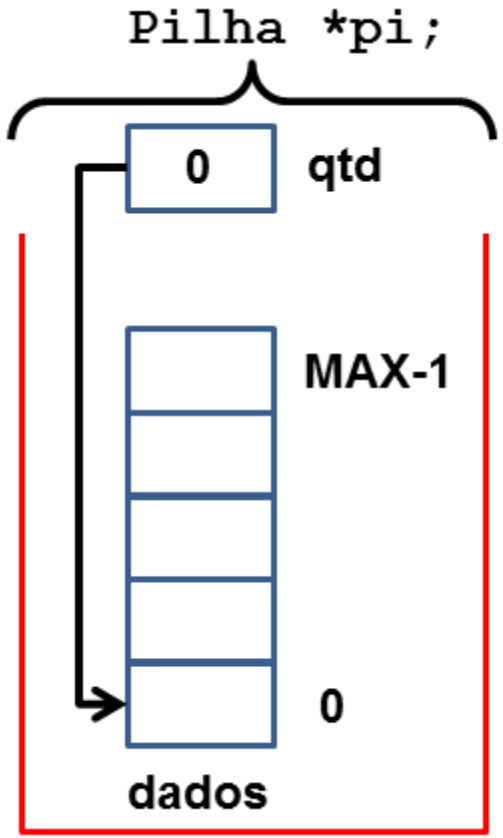
- Criação
  - Aloca uma área de memória para a pilha
  - Corresponde a memória necessária para armazenar a estrutura da fila
  - Define a quantidade como zero

- Liberação
  - Basta liberar a memória alocada para a estrutura pilha

```
Pilha* cria_Pilha(){
    Pilha *pi;
    pi = (Pilha*) malloc(sizeof(struct pilha));
    if(pi != NULL)
        pi->qtd = 0;
    return pi;
}

void libera_Pilha(Pilha* pi){
    free(pi);
}
```

