

Estruturas lineares: filas e pilhas

Profa. Mirtha Lina Fernández Venero

Prof. Paulo Henrique Pisani

<http://professor.ufabc.edu.br/~paulo.pisani/>

Agenda

- Pilhas:
 - Pilha estática;
 - Pilha dinâmica (com lista simplesmente ligada).
- Filas:
 - Fila estática;
 - Fila estática circular;
 - Fila dinâmica (com lista simplesmente ligada).
- Outros tipos: deque, pilhas múltiplas;
- Exercícios.

Pilha

Pilha estática

- **Pilha (*stack*):** estrutura de dados que adota a estratégia **LIFO** (*Last In First Out*): último a entrar é o primeiro a sair;
- Operações básicas:
 - Empilhar / *Push* (inserção);
 - Desempilhar / *Pop* (remoção).

Pilha estática

Pilha estática

- **Pilha estática**: implementa a estrutura de dados utilizando um **arranjo**;
- Portanto, os itens são armazenados em posições consecutivas na memória.

Pilha estática

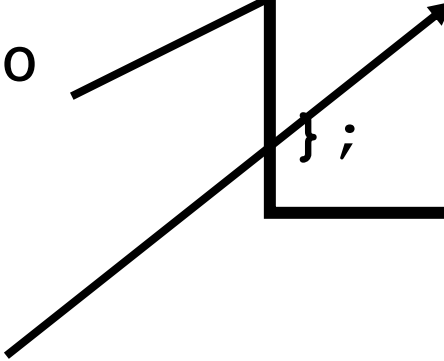
- Estrutura básica:

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```


Ponteiro para o
vetor alocado



Tamanho do
vetor alocado



Índices do elemento no
topo da pilha



Pilha estática

- Inicialização

(topo == -1) ==> Pilha vazia!

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=-1
```



0

1

2

3

Pilha estática

- Como empilhar?
 - **topo==tamanho-1** ?
 - Sim: pilha cheia!
 - Não: então podemos empilhar.

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=-1
```



0

1

2

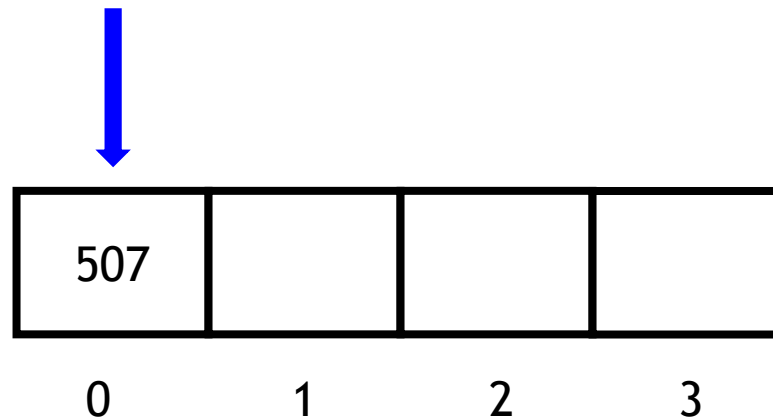
3

Pilha estática

- Empilhar 507:
 - Verificar se pilha está cheia
 - `topo++`
 - `itens[topo]=507`

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=0
```

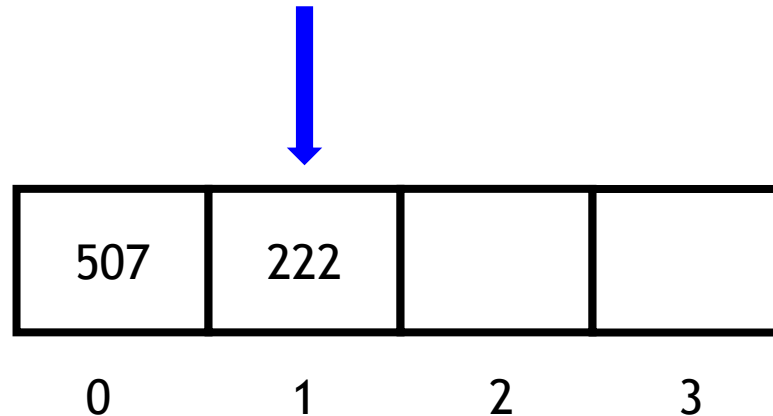


Pilha estática

- Empilhar 222:
 - Verificar se pilha está cheia
 - `topo++`
 - `itens[topo]=222`

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=1
```

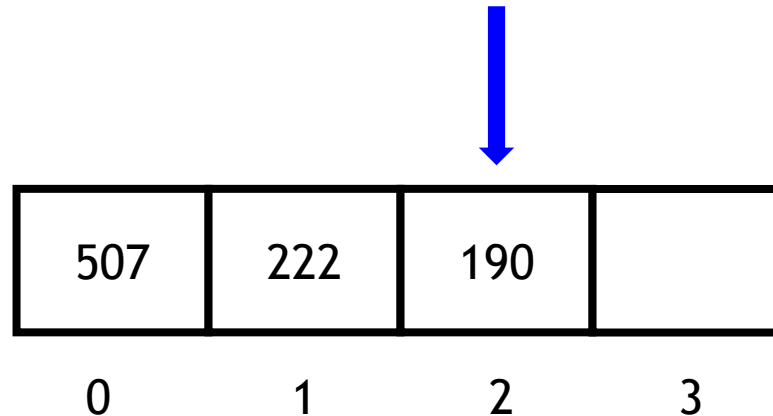


Pilha estática

- Empilhar 190:
 - Verificar se pilha está cheia
 - `topo++`
 - `itens[topo]=190`

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=2
```

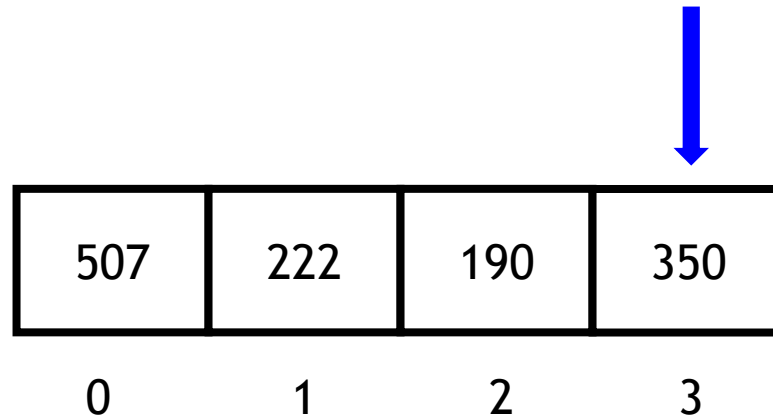


Pilha estática

- Empilhar 350:
 - Verificar se pilha está cheia
 - `topo++`
 - `itens[topo]=350`

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=3
```

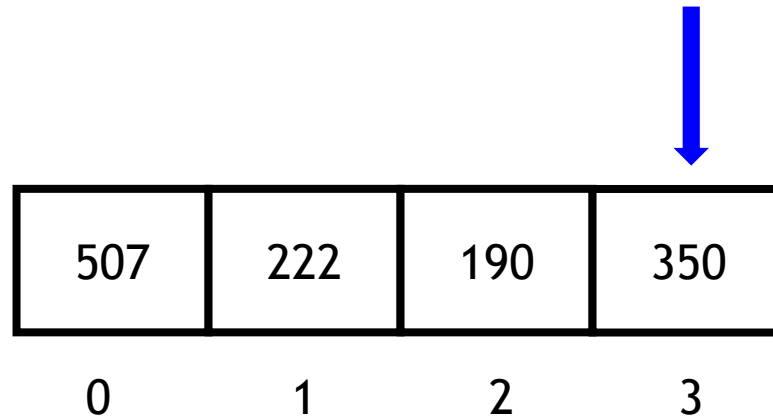


Pilha estática

- Empilhar 500:
 - **Pilha está cheia!**
 - **topo==tamanho-1**

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=3
```



Pilha estática

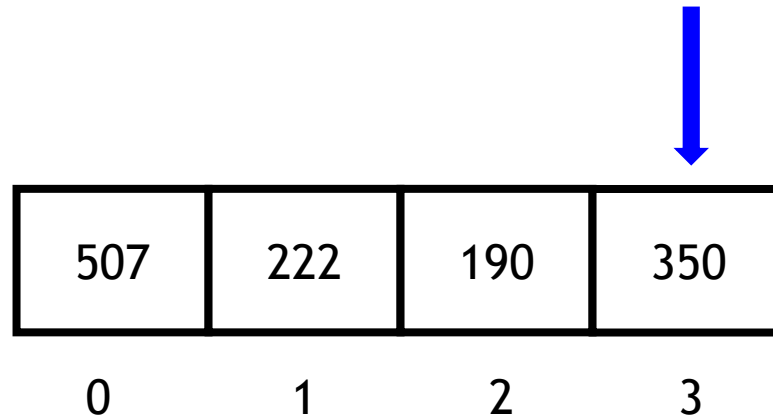
- Como desempilhar?

- **topo != -1?**

- Sim: podemos desempilhar
 - Não: pilha vazia!

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=3
```

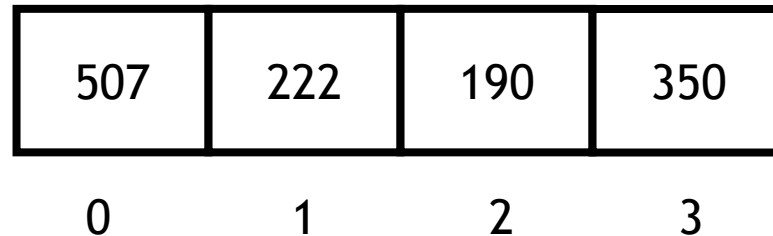


Pilha estática

- Desempilhar
 - Verificar se pilha está vazia
 - Salvar itens[topo]
 - Topo--
 - Retonar item salvo

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=3
```



Pilha estática

- Desempilhar

Não • Verificar se pilha está vazia

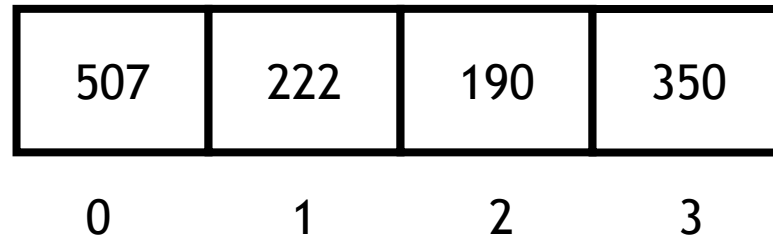
350 • Salvar itens[topo]

- Topo--

- Retonar item salvo

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=3
```



Pilha estática

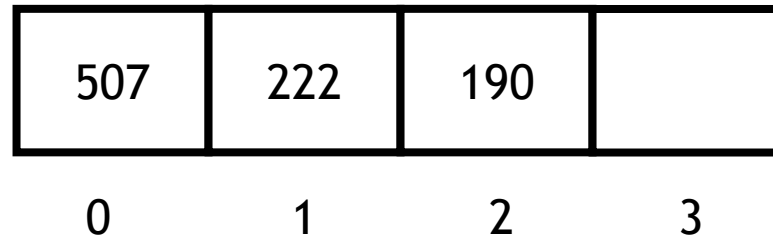
- Desempilhar
 - Verificar se pilha está vazia
 - Salvar itens[topo]
 - Topo--
 - Retonar item salvo

2

350

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=2
```



Pilha estática

- Desempilhar

Não • Verificar se pilha está vazia

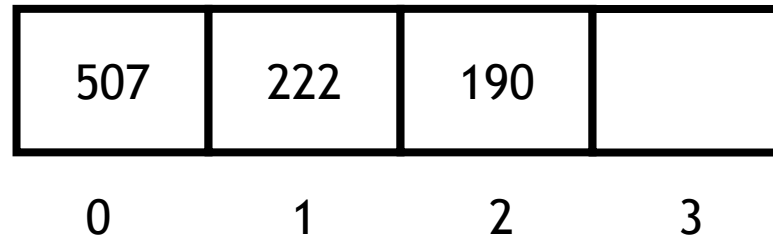
190 • Salvar itens[topo]

- Topo--

- Retonar item salvo

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=2
```



Pilha estática

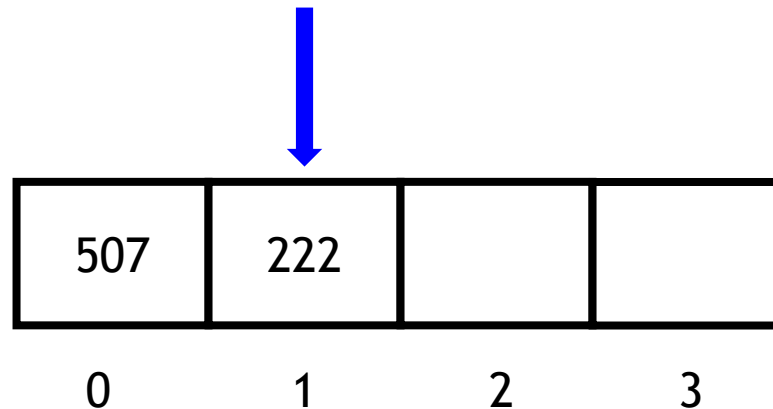
- Desempilhar
 - Verificar se pilha está vazia
 - Salvar itens[topo]
 - Topo--
 - Retonar item salvo

1

190

```
typedef struct Pilha Pilha;  
struct Pilha {  
    int* itens;  
    int tamanho;  
    int topo;  
};
```

```
itens=malloc(sizeof(int)*4)  
tamanho=4  
topo=1
```



Pilha dinâmica

Pilha dinâmica

- **Pilha dinâmica**: implementa a estrutura de dados utilizando uma **lista ligada**;
- Portanto, os itens são alocados em memória de acordo com a necessidade.

