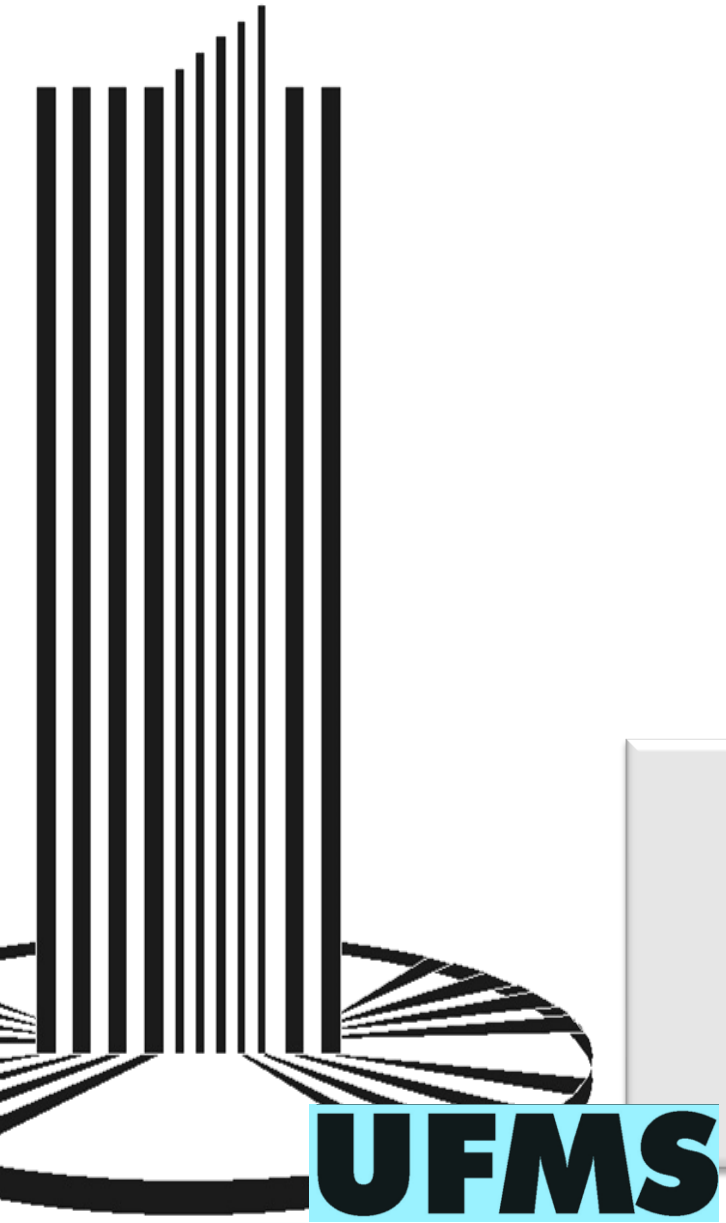
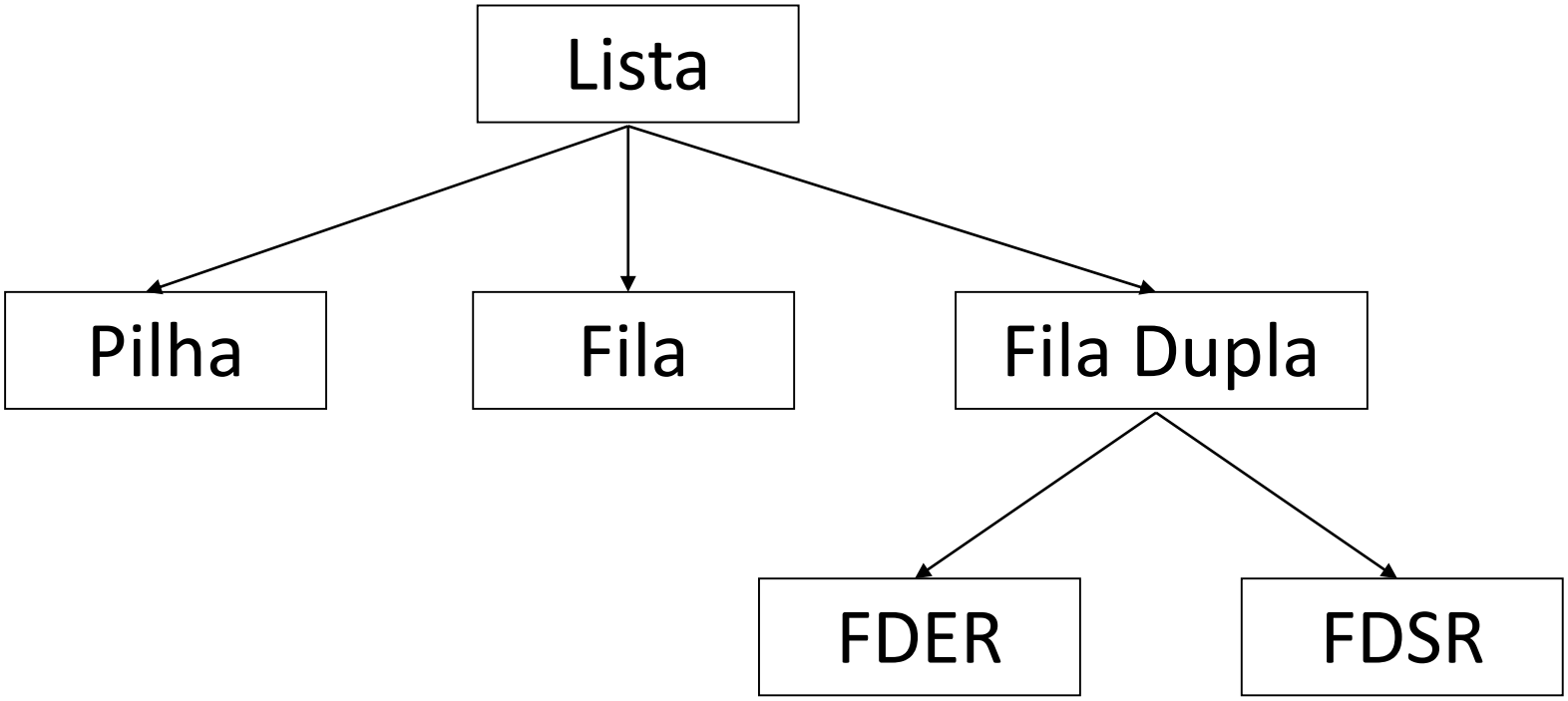