Pilha Estática | Criação



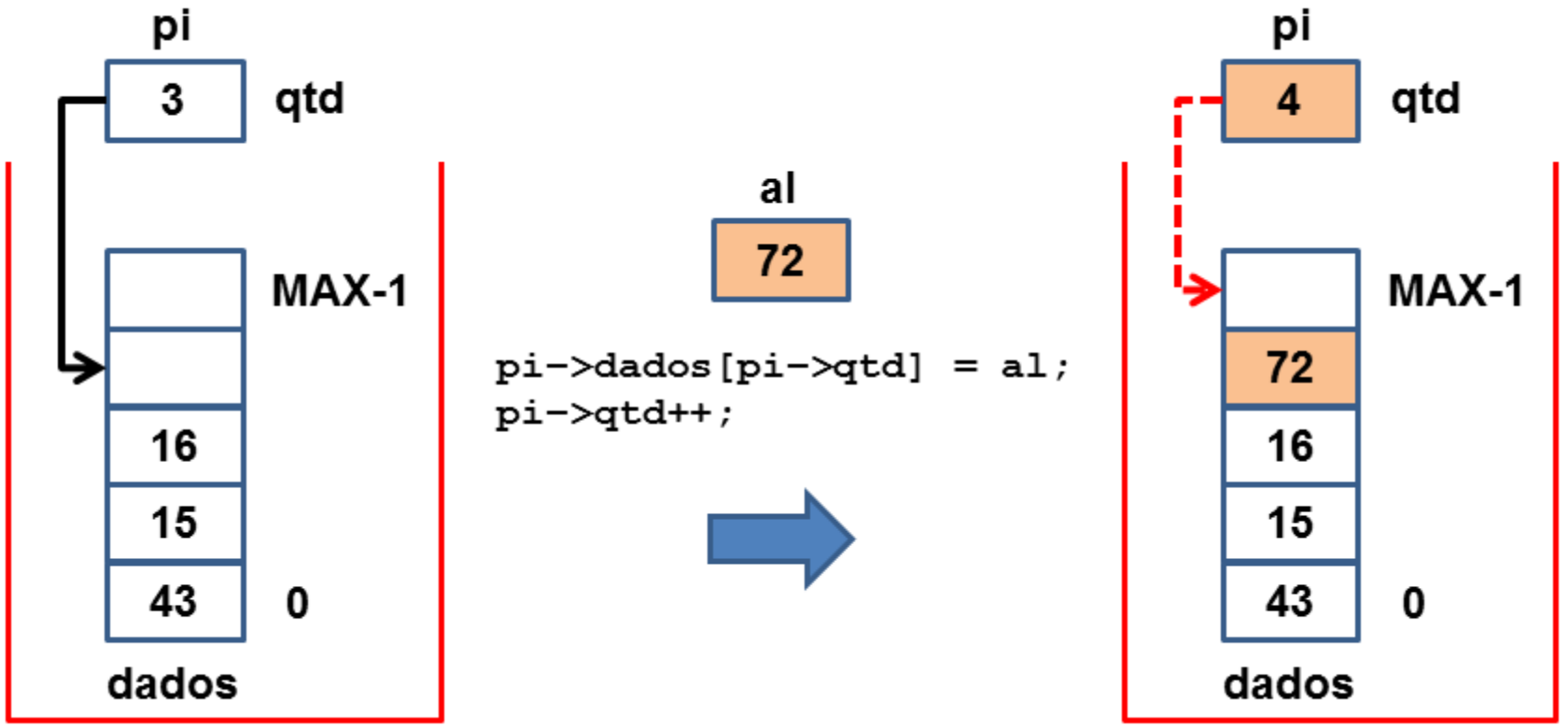
Pilha Estática | Inserção

- Tarefa semelhante a inserção no final de uma Lista Sequencial Estática

- Precisamos verificar
  - se a pilha existe
  - se a pilha está cheia
- E só depois
  - copiar os dados
  - incrementar a quantidade

```
int insere_Pilha(Pilha* pi, struct aluno al){
    if(pi == NULL)
        return 0;
    if(pi->qtd == MAX)//pilha cheia
        return 0;
    pi->dados[pi->qtd] = al;
    pi->qtd++;
    return 1;
}
```