Pilha dinâmica

- Estrutura básica:

Ponteiro para o primeiro item da pilha

```
typedef struct Pilha Pilha;  
struct Pilha {  
    → LinkedNode* topo;  
};
```

```
typedef struct LinkedNode LinkedNode;  
struct LinkedNode {  
    int dados;  
    LinkedNode *next;  
};
```

Pilha dinâmica

- Inicialização

```
topo = NULL
```

(topo == NULL) ==> Pilha vazia!

Pilha dinâmica

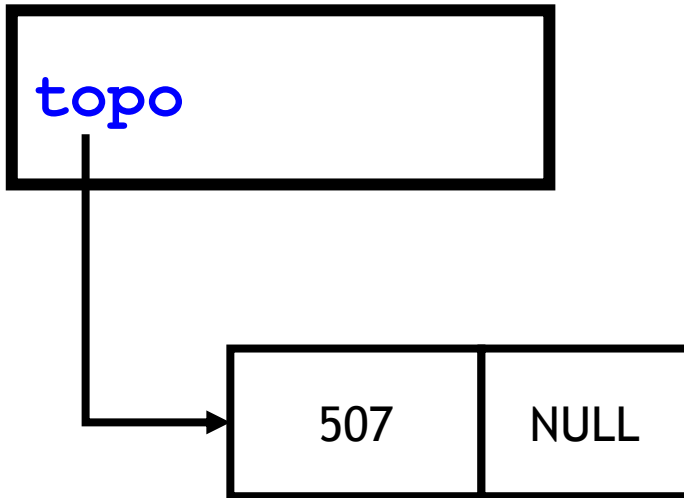
- Como empilhar?

```
topo = NULL
```

1. Alocar novo ListNode
2. Adicioná-lo logo após o item apontado por `topo`
3. Atualizar ponteiro `topo`

Pilha dinâmica

- Como empilhar?



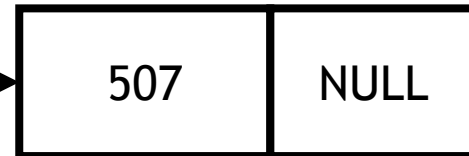
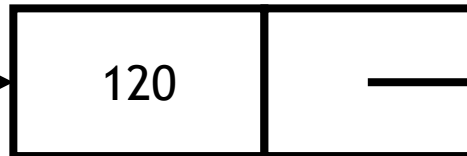
1. Alocar novo ListNode
2. Adicioná-lo logo após o item apontado por **topo**
3. Atualizar ponteiro **topo**

Pilha dinâmica

- Como empilhar?



1. Alocar novo `LinkedListNode`
2. Adicioná-lo **ANTES** o item apontado por `topo`
3. Atualizar ponteiro `topo`

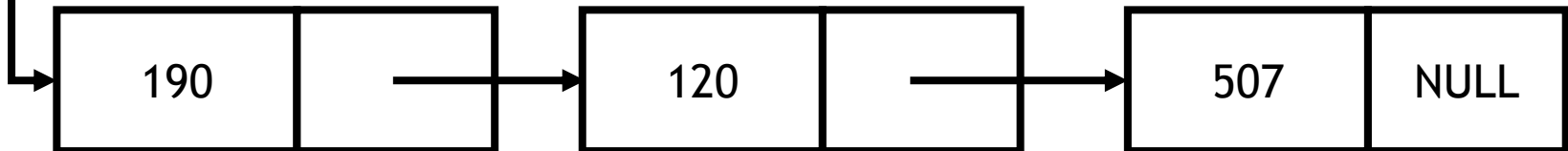


Pilha dinâmica

- Como empilhar?

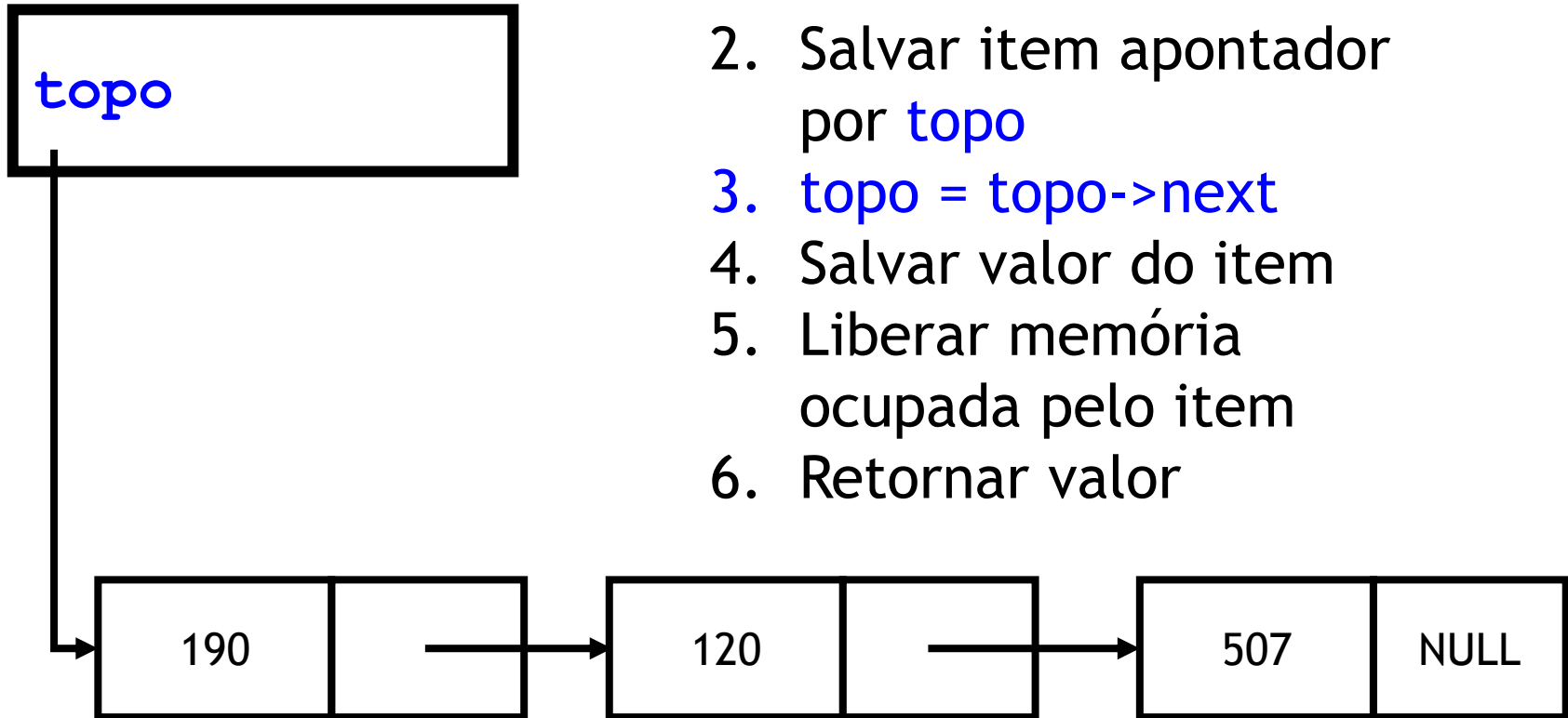


1. Alocar novo ListNode
2. Adicioná-lo **ANTES** o item apontado por **topo**
3. Atualizar ponteiro **topo**



Pilha dinâmica

- Como desempilhar?



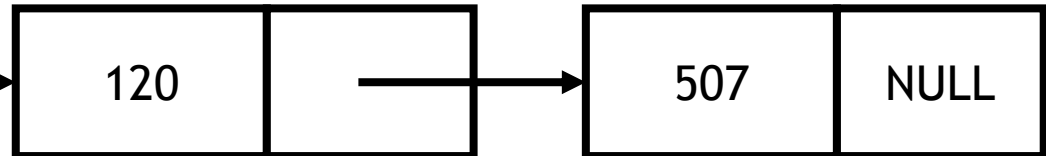
1. Verificar se pilha está vazia
2. Salvar item apontador por **topo**
3. **topo = topo->next**
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor

Pilha dinâmica

- Como desempilhar?



1. Verificar se pilha está vazia
2. Salvar item apontador por **topo**
3. **topo = topo->next**
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor



Pilha dinâmica

- Como desempilhar?



1. Verificar se pilha está vazia
2. Salvar item apontador por `topo`
3. `topo = topo->next`
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor

Pilha dinâmica

- Como desempilhar?

```
topo=NULL
```

1. Verificar se pilha está vazia
2. Salvar item apontador por `topo`
3. `topo = topo->next`
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor

Fila

Fila

- **Fila (*queue*)**: estrutura de dados que adota a estratégia **FIFO** (*First In First Out*): primeiro a entrar é o primeiro a sair;
- Operações básicas:
 - Enfileirar (inserção);
 - Desenfileirar (remoção).

Fila estática

Fila estática

- **Fila estática**: implementa a estrutura de dados utilizando um **arranjo**;
- Portanto, os itens são armazenados em posições consecutivas na memória.

Fila estática

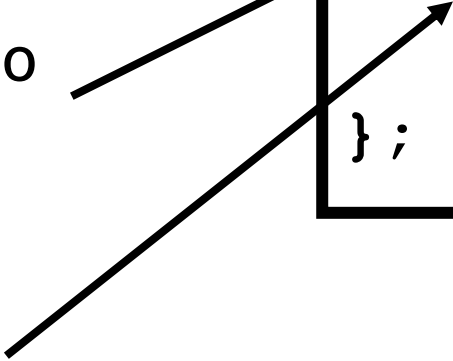
- Estrutura básica:

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```


Ponteiro para o
vetor alocado



Tamanho do
vetor alocado



Índices dos elementos
de início e fim da fila

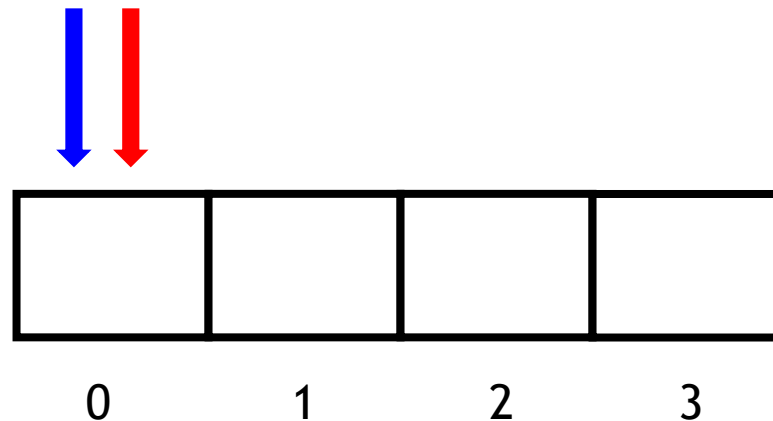


Fila estática

- Inicialização

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=0
```

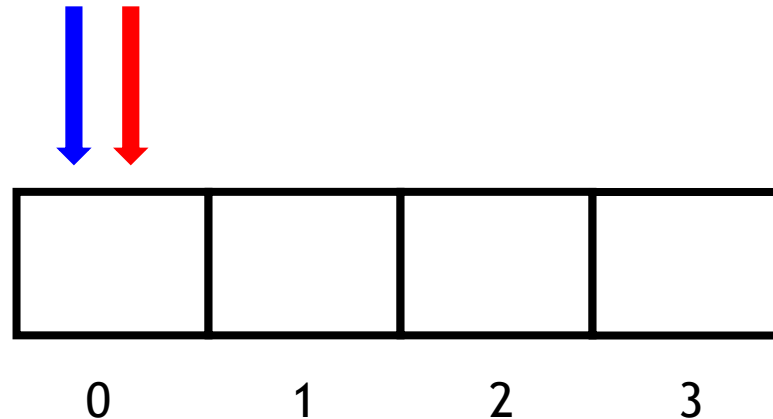


Fila estática

- Enfileirar o número 507

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=0
```