Fila

Profª Yorah Bosse

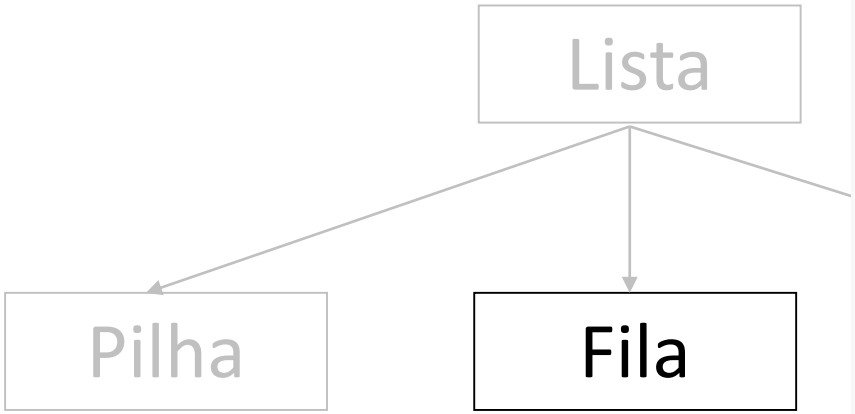
yorah.bosse@gmail.com
yorah.bosse@ufms.br

Listas Especiais



Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Listas Especiais

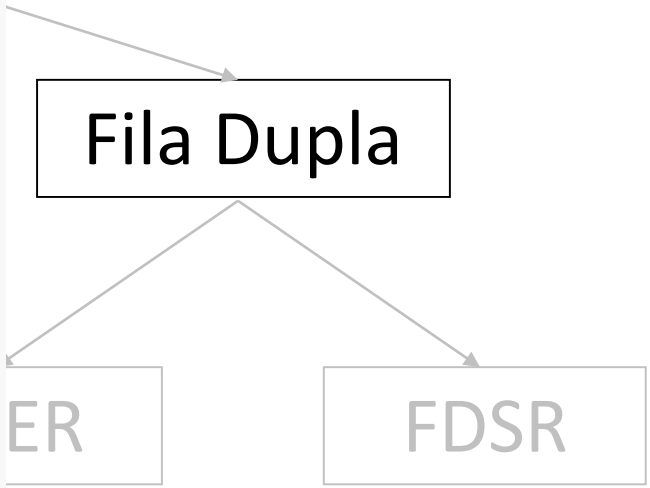


Fila é uma lista especial também denominada de FIFO (First In, First Out), ou seja, o primeiro que entra é o primeiro que sai. As entradas e saídas, nesta lista, ocorrem em extremidades opostas.

Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Listas Especiais

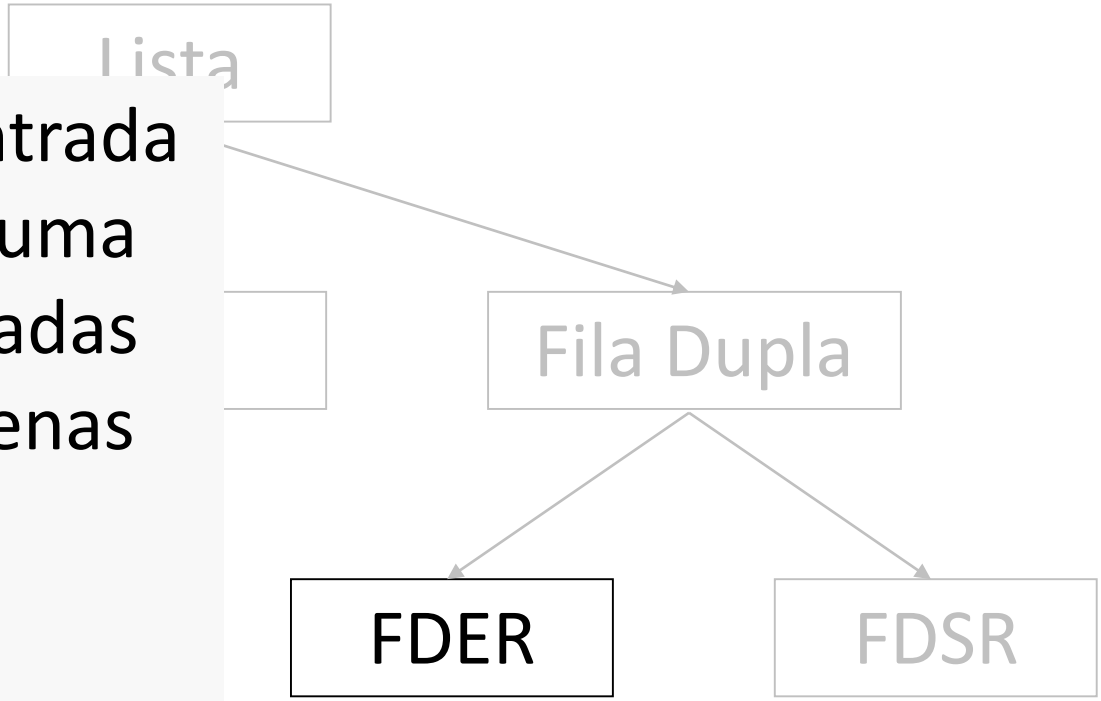
Fila Dupla é uma lista especial também denominada de DEQUE (Double-Ended QUEue), ou seja, fila de extremidade dupla. As entradas, saídas e acessos podem ser feitos pelas duas extremidades.



Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Listas Especiais

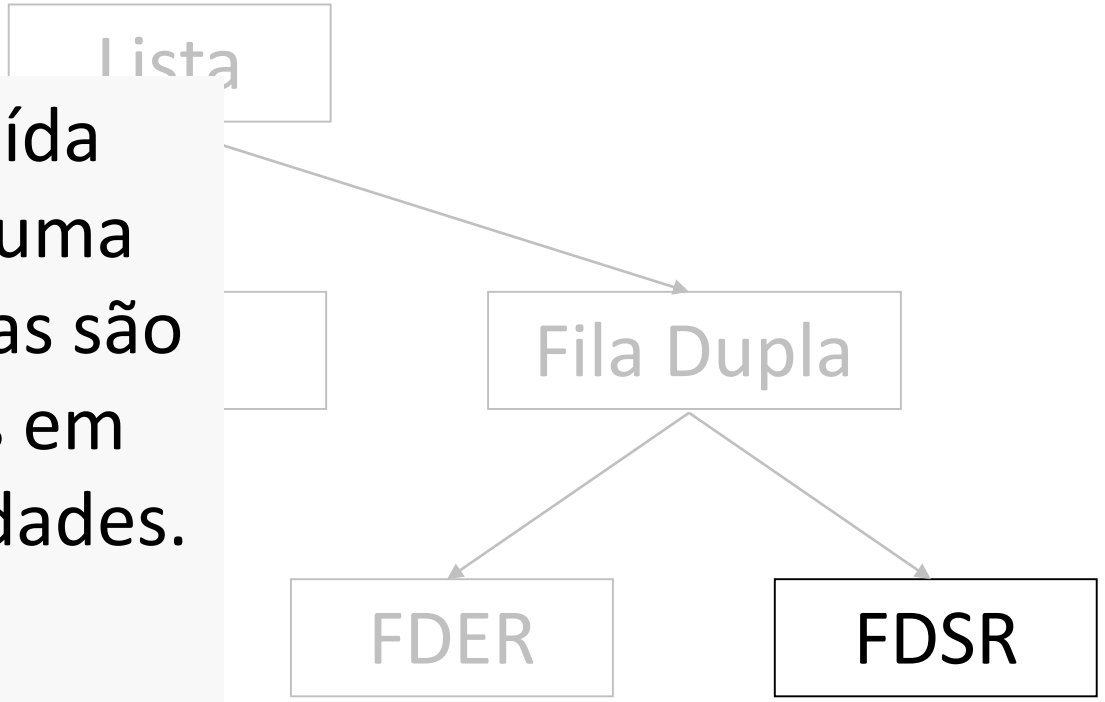
Fila Dupla com Entrada Restrita (FDER) é uma lista onde as entradas são realizadas apenas em uma das extremidades.



Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Listas Especiais

Fila Dupla com Saída Restrita (FDSR) é uma lista onde as saídas são realizadas apenas em uma das extremidades.



Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Formas de Implementação

- **Seqüencial**: Os elementos estão fisicamente contíguos, um após o outro. Pode ser representada por um vetor na memória principal ou um arquivo sequencial em disco.

- **Encadeada**:
  - Estática: Inicio da lista é definido por uma variável inteiro que estabelece onde começa o encadeamento. Cada elemento possui um campo do tipo int, que estabelece o sucessor do mesmo.
    - Desvantagens: Quantidade máxima de elementos estabelecida (Alocação Estática).
  - Dinâmica: Início da lista é definido por um apontador que armazena o endereço do elemento inicial da lista. Cada elemento da lista possui um campo do tipo apontador que armazena o endereço do próximo elemento da lista. Cada elemento da lista é alocado dinamicamente.

Listas Lineares

- Abaixo uma relação de algumas Listas Lineares :
  - Sequêncial
  - Estática Simples* ou duplamente Encadeada
  - Dinâmica Simples* ou duplamente* Encadeada
  - Dinâmica Simples ou duplamente* Encadeada com descritor
  - Dinâmica Circular Simples ou duplamente* Encadeada
  - Pilhas (stack) estáticas e dinâmicas* simples* ou duplamente Encadeadas
  - Filas (queue) estáticas e dinâmicas* simples* ou duplamente Encadeadas
    - Fila Estática
    - Fila Dinâmica

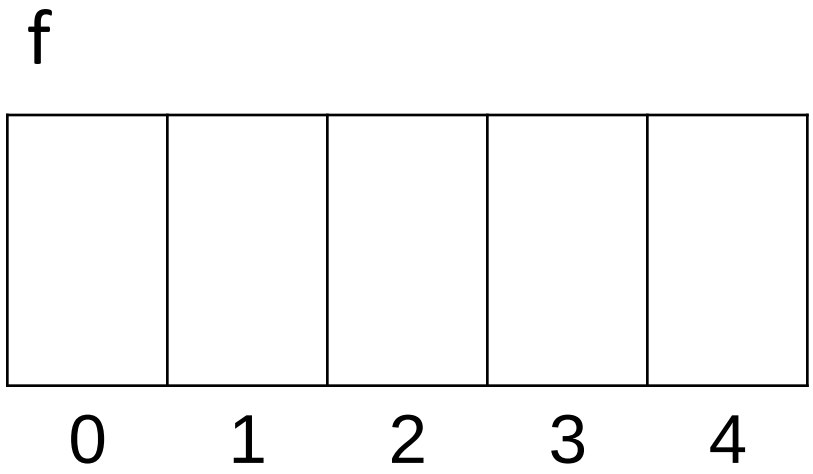
* serão estudadas nesta disciplina

- Operações possíveis
  - Inicializar
  - Incluir no fim da fila – Empilhar - Enqueue
  - Excluir do inicio da fila - Desempilhar - Dequeue
  - Acessar Inicio e Fim da fila sem removê-los
  - Verificar se a fila está vazia