Pilha Estática | Inserção



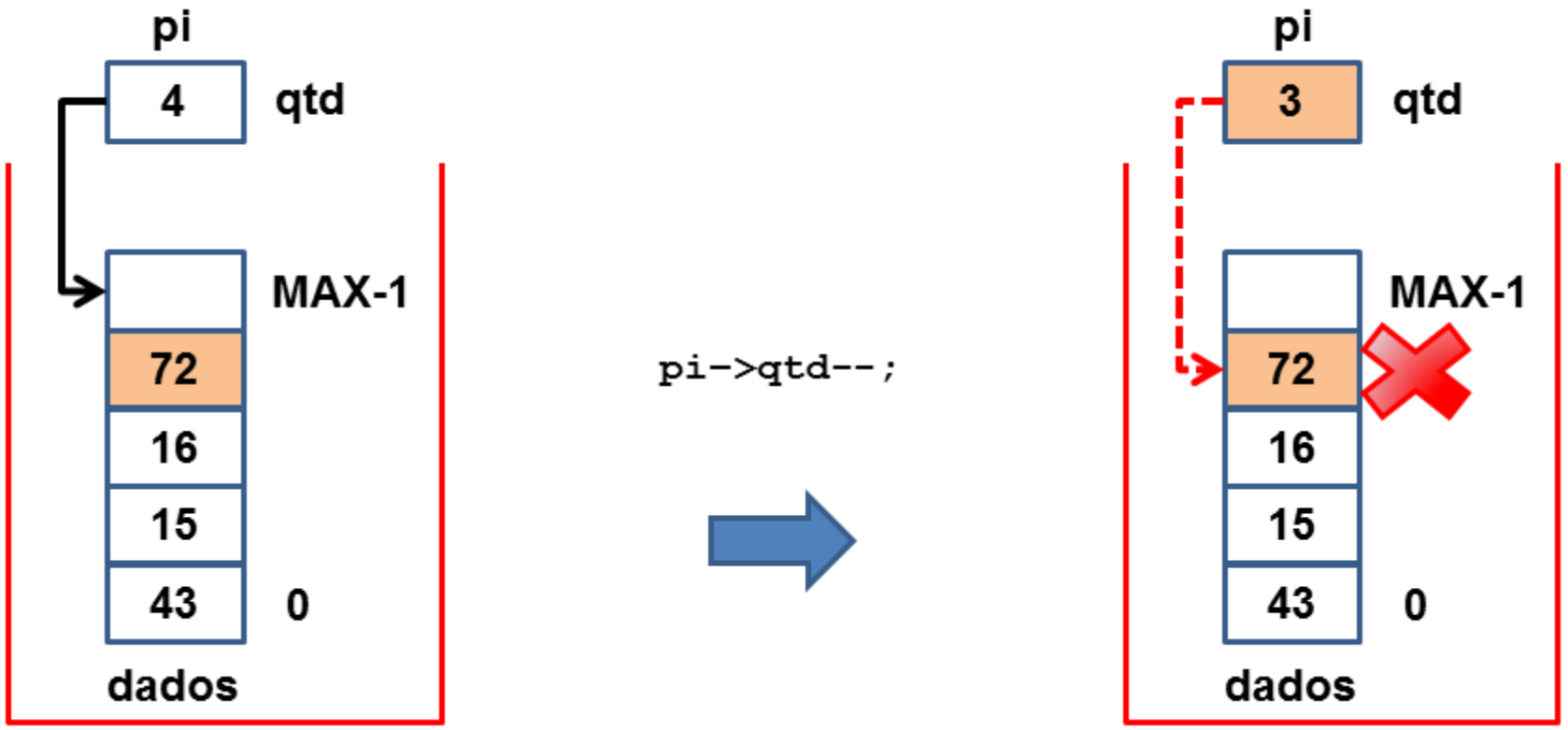
Pilha Estática | Remoção

- Tarefa semelhante a remoção do final de uma Lista Sequencial Estática

- Precisamos verificar
  - se a pilha existe
  - se a pilha não está vazia
- E só depois
  - Mover o início
  - Diminuir a quantidade

```
int remove_Pilha(Pilha* pi){
    if(pi == NULL || pi->qtd == 0)
        return 0;
    pi->qtd--;
    return 1;
}
```

Pilha Estática | Remoção

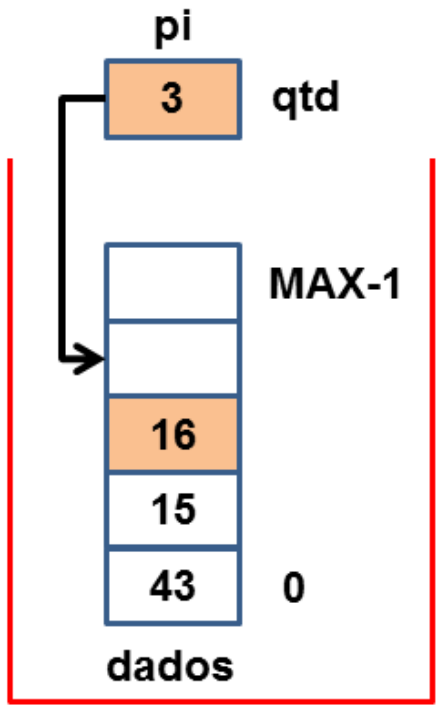


Pilha Estática | Acesso

- Podemos acessar apenas o elemento no topo da pilha

- Precisamos verificar
  - se a pilha existe
  - se a pilha não está vazia
- E só depois
  - Acessar os dados do topo

```
int consulta_topo_Pilha(Pilha* pi, struct aluno *al){
    if(pi == NULL || pi->qtd == 0)
        return 0;
    *al = pi->dados[pi->qtd-1];
    return 1;
}
```



Acesso :

```
*al = pi->dados[pi->qtd -1];
```

Pilha Dinâmica | TAD

- Cada elemento é alocado e liberado conforme a necessidade
  - Vantagem: melhor uso dos recursos de memória
  - Desvantagem: necessidade de percorrer toda a pilha para destruí-la