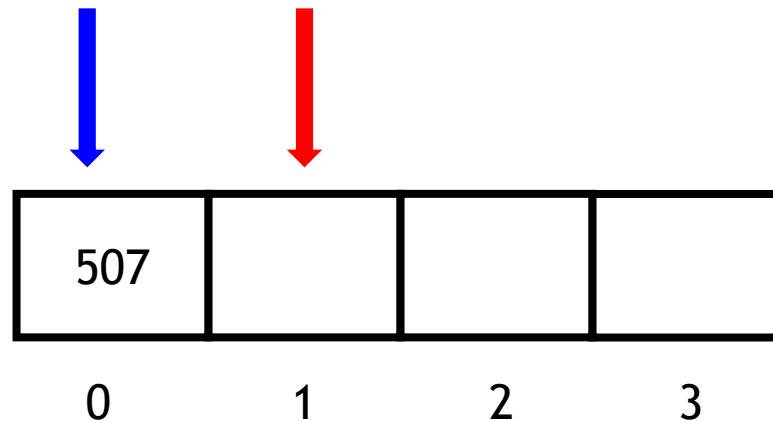


Fila estática

- Enfileirar o número 507

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=1
```

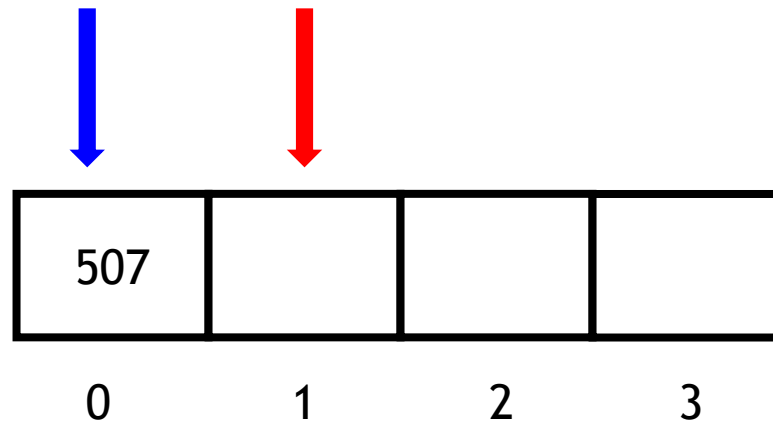


Fila estática

- Enfileirar o número 222

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=1
```

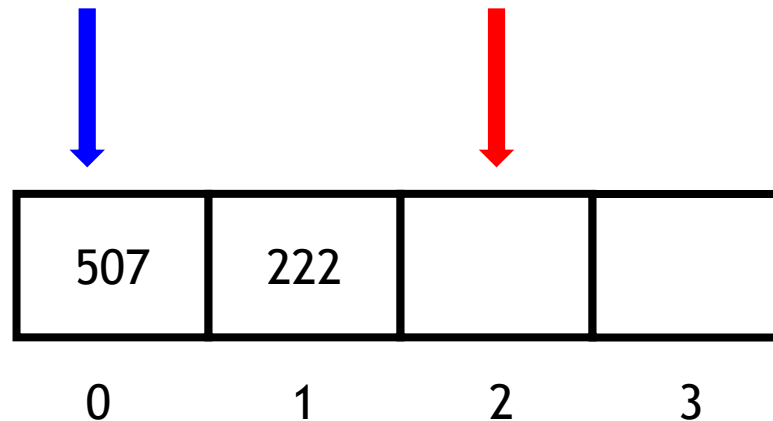


Fila estática

- Enfileirar o número 222

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=2
```

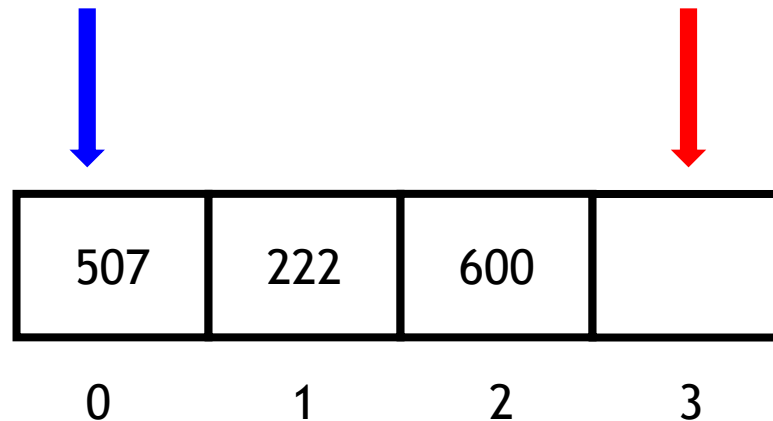


Fila estática

- Enfileirar o número 600

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=3
```



Fila estática

- Enfileirar o número 120

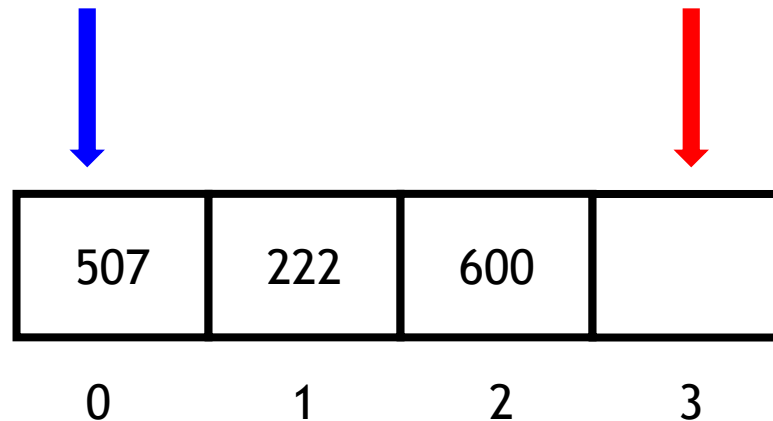


Erro! Fila está cheia!

$fim == tamanho - 1$

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=3
```

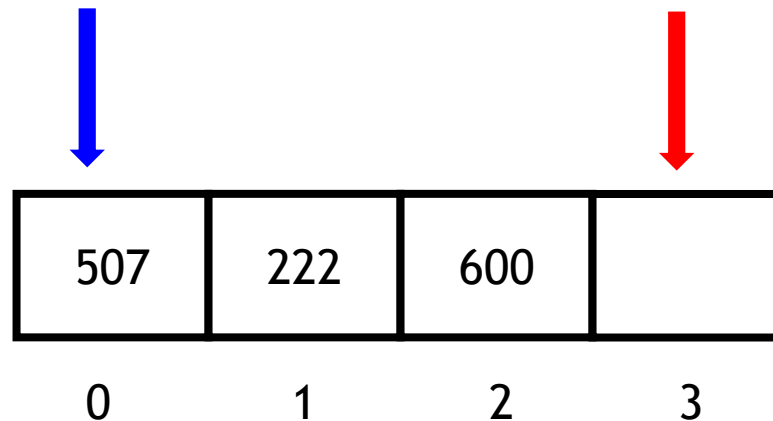


Fila estática

- Como desenfileirar?

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=3
```

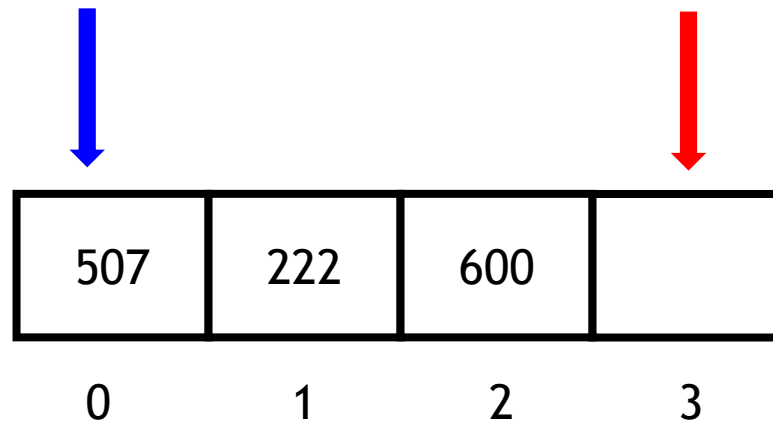


Fila estática

- Como desenfileirar?
 1. Salva número indicado por inicio (507);
 2. inicio++
 3. Retorna o número 507

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=3
```

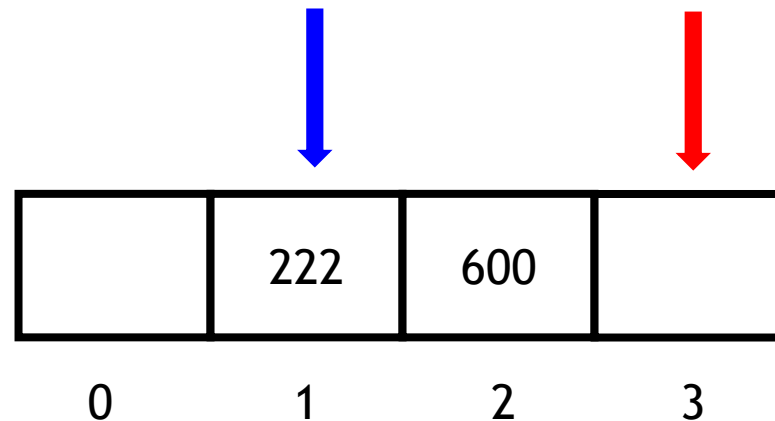


Fila estática

- Como desenfileirar?
 1. Salva número indicado por inicio (507);
 2. inicio++
 3. Retorna o número 507

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=1  
fim=3
```



Fila estática

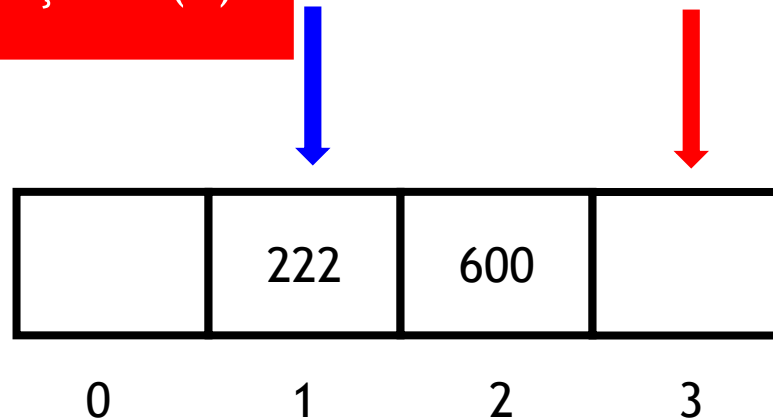
- Como desenfileirar?

Outra estratégia seria mover todos os itens para o início do arranjo (assim como a lista estática faz)

Mas isso tornaria a operação $O(n)$!

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=1  
fim=3
```

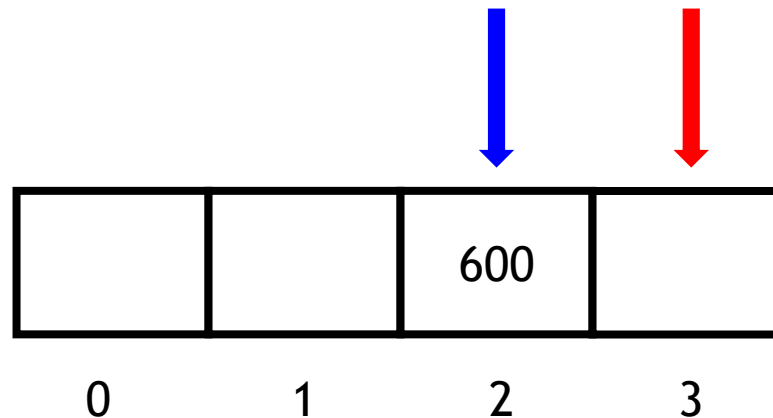


Fila estática

- Como desenfileirar?
 1. Salva número indicado por inicio (222);
 2. inicio++
 3. Retorna o número 222

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=2  
fim=3
```

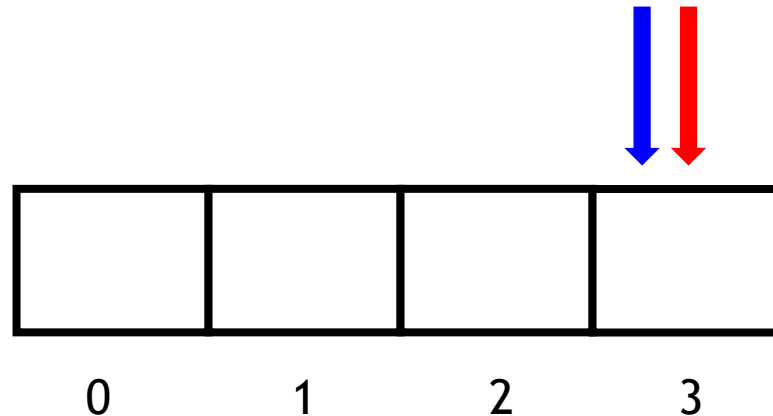


Fila estática

- Como desenfileirar?
 1. Salva número indicado por inicio (600);
 2. inicio++
 3. Retorna o número 600