Implementação - Fila Estática (Problema)

```
typedef struct dadosFila{
        char f[max];
        int inicio, fim;
}sFila;
```

- Criação:



inicio :

fim :

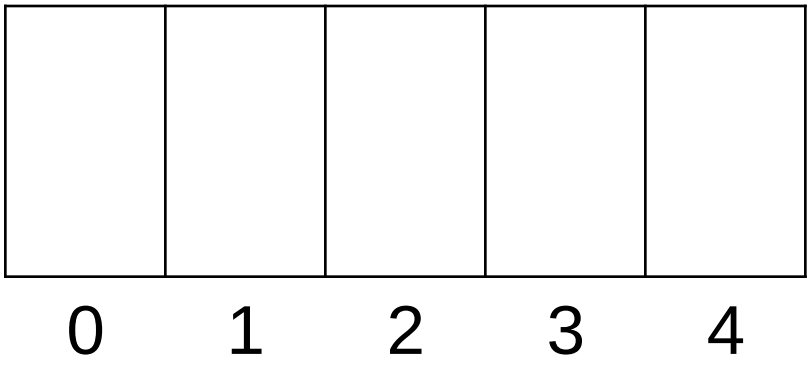
Implementação - Fila Estática (Problema)

```
typedef struct dadosFila{
        char f[max];
        int inicio, fim;
}sFila;
```

- Criação:

Nome da Fila (vetor)

f



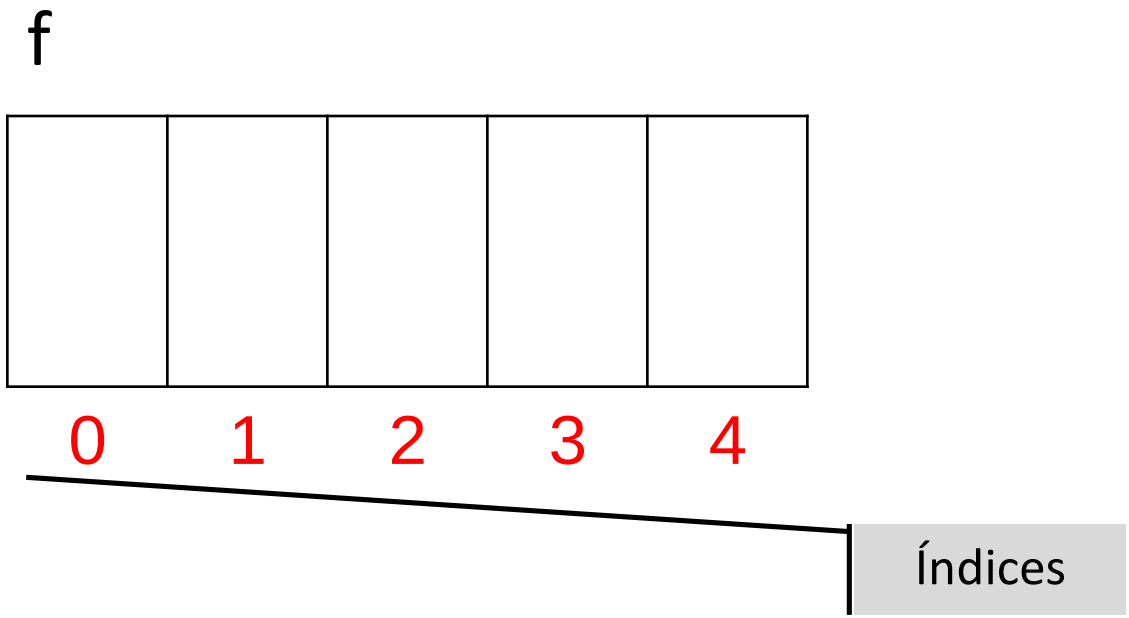
inicio :

fim :