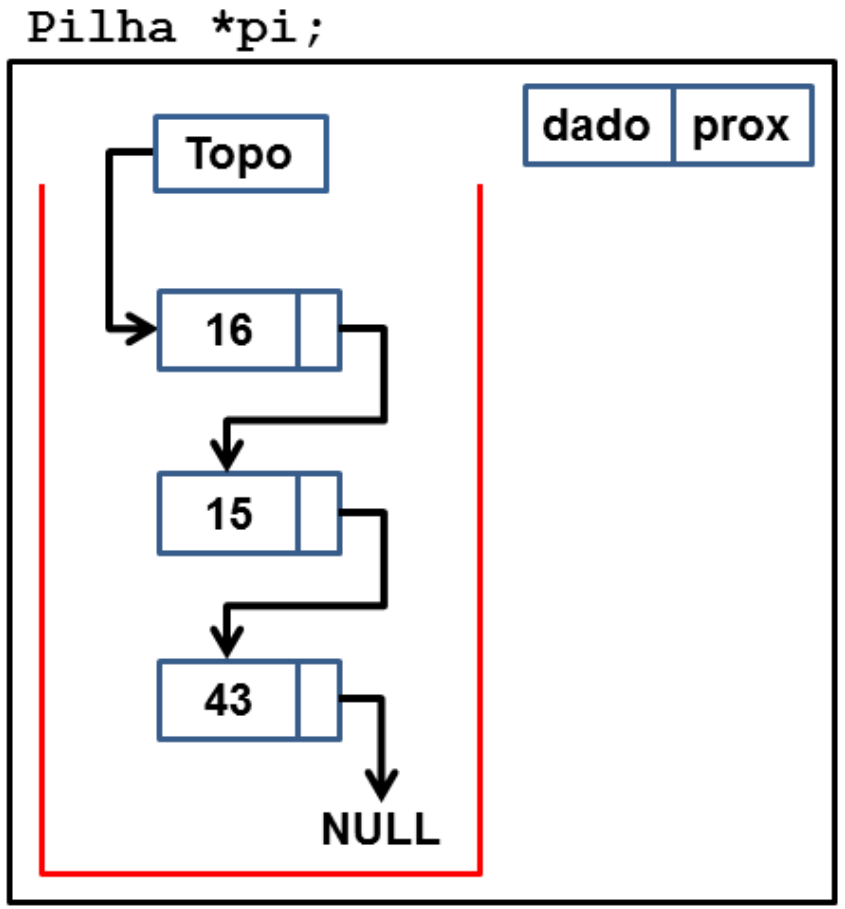
- Utiliza um ponteiro para ponteiro
  - permite inserção no início sem usar nó descritor
  - último elemento da pilha aponta para NULL

```
//Definição do tipo Pilha
struct elemento{
    struct aluno dados;
    struct elemento *prox;
};

typedef struct elemento Elem;
typedef struct elemento* Pilha;
```

Pilha Dinâmica | TAD

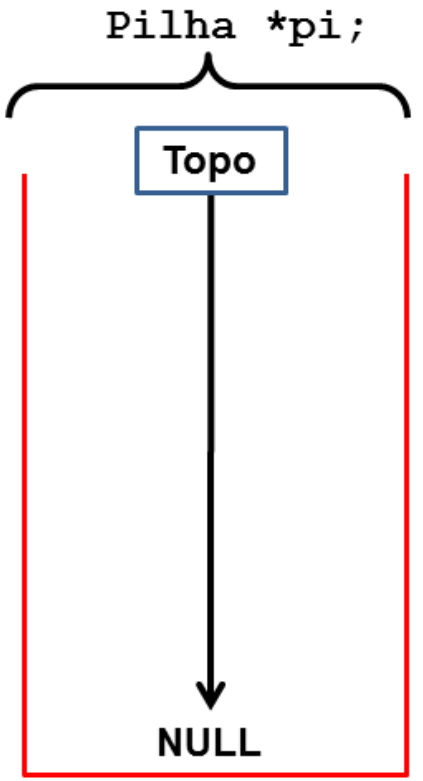
- Equivale a uma Lista Sequencial Estática
  - Diferente da lista, a pilha permite apenas um único tipo de inserção e remoção (no início)



Pilha Dinâmica | Criação

- Faz a alocação de uma área de memória para armazenar o endereço do início da pilha
  - Equivale a criar uma pilha vazia

```
Pilha* cria_Pilha(){
    Pilha* pi = (Pilha*) malloc(sizeof(Pilha));
    if(pi != NULL)
        *pi = NULL;
    return pi;
}
```