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=3  
fim=3
```



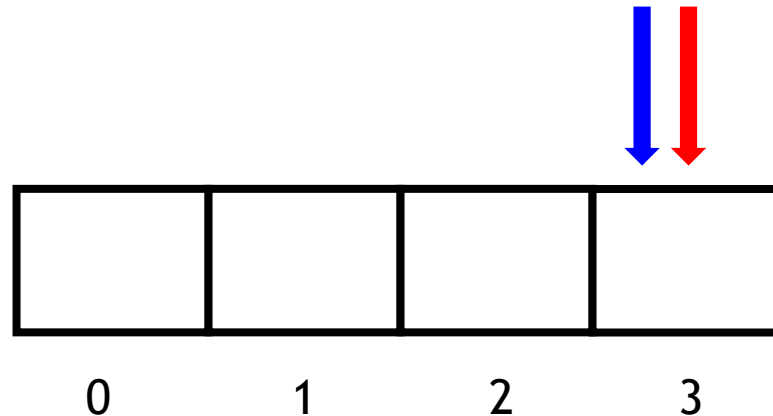
Fila estática

- **Fila ficou vazia!**

1. E se enfileirar novo item?
 - Overflow (retorna fila cheia)
 - **fim == tamanho-1**
2. Como solucionar esse problema?

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=3  
fim=3
```



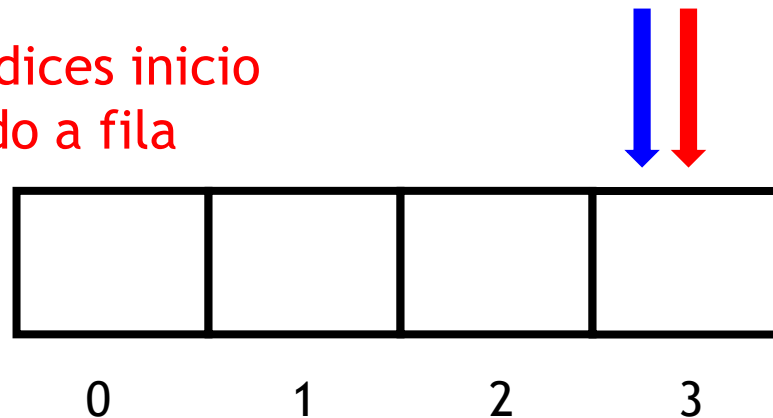
Fila estática

- **Fila ficou vazia!**

1. E se enfileirar novo item?
 - Overflow (retorna fila cheia)
 - **fim == tamanho-1**
2. Como solucionar esse problema?
 - **Reiniciar índices inicio e fim quando a fila ficar vazia.**

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=3  
fim=3
```



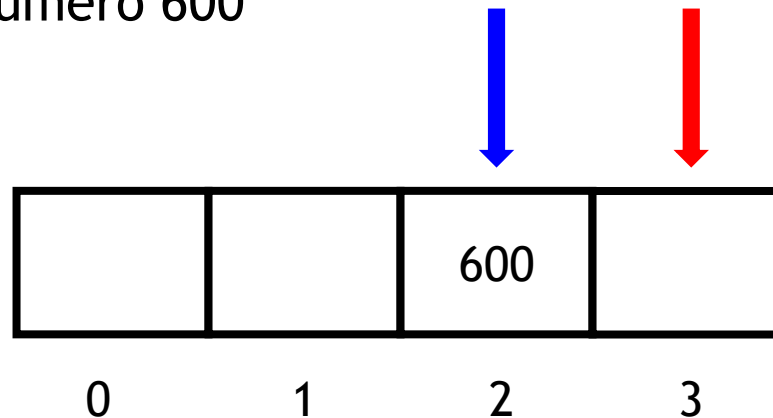
Fila estática

- Retomando a situação anterior...
 1. Salva número indicado por inicio (600);
 2. **inicio == fim-1**
 1. inicio=0
 2. fim=0
 3. Retorna o número 600

Fila vai ficar vazia!

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

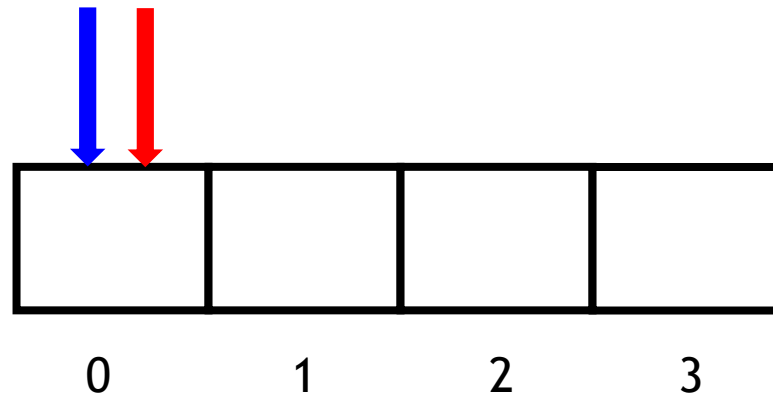
```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=2  
fim=3
```



Fila estática

- Retomando a situação anterior...
 1. Salva número indicado por inicio (600);
 2. **inicio == fim-1**
 1. inicio=0
 2. fim=0
 3. Retorna o número 600

Fila vai ficar vazia!



```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0  
fim=0
```

Fila estática

- Retomando a situação anterior...
 1. Salva número indicado por inicio (600);
 2. ~~inicio -= fim - 1~~

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0
```

Podemos melhorar essa solução?

Fila vai ficar vazia!



Fila estática

- Retomando a situação anterior...
 1. Salva número indicado por inicio (600);
 2. ~~inicio -= fim - 1~~

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=0
```

Podemos melhorar essa solução?
Sim, com a fila estática circular!

Fila vai ficar vazia!



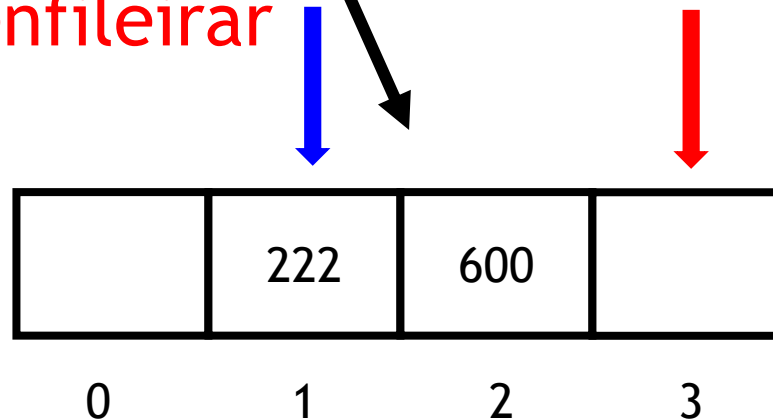
Fila estática

```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

- Neste caso, a fila não ficará vazia após desenfileirar 222

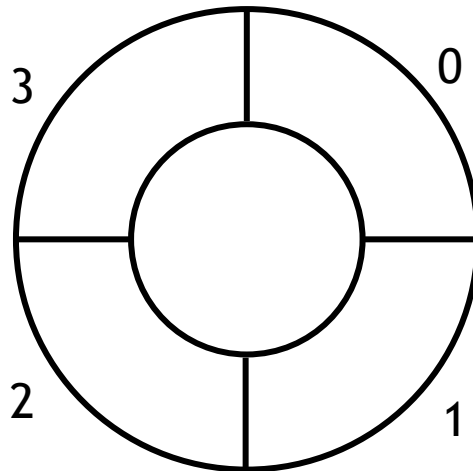
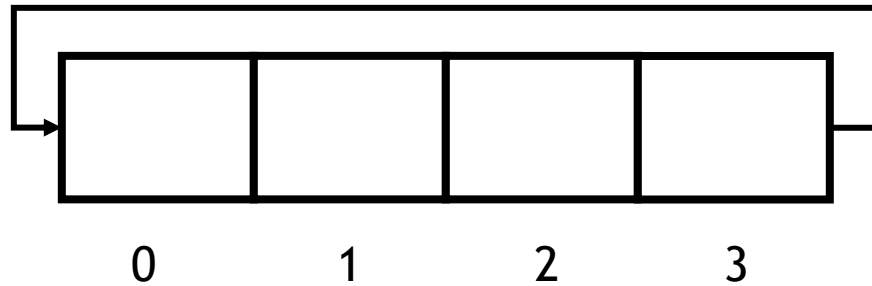
```
tamanho=4  
itens=malloc(sizeof(int)*4)  
inicio=1  
fim=3
```

- Mas ainda assim ocorrerá overflow se tentarmos enfileirar outro item!



Fila estática circular

- Trata o vetor como se estivesse em um círculo:



Fila estática circular

- Inicialização:
 - Estrutura igual à fila estática não circular;
 - O que muda é a forma de tratar os índices **inicio** e **fim**.

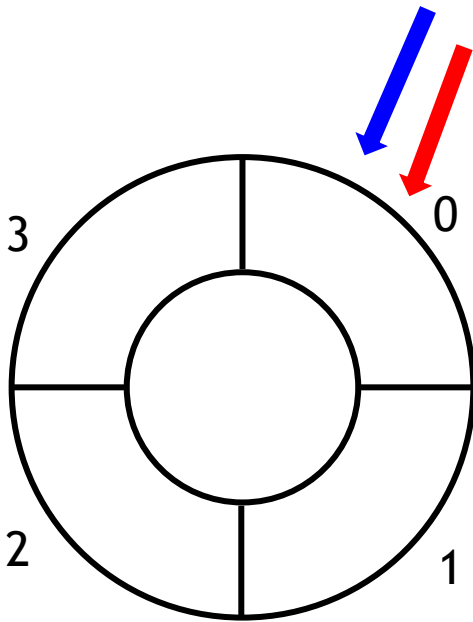
```
typedef struct Fila Fila;  
struct Fila {  
    int* itens;  
    int tamanho;  
    int inicio, fim;  
};
```

Fila estática circular

- Inicialização:

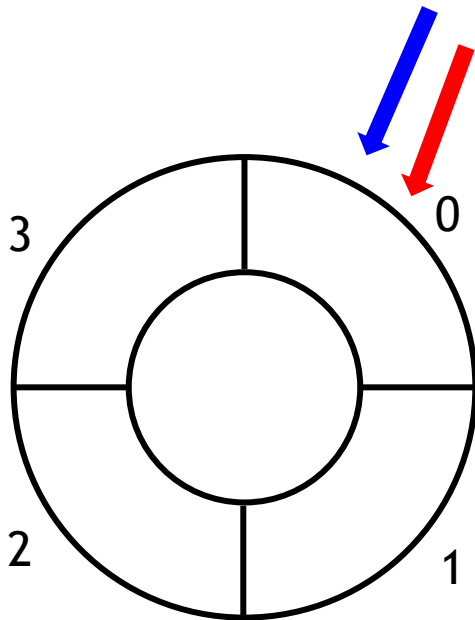
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=0
```

(início == fim) ==> Fila vazia!



Fila estática circular

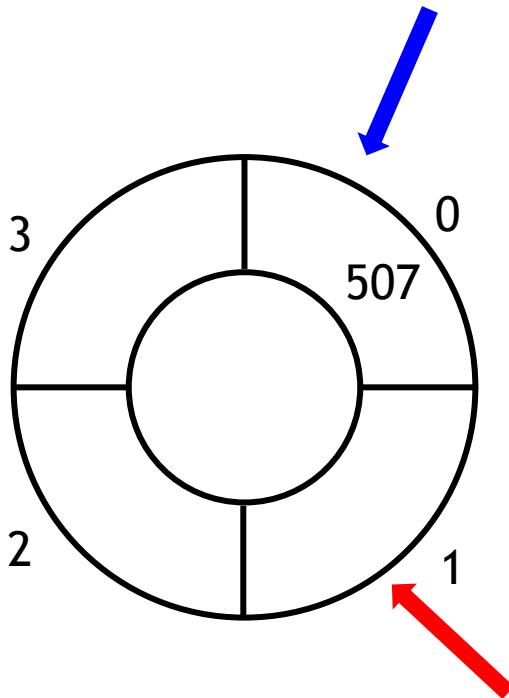
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=0
```



- Como enfileirar?
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - **novo_fim == início ?**
 - **Sim: fila cheia!**
 - Não: podemos enfileirar.