Implementação - Fila Estática (Problema)

```
typedef struct dadosFila{
        char f[max];
        int inicio, fim;
}sFila;
```

- Criação:



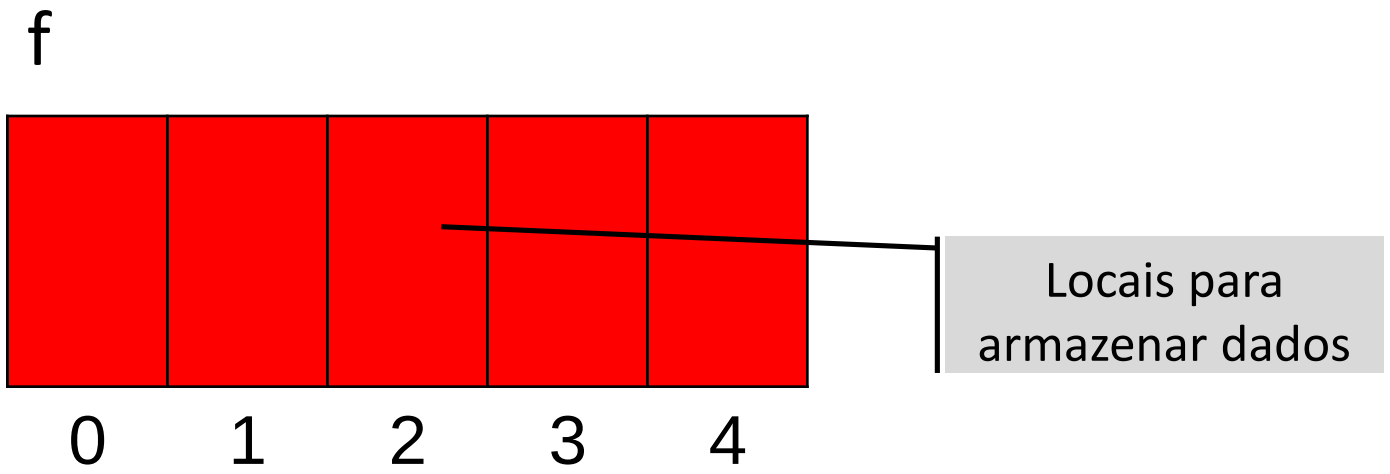
inicio :

fim :

Implementação - Fila Estática (Problema)

```
typedef struct dadosFila{
        char f[max];
        int inicio, fim;
}sFila;
```

- Criação:



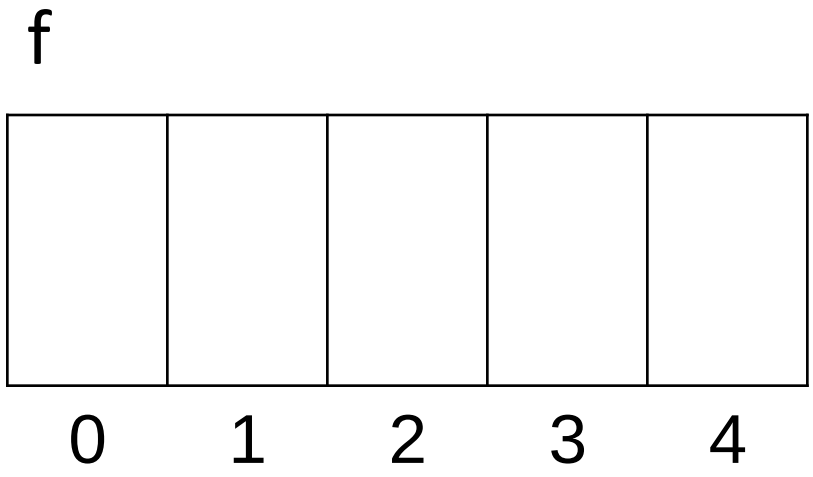
inicio :

fim :

Implementação - Fila Estática (Problema)

```
typedef struct dadosFila{
        char f[max];
        int inicio, fim;
}sFila;
```

- Criação:



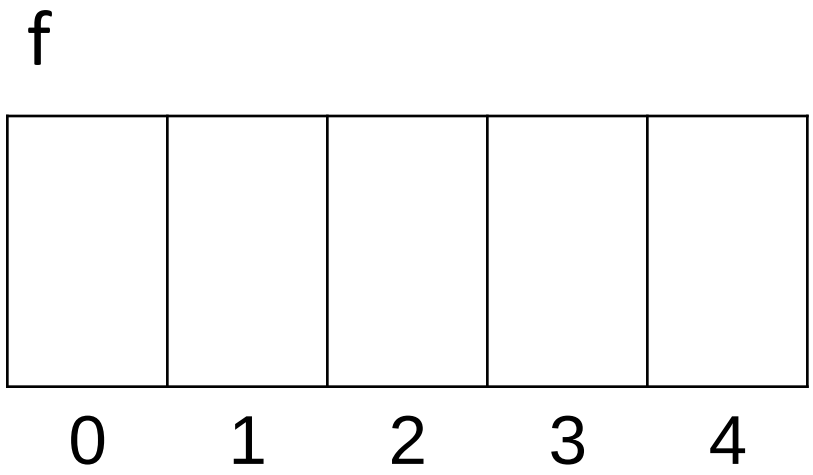
Campos do tipo inteiro para marcar posição início e fim

inicio : fim :

Implementação - Fila Estática (Problema)

```
typedef struct dadosFila{
        char f[max];
        int inicio, fim;
}sFila;
```

- Criação:



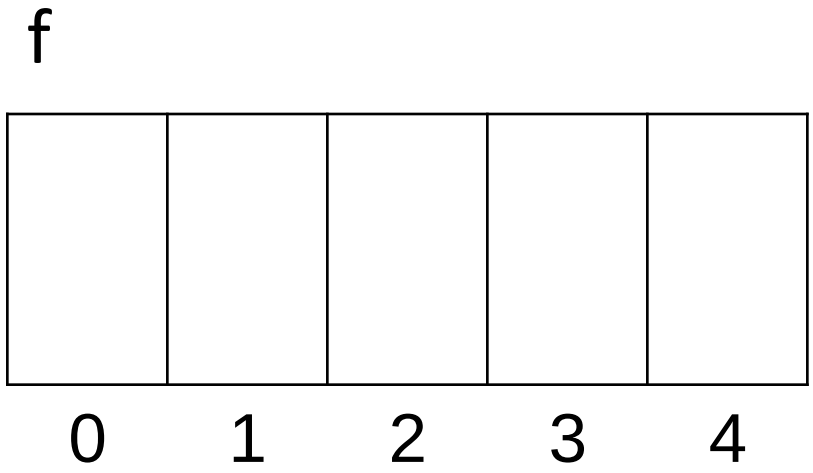
inicio : fim :

Variável do tipo sFila

```
int main(){
    sFila lista;
    ...
```

Implementação - Fila Estática - Inicialização

```
void inicializa(sFila *fila){
    fila->inicio = 0;
    fila->fim = 0;
}
```



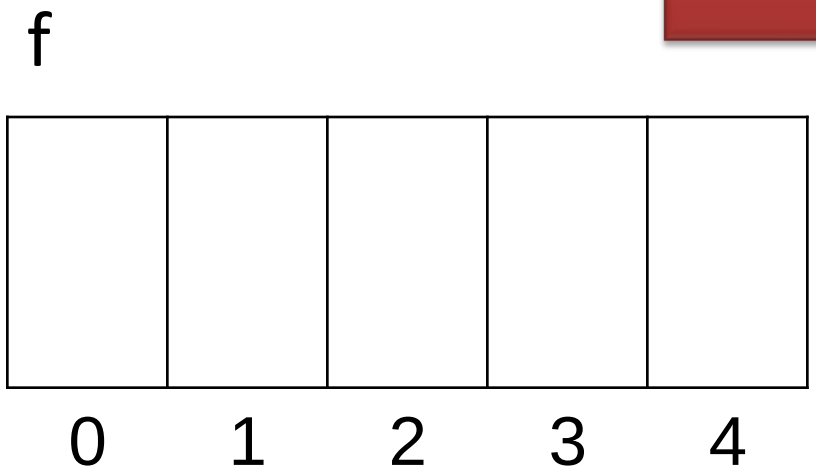
inicio : 0 fim : 0

Chamada da função:
inicializa(&lista);

Implementação - Fila Estática (Problema)

- Observações:

Dados Importantes:
inicio == fim ➔ a fila está vazia
fim == max (5) ➔ a fila está cheia