Pilha Dinâmica | Destruindo

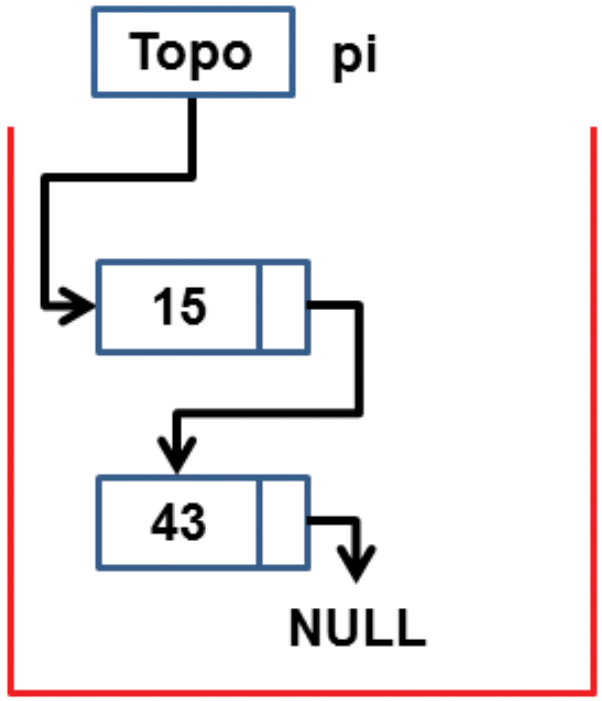
- Para liberar uma pilha dinâmica é preciso percorrer toda a pilha liberando a memória alocada para cada elemento inserido

- Ao final, liberamos a memória da pilha em si

```
void libera_Pilha(Pilha* pi){
    if(pi != NULL){
        Elem* no;
        while((*pi) != NULL){
            no = *pi;
            *pi = (*pi)->prox;
            free(no);
        }
        free(pi);
    }
}
```