Fila estática circular

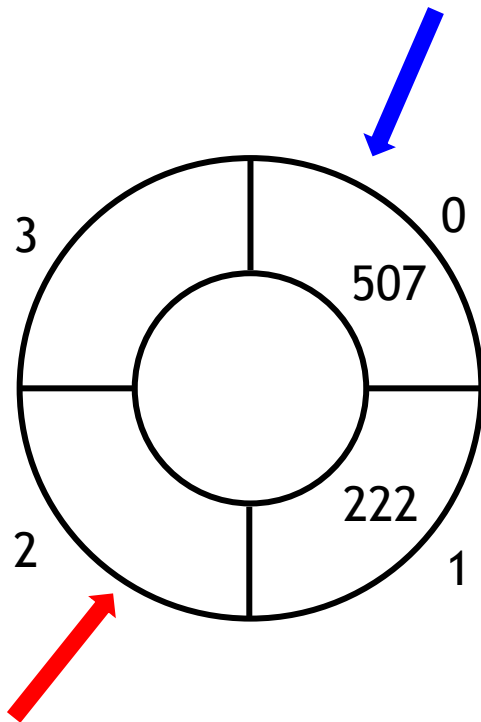
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=1
```



- Como enfileirar?
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - **novo_fim == início ?**
 - **Sim: fila cheia!**
 - Não: podemos enfileirar.
- Enfileirar 507:
 - Verifica se fila está cheia
 - $\text{itens}[\text{fim}] = 507$
 - $\text{fim} = \text{novo_fim}$

Fila estática circular

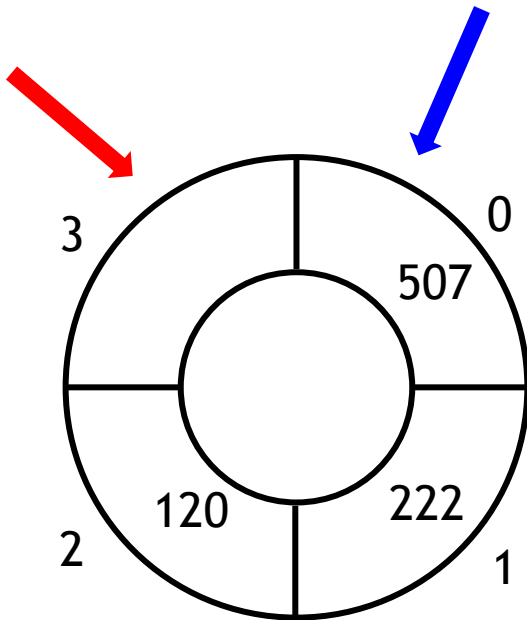
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=2
```



- Como enfileirar?
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - **novo_fim == início ?**
 - **Sim: fila cheia!**
 - Não: podemos enfileirar.
- Enfileirar 222:
 - Verifica se fila está cheia
 - $\text{itens}[\text{fim}] = 222$
 - $\text{fim} = \text{novo_fim}$

Fila estática circular

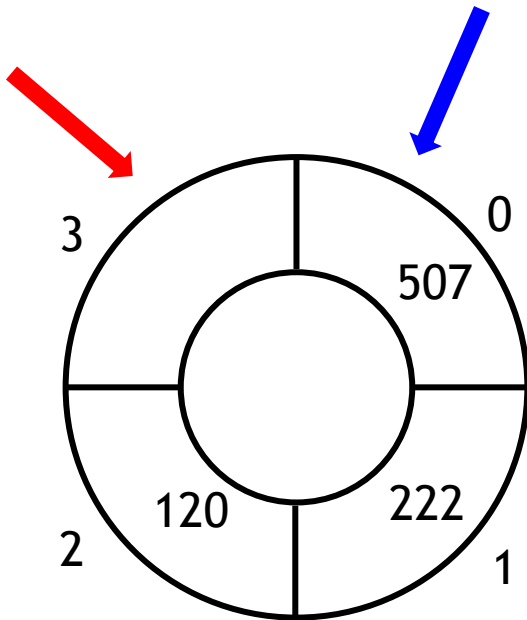
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=3
```



- Como enfileirar?
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - **novo_fim == início ?**
 - **Sim: fila cheia!**
 - Não: podemos enfileirar.
- Enfileirar 120:
 - Verifica se fila está cheia
 - $\text{itens}[\text{fim}] = 120$
 - $\text{fim} = \text{novo_fim}$

Fila estática circular

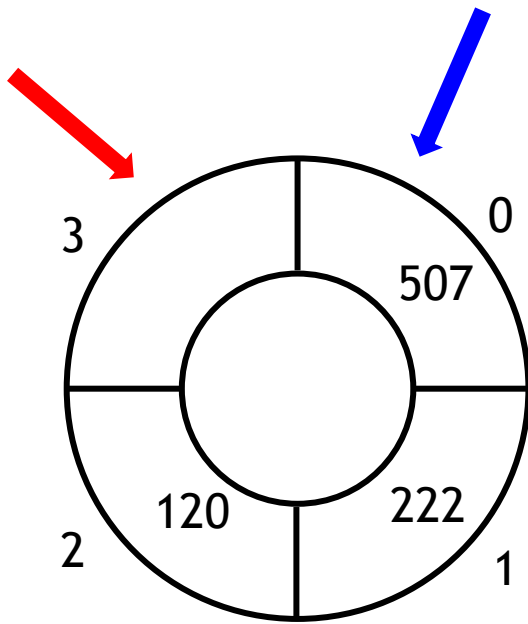
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=3
```



- Como enfileirar?
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - **$\text{novo_fim} == \text{início}$?**
 - **Sim: fila cheia!**
 - Não: podemos enfileirar.
- Enfileirar 300:
 - Verifica se fila está cheia
 - **Fila está cheia!**
 - **$(3+1)\%4 == 0$**

Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=3
```

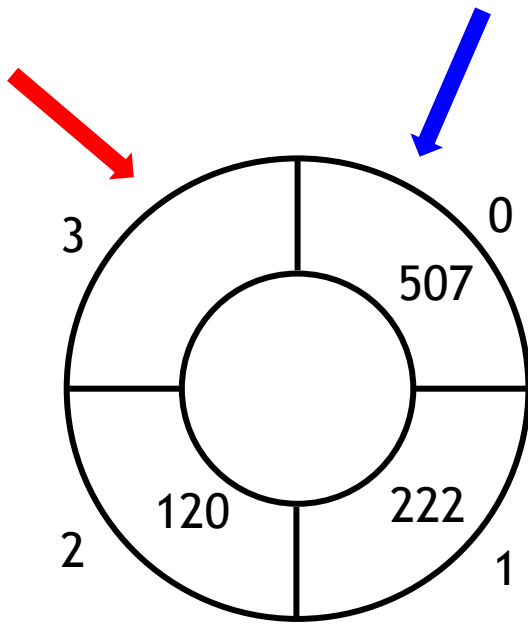


• Como desenfilear?

- **início==fim?**
 - **Sim: fila vazia!**
 - Não: Podemos desenfilear.
- Salva `itens[início]`
- `novo_início=(início+1) % tamanho`
- **início = novo_início**
- **Retorna item salvo**

Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=3
```

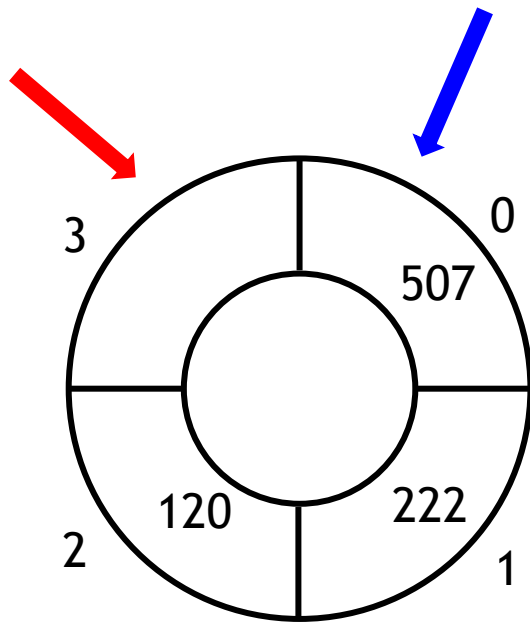


• Como desenfilear?

- **início==fim?**
 - **Sim: fila vazia!**
 - Não: Podemos desenfilear.
- Salva `itens[início]`
- `novo_início=(início+1) % tamanho`
- **início = novo_início**
- **Retorna item salvo**

Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=0
fim=3
```



• Como desenfilear?

Não

• início==fim?

- Sim: fila vazia!

- Não: Podemos desenfilear.

507

- Salva itens[início]

1

- novo_início=(início+1) % tamanho

- início = novo_início

- Retorna item salvo

Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=1
fim=3
```

- Como desenfileirar?

- **início==fim?**

- **Sim: fila vazia!**

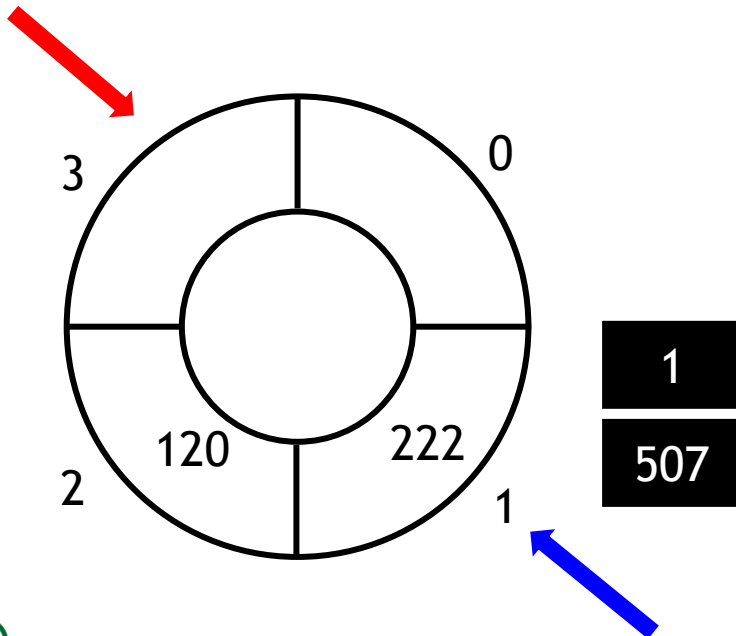
- Não: Podemos desenfileirar.

- Salva itens[início]

- $\text{novo_início} = (\text{início} + 1) \% \text{tamanho}$

- **início = novo_início**

- Retorna item salvo



Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=1
fim=3
```

• Como desenfilear?

Não

• início==fim?

- Sim: fila vazia!

- Não: Podemos desenfilear.

222

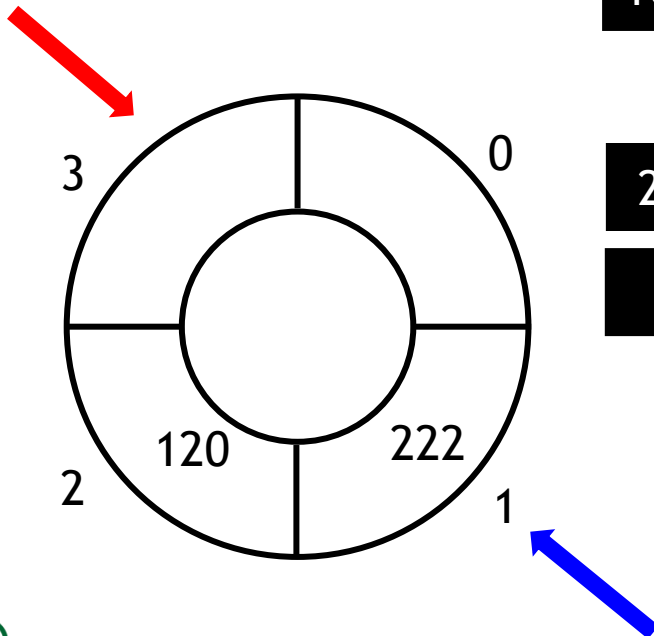
- Salva itens[início]

2

- novo_início=(início+1) % tamanho

- início = novo_início

- Retorna item salvo



Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=2
fim=3
```

- Como desenfileirar?

- **início==fim?**

- **Sim: fila vazia!**

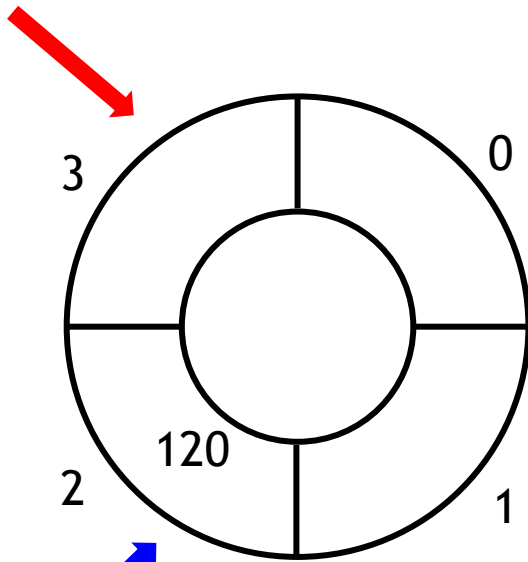
- Não: Podemos desenfileirar.

- Salva `itens[início]`

- `novo_início=(início+1) % tamanho`

- `início = novo_início`

- Retorna item salvo



2

222

Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=2
fim=3
```

• Como desenfileirar?

Não

• início==fim?

- Sim: fila vazia!

- Não: Podemos desenfileirar.

120

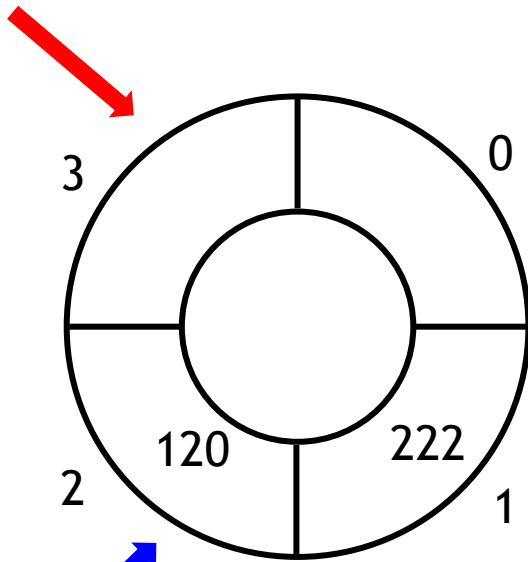
- Salva itens[início]

3

- novo_início=(início+1) % tamanho

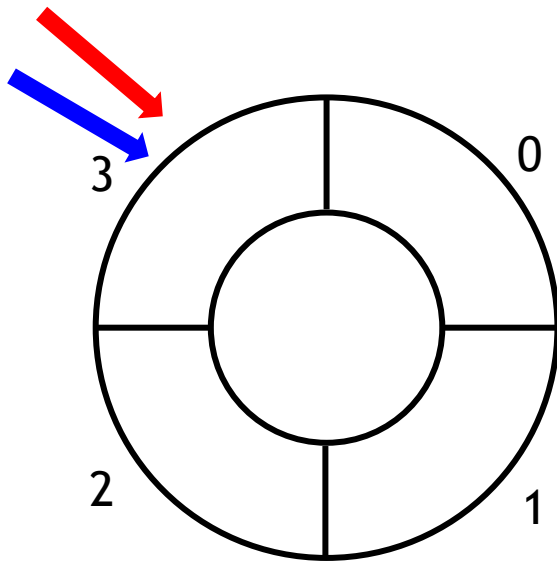
- início = novo_início

- Retorna item salvo



Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=3
fim=3
```



3
120

- Como desenfilear?

- **início==fim?**

- **Sim: fila vazia!**

- Não: Podemos desenfilear.

- Salva `itens[início]`

- `novo_início=(início+1) % tamanho`

- `início = novo_início`

- Retorna item salvo

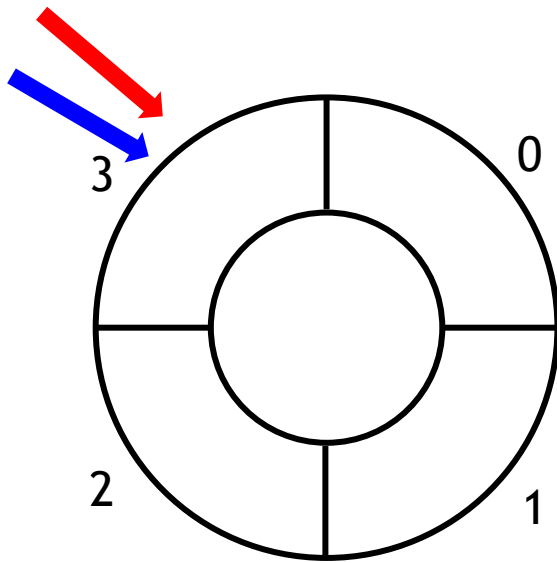
Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=3
fim=3
```

- Como desenfileirar?

- **início==fim?**

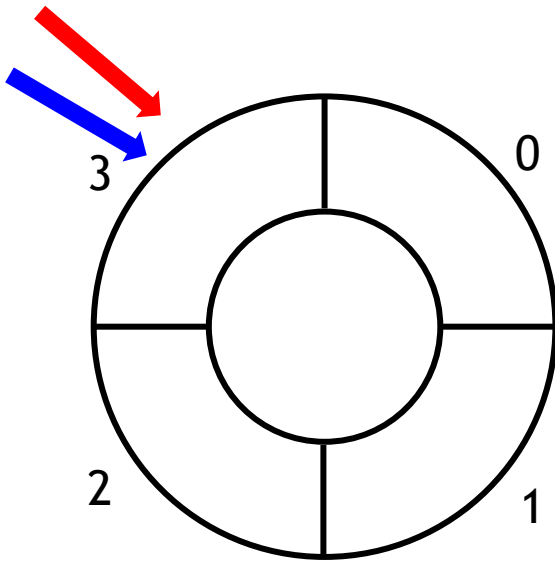
- **Sim: fila vazia!**
 - Não: Podemos desenfileirar.



(início == fim) ==> Fila vazia!

Fila estática circular

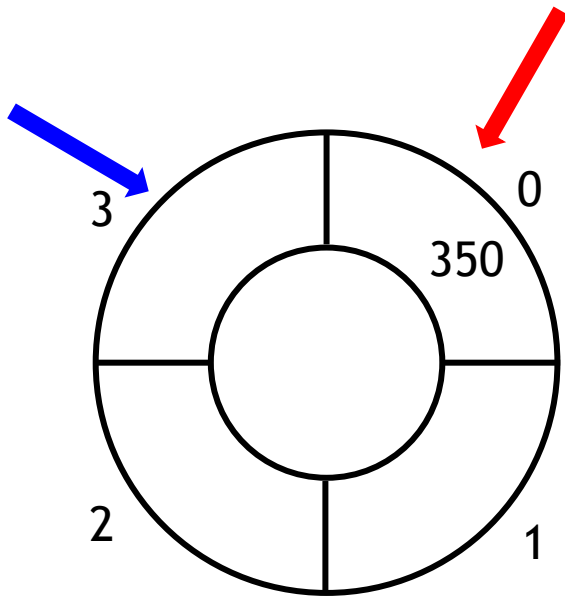
```
itens=malloc(sizeof(int)*4)
tamanho=4
início=3
fim=3
```



- Enfileirar 350:
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - $\text{novo_fim} = (3 + 1) \% 4 = 0$
 - **$\text{novo_fim} = \text{início} ?$**
 - Não: então podemos enfileirar!

Fila estática circular

```
itens=malloc(sizeof(int)*4)
tamanho=4
início=3
fim=0
```



- Enfileirar 350:
 - $\text{novo_fim} = (\text{fim} + 1) \% \text{tamanho}$
 - $\text{novo_fim} = (3 + 1) \% 4 = 0$
 - **$\text{novo_fim} == \text{início} ?$**
 - Não: então podemos enfileirar!
 - $\text{itens}[\text{fim}] = 350$
 - $\text{fim} = \text{novo_fim}$

Fila dinâmica

Fila dinâmica

- **Fila dinâmica**: implementa a estrutura de dados utilizando uma **lista ligada**;
- Portanto, os itens são alocados em memória de acordo com a necessidade.

Fila dinâmica

- Estrutura básica:

Ponteiro para o início da fila

Ponteiro para o fim da fila

```
typedef struct Fila Fila;  
struct Fila {  
    ListNode *inicio;  
    ListNode *fim;  
};
```

```
typedef struct ListNode ListNode;  
struct ListNode {  
    int dados;  
    ListNode *next;  
};
```

Fila dinâmica

- Inicialização

```
início=NULL  
fim=NULL
```

Início==NULL ==> Fila vazia!

Fila dinâmica

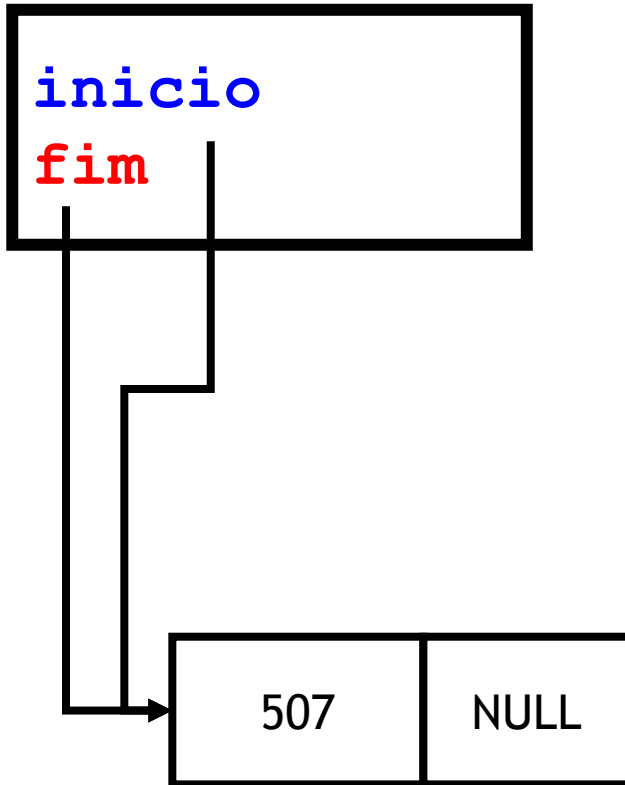
- Como enfileirar?