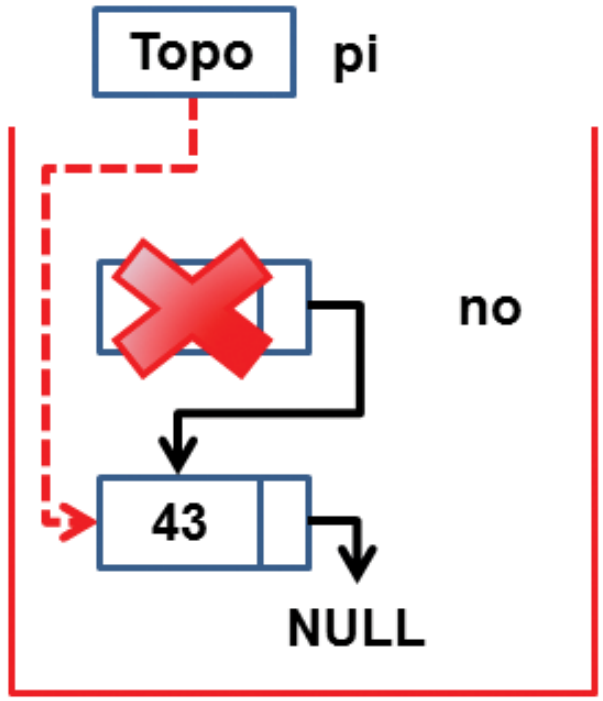
Pilha Dinâmica | Destruindo

Pilha Inicial



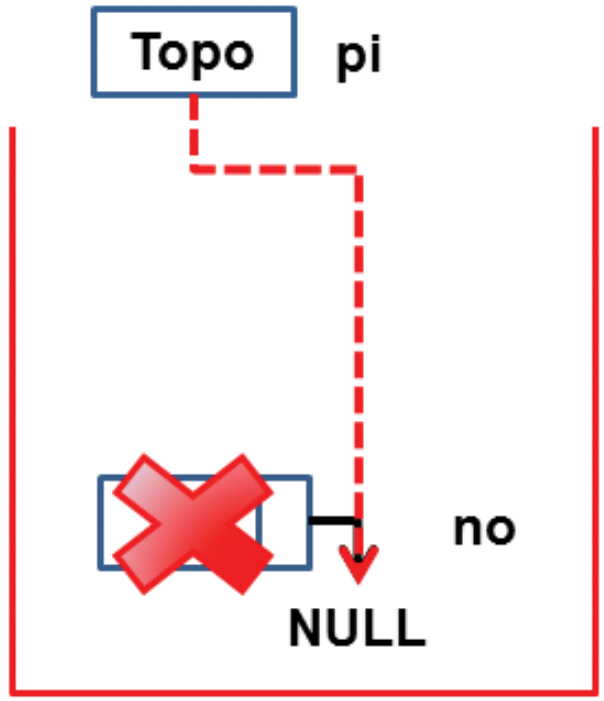
Passo 1:

```
no = *pi;
*pi = (*pi)->prox;
free(no);
```



Passo 2:

```
no = *pi;
*pi = (*pi)->prox;
free(no);
```



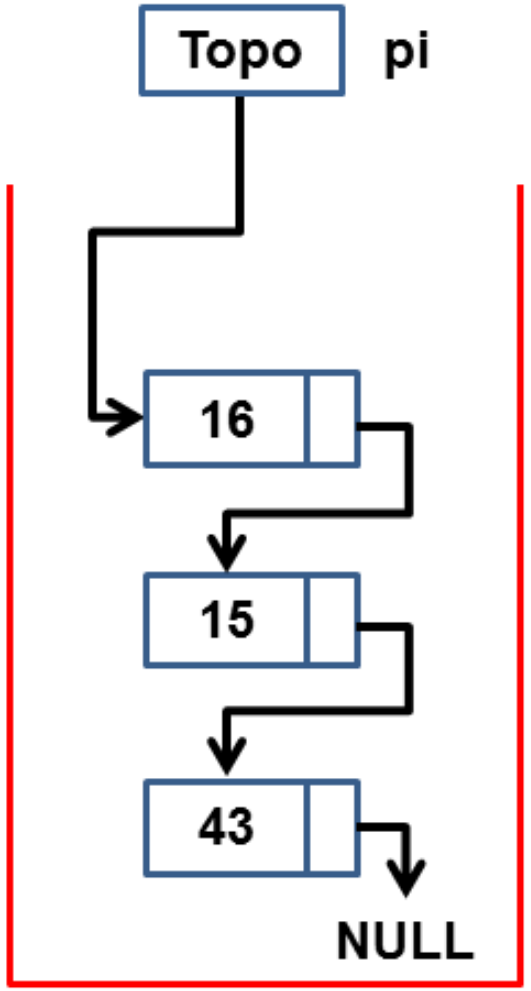
Pilha Dinâmica | Inserção

- Tarefa simples, envolve alocar espaço para o novo elemento e ajustar alguns ponteiros

- Primeiro, verificamos se a pilha existe
- Em seguida
  - alocar memória para o novo nó
  - copiar os dados
  - mudar o início

```
int insere_Pilha(Pilha* pi, struct aluno al){
    if(pi == NULL)
        return 0;
    Elem* no;
    no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL)
        return 0;
    no->dados = al;
    no->prox = (*pi);
    *pi = no;
    return 1;
}
```

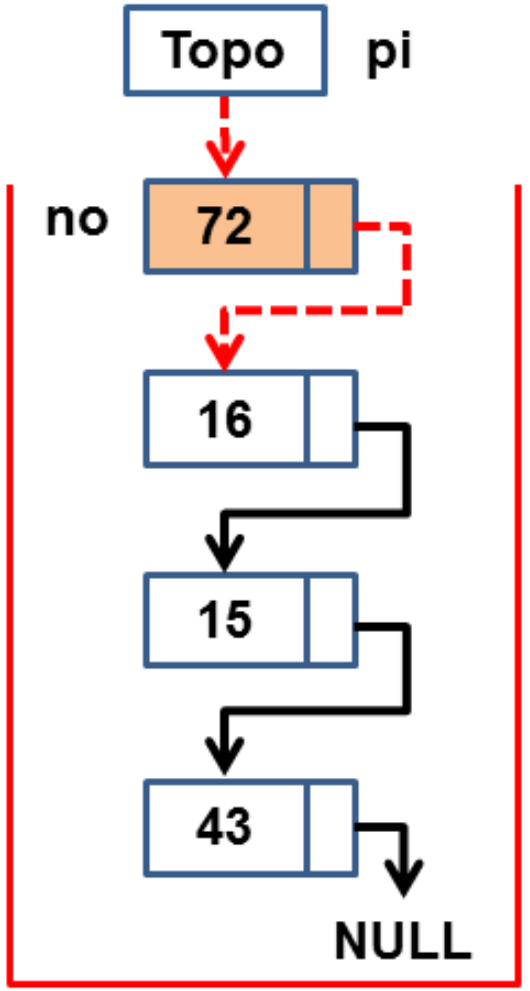
Pilha Dinâmica | Inserção



al

72

```
no->dados = al;
no->prox = (*pi);
*pi = no;
```



Pilha Dinâmica | Remoção

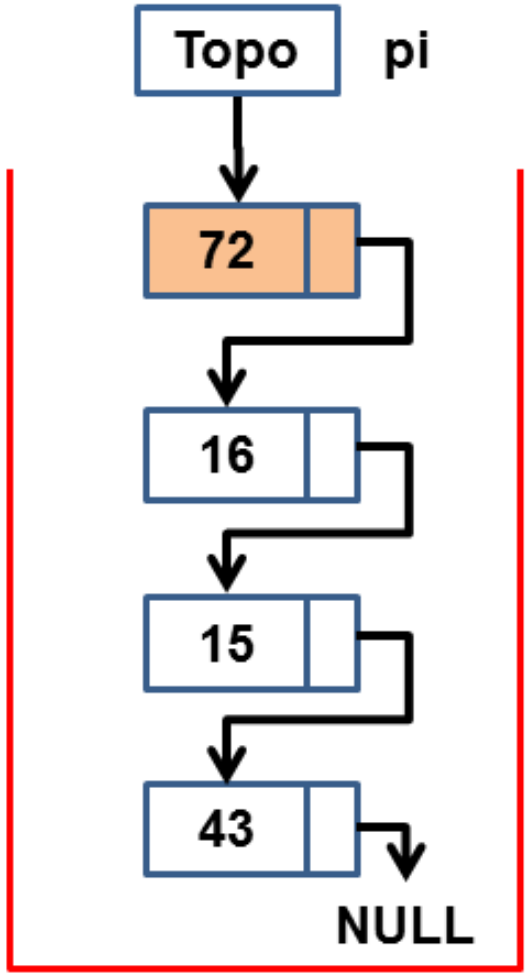
- Envolve liberar a memória do elemento removido e ajustar alguns ponteiros

- Primeiro, verificamos se
  - a pilha existe
  - a pilha possui elementos

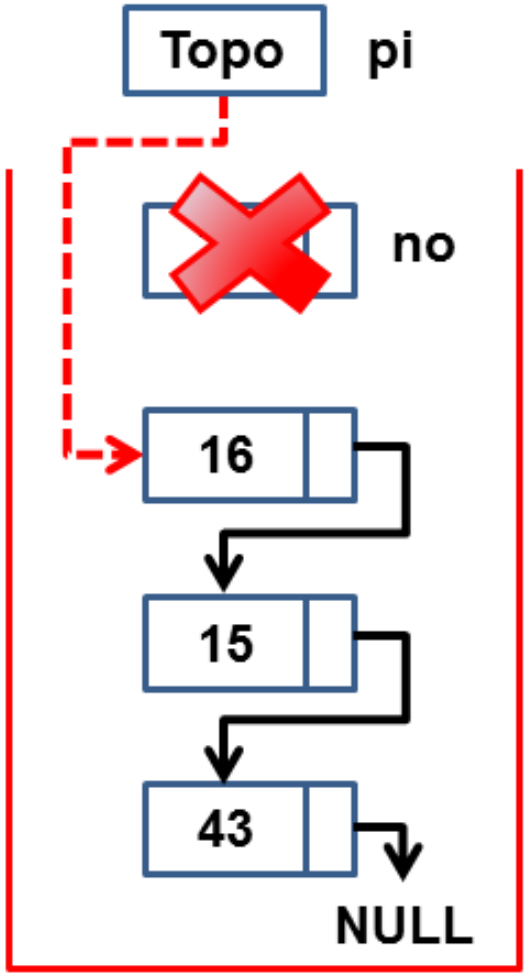
- E só depois
  - mudar o início da pilha
  - liberar a memória do nó

```
int remove_Pilha(Pilha* pi){
    if(pi == NULL)
        return 0;
    if((*pi) == NULL)
        return 0;
    Elem *no = *pi;
    *pi = no->prox;
    free(no);
    return 1;
}
```