```
início=NULL  
fim=NULL
```

1. Alocar novo ListNode
2. Adicioná-lo logo após o item apontado por **fim**
3. Atualizar ponteiro **fim**
4. Quando a fila está vazia, atualizar ponteiro **início** também

Fila dinâmica

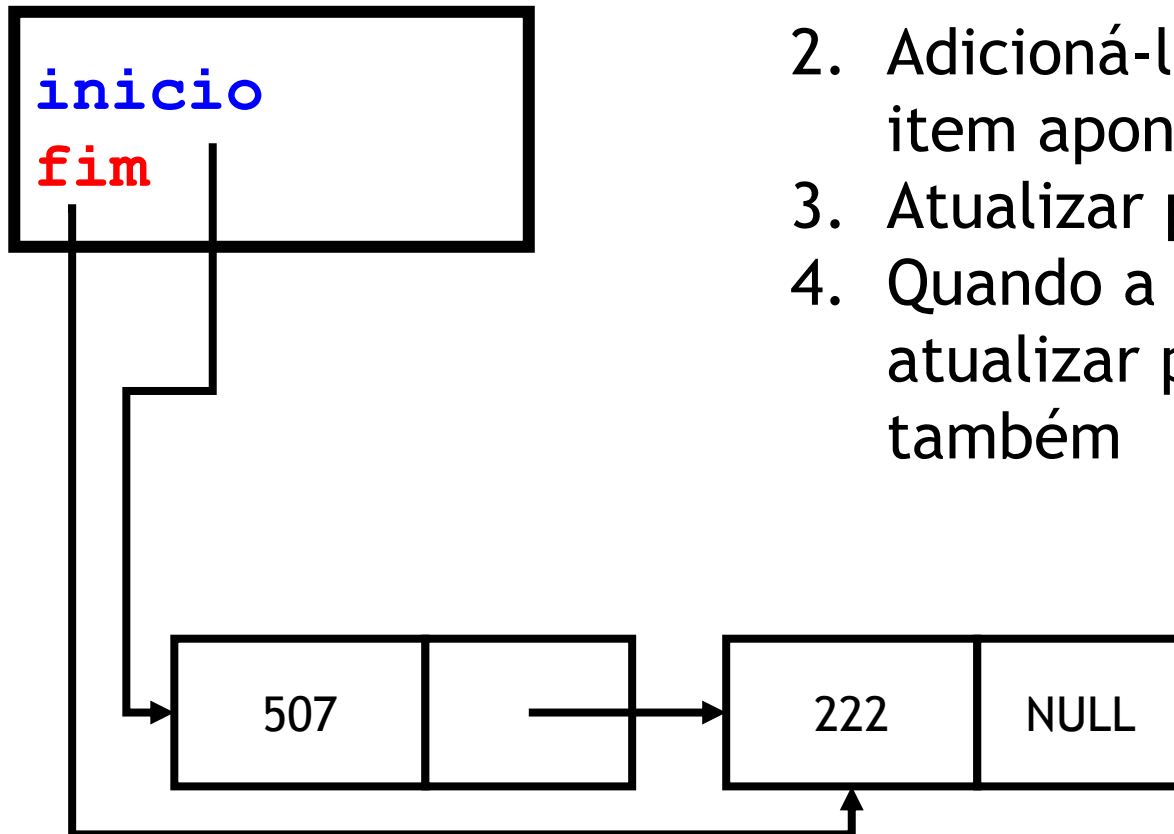
- Como enfileirar?



1. Alocar novo ListNode
2. Adicioná-lo logo após o item apontado por **fim**
3. Atualizar ponteiro **fim**
4. Quando a fila está vazia, atualizar ponteiro **inicio** também

Fila dinâmica

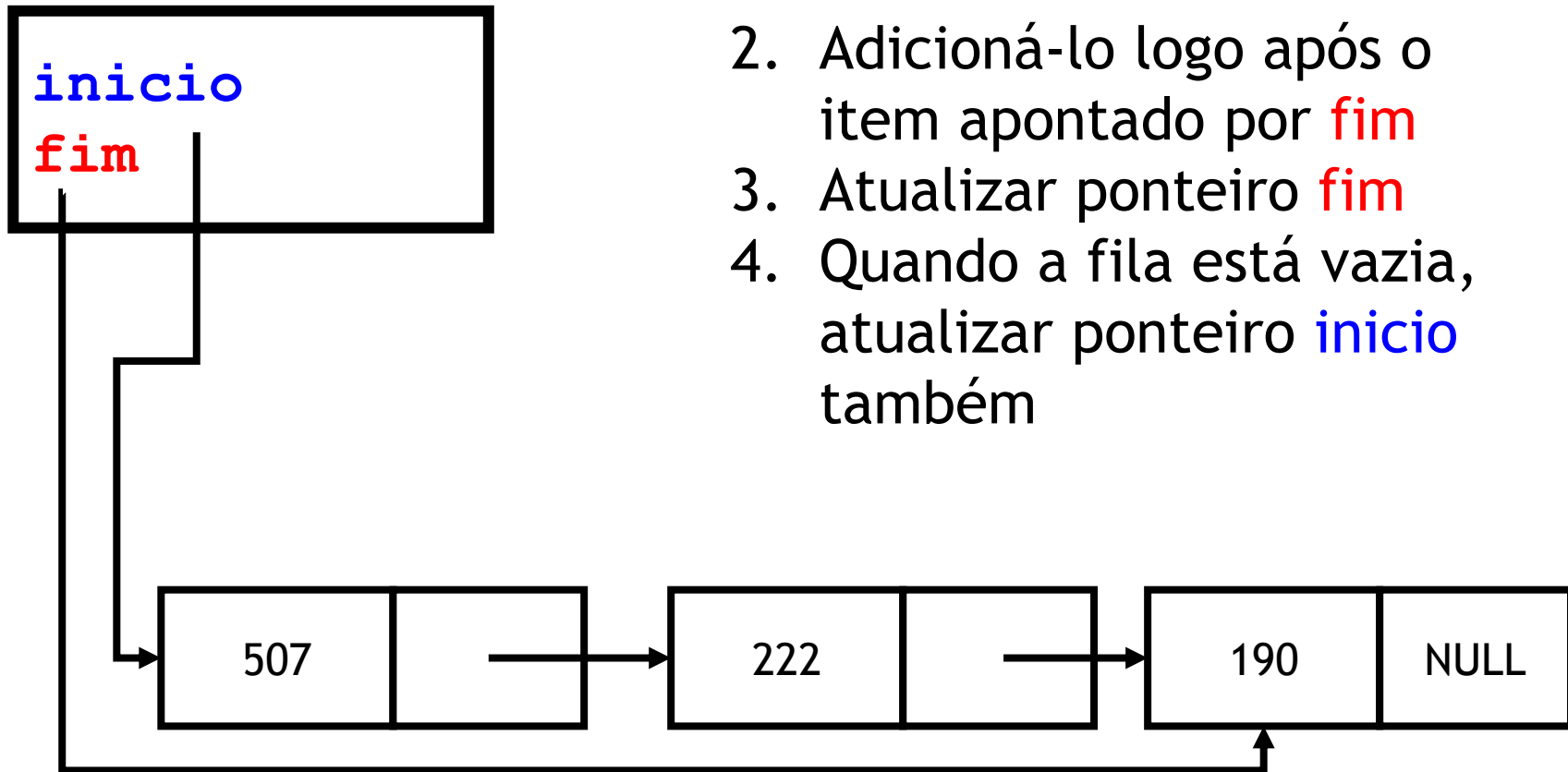
- Como enfileirar?



1. Alocar novo ListNode
2. Adicioná-lo logo após o item apontado por **fim**
3. Atualizar ponteiro **fim**
4. Quando a fila está vazia, atualizar ponteiro **inicio** também

Fila dinâmica

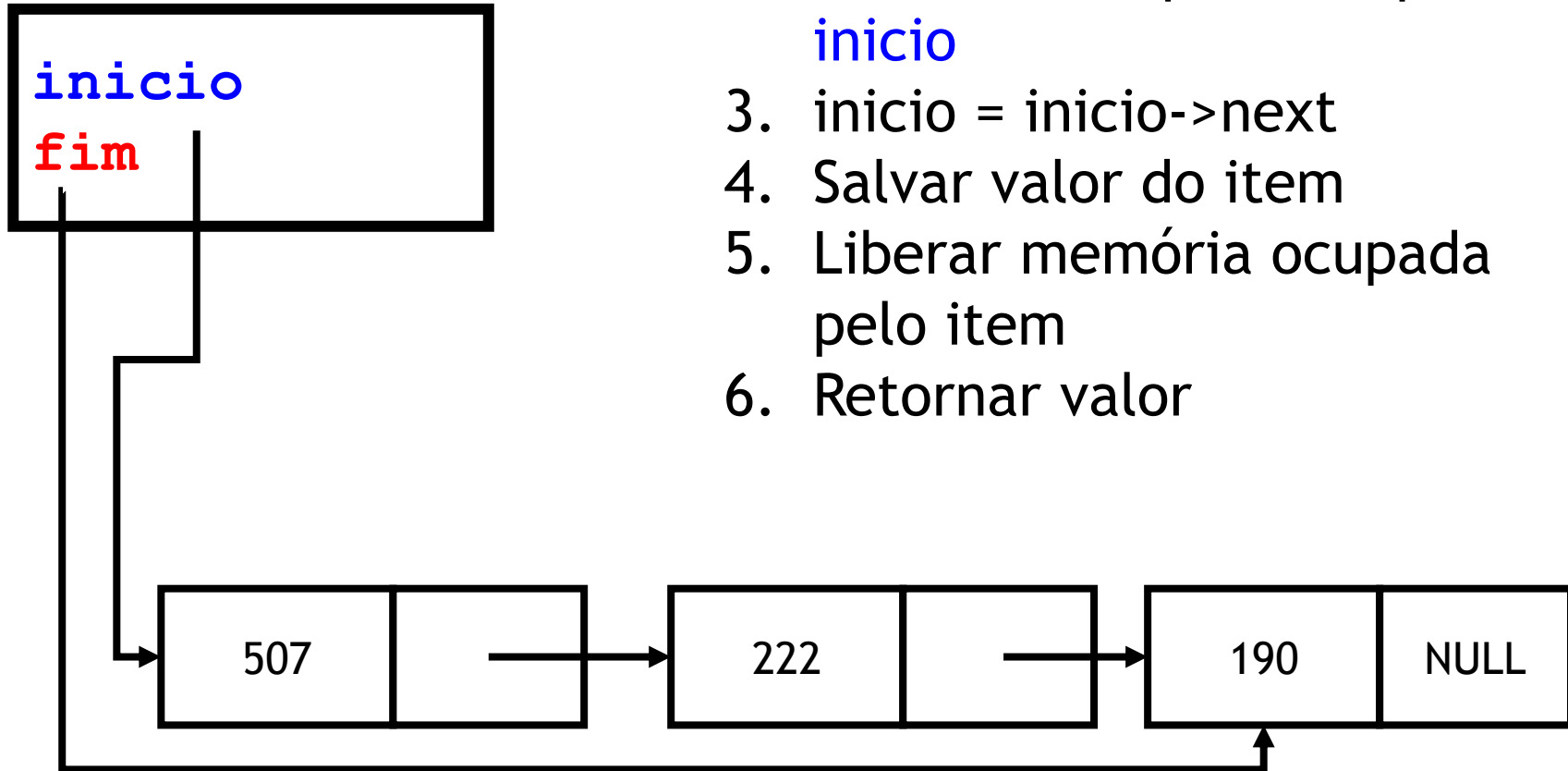
- Como enfileirar?



1. Alocar novo `LinkedListNode`
2. Adicioná-lo logo após o item apontado por `fim`
3. Atualizar ponteiro `fim`
4. Quando a fila está vazia, atualizar ponteiro `inicio` também

Fila dinâmica

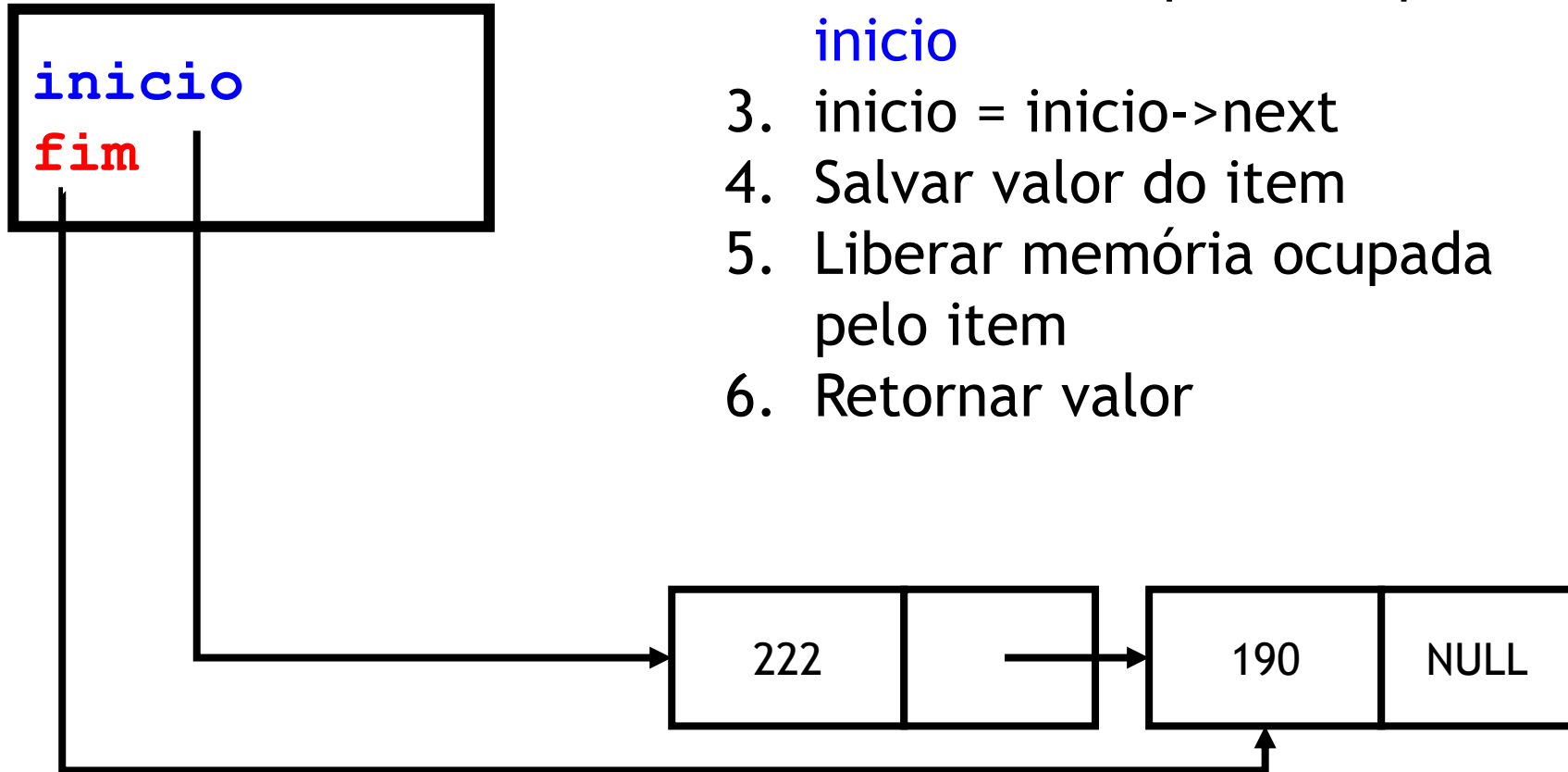
- Como desenfileirar?



1. Verificar se fila está vazia
2. Salvar item apontado por **inicio**
3. $inicio = inicio \rightarrow next$
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor

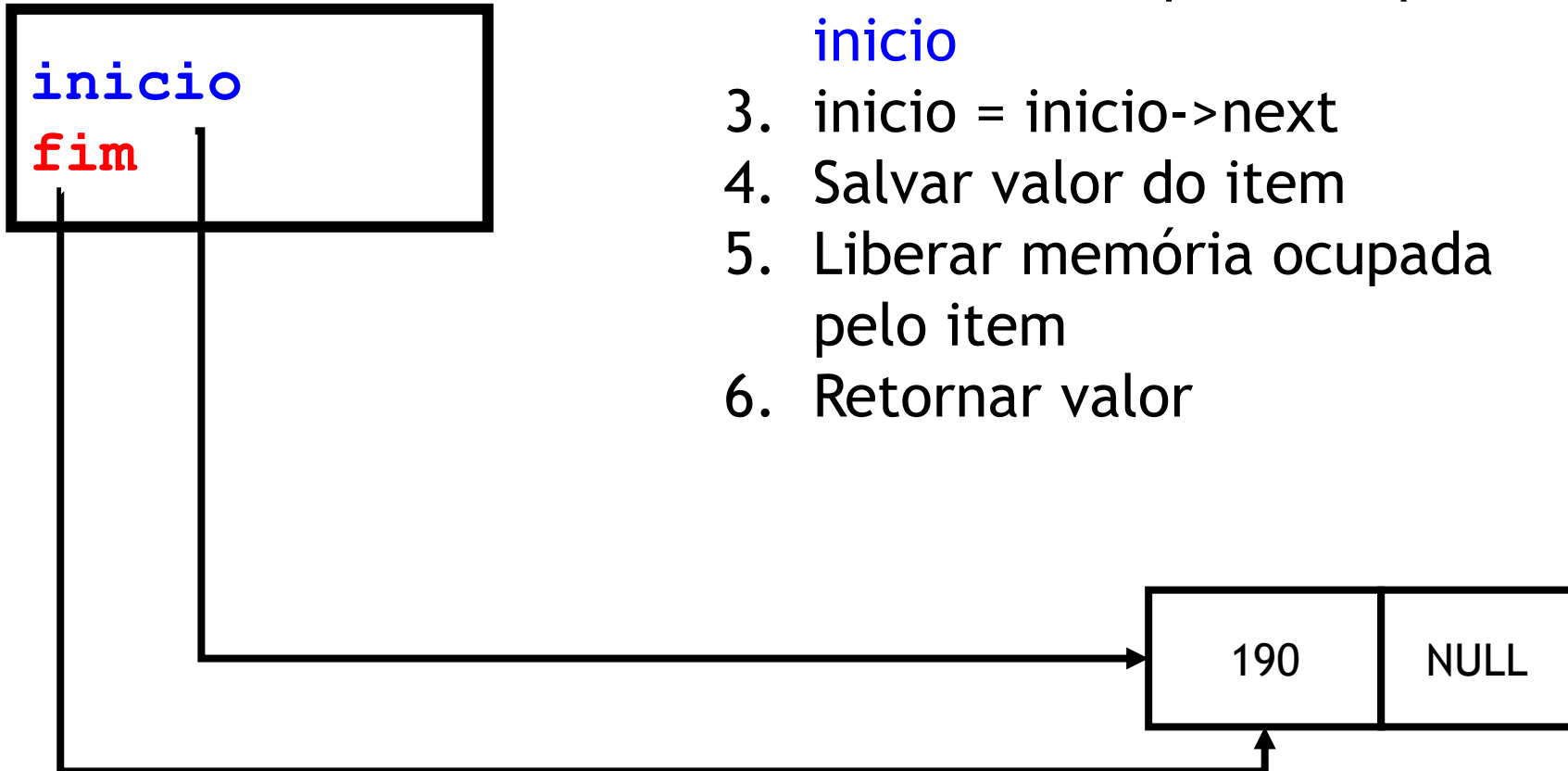
Fila dinâmica

- Como desenfileirar?



Fila dinâmica

- Como desenfileirar?



1. Verificar se fila está vazia
2. Salvar item apontado por **inicio**
3. $inicio = inicio \rightarrow next$
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor

Fila dinâmica

- Como desenfileirar?