Pilha Dinâmica | Remoção



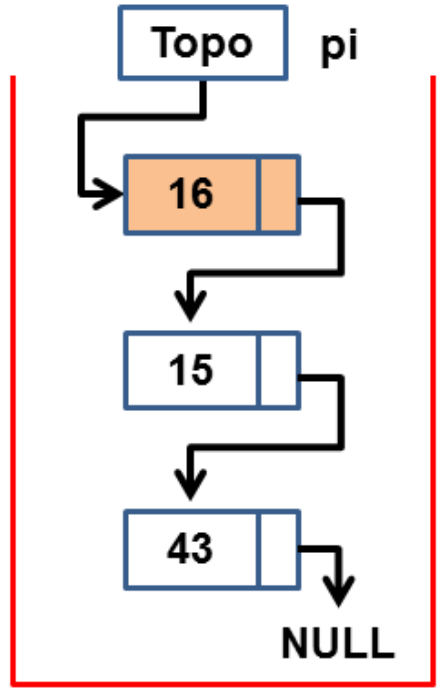

```
Elem *no = *pi;
*pi = no->prox;
free(no);
```



Pilha Dinâmica | Acesso

- Podemos acessar apenas o elemento no topo da pilha
- Precisamos verificar
  - se a pilha existe
  - se a pilha não está vazia
- E só depois
  - Acessar os dados do topo

```
int consulta_topo_Pilha(Pilha* pi, struct aluno *al){
    if(pi == NULL)
        return 0;
    if((*pi) == NULL)
        return 0;
    *al = (*pi)->dados;
    return 1;
}
```



Acesso:

```
*al = (*pi)->dados;
```

Material Complementar | Vídeo Aulas

- Aula 31 - Fila - Definição
  - http://youtu.be/aEfOzz_KXl8
- Aula 32 - Criando e Destruindo uma Fila Estática
  - https://www.youtube.com/watch?v=y93DzmBskGQ
- Aula 33 - Informações da Fila Estática
  - http://youtu.be/RLu9QLd_xpY
- Aula 34 - Inserção e Remoção na Fila Estática
  - http://youtu.be/0KXFoxSCEJE
- Aula 35 - Criando e Destruindo uma Fila Dinâmica
  - http://youtu.be/4YXnrKJCWrE
- Aula 36 - Informações da Fila Dinâmica
  - http://youtu.be/aIFK1n9Sp30
- Aula 37 - Inserção e Remoção na Fila Dinâmica
  - http://youtu.be/yOjgEXbKtME

programação descomplicada

Material Complementar | Vídeo Aulas

- Aula 38 - Pilha - Definição
  - http://youtu.be/2RCrd7gOUMM
- Aula 39 - Criando e Destruindo uma Pilha Estática
  - http://youtu.be/OIQJL-K2SUI
- Aula 40 - Informações da Pilha Estática
  - http://youtu.be/X_-paXn4At8
- Aula 41 - Inserção e Remoção na Pilha Estática
  - http://youtu.be/n7fKiYoCvUo
- Aula 42 - Criando e Destruindo uma Pilha Dinâmica
  - http://youtu.be/9GGJH2sjOac
- Aula 43 - Informações da Pilha Dinâmica
  - http://youtu.be/RPf6DgjS3OM
- Aula 44 - Inserção e Remoção na Pilha Dinâmica
  - http://youtu.be/06sqFaB3gU0

programação descomplicada

Material Complementar | GitHub

- https://github.com/arbackes