```
início=NULL  
fim=NULL
```

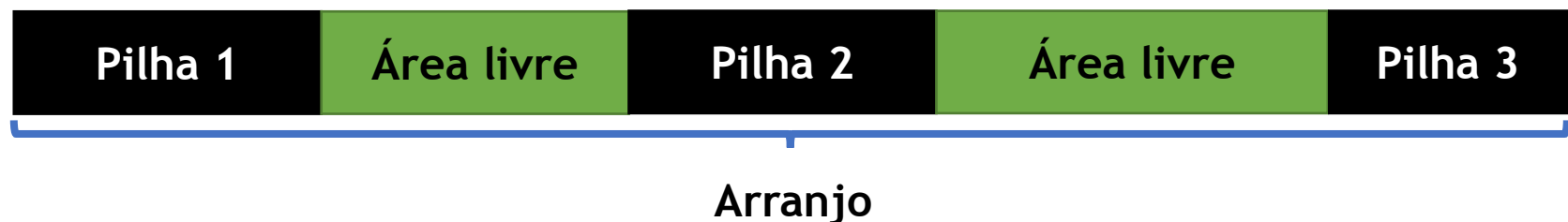
1. Verificar se fila está vazia
2. Salvar item apontado por **início**
3. `início = início->next`
4. Salvar valor do item
5. Liberar memória ocupada pelo item
6. Retornar valor

Quando desenfileirar o último elemento, volte para `início=fim=NULL`

Outros tipos

Outros tipos

- **Deque:** permite remover elementos do início ou do fim (combinação de fila e pilha);
- **Pilha auto-ajustável:** pilha estática que aloca/desaloca memória de acordo com a carga;
- **Pilhas múltiplas:** permite que mais de uma pilha compartilhe o mesmo arranjo.



Resumo

Resumo

Filas

Operações:

- Enfileirar
- Desenfileirar

FIFO

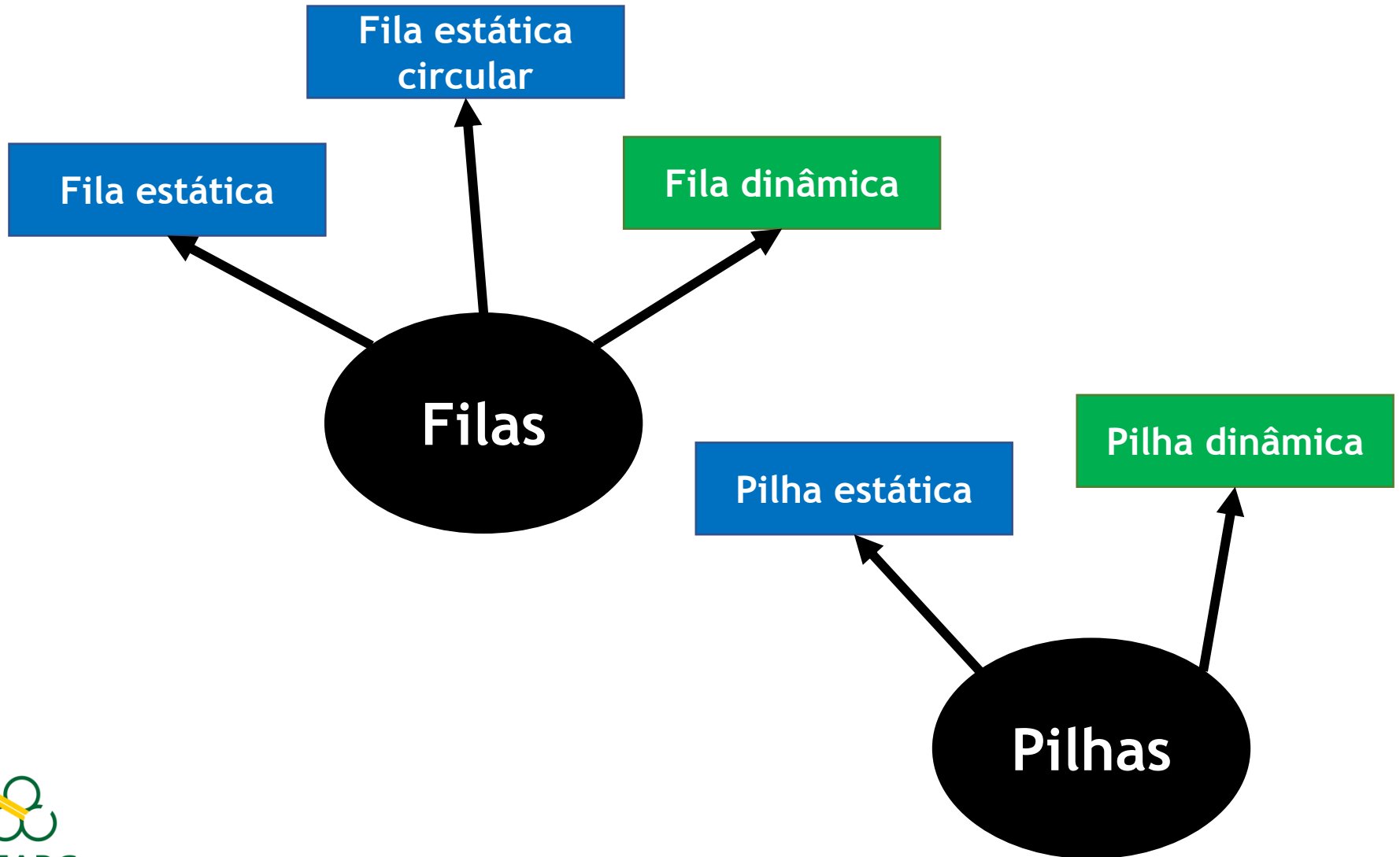
Pilhas

Operações:

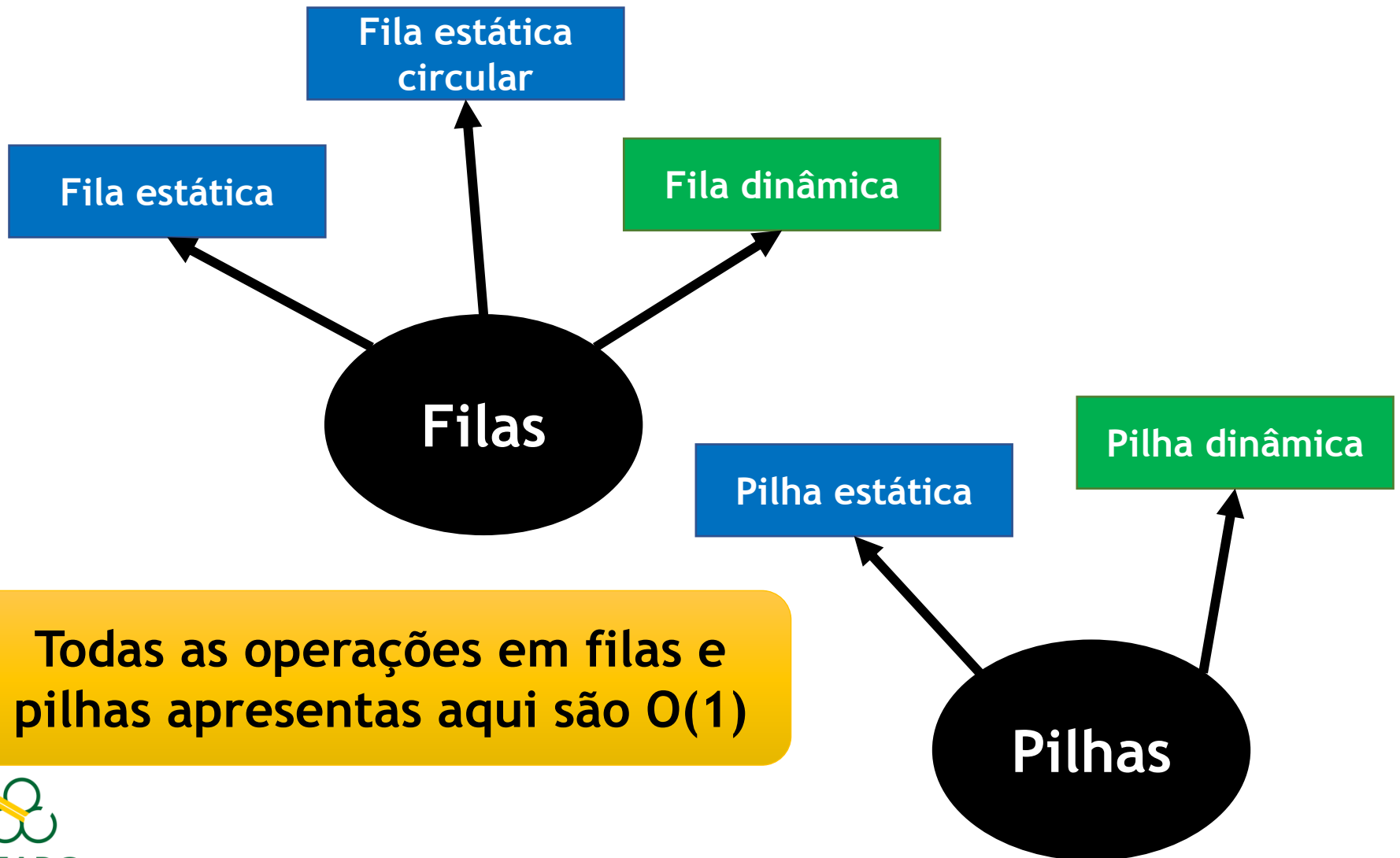
- Empilhar
- Desempilhar

LIFO

Resumo



Resumo



Todas as operações em filas e pilhas apresentas aqui são $O(1)$

Exercícios

- Implemente as operações:
 - Pilha estáticas: Empilhar/Desempilhar
 - Pilha dinâmica: Empilhar/Desempilhar
 - Pilha auto-ajustável (material Profa. Mirtha): Empilhar/Desempilhar
 - Fila estática circular: Enfileirar/Desenfileirar
 - Fila dinâmica: Enfileirar/Desenfileirar

Referências

- Nivio Ziviani. Projeto de Algoritmos: com implementações em Pascal e C. Cengage Learning, 2015.
- Robert Sedgwick. Algorithms in C/C++/Java, Parts 1-4 (Fundamental Algorithms, Data Structures, Sorting, Searching). Addison-Wesley Professional.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos: Teoria e Prática. Elsevier, 2012.