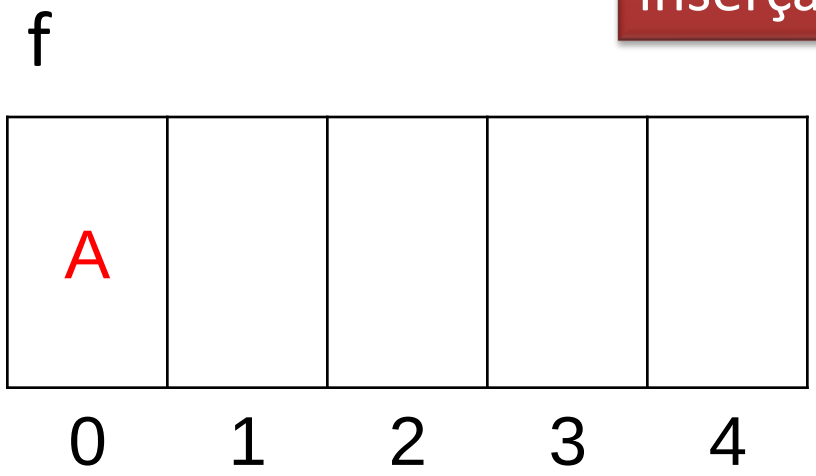


inicio : 0 fim : 0

Implementação - Fila Estática (Problema)

A entrada de dados é dada na posição indicada pela variável “fim”, cujo valor é incrementado em um após a inserção

- Cadastro:

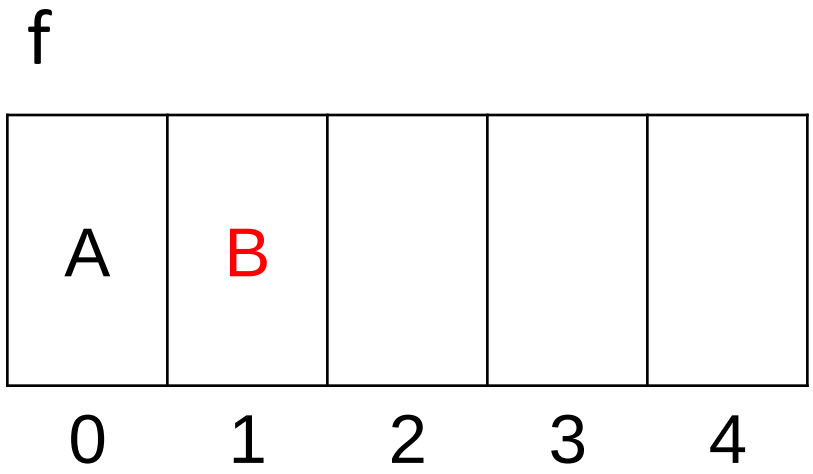


inicio : 0 fim : ~~0~~ 1

Implementação - Fila Estática (Problema)

- Cadastro:

Do segundo dado



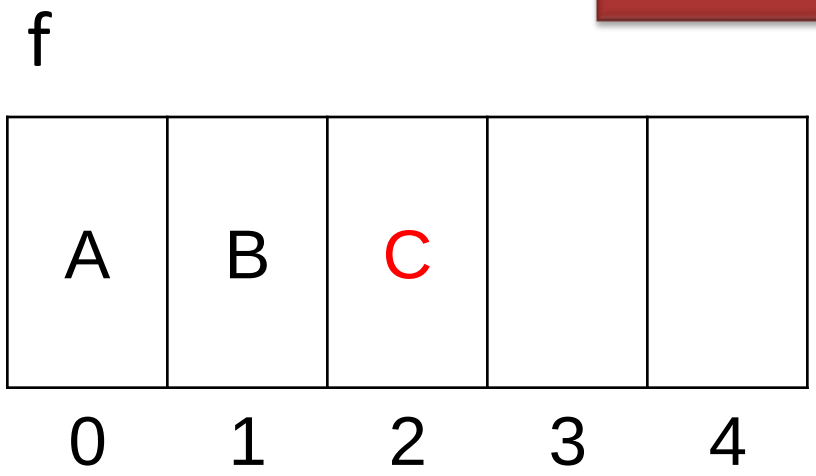
inicio : 0 fim : ~~0~~ ~~1~~ 2

Implementação - Fila Estática (Problema)

Do terceiro dado

A Fila não está vazia e nem cheia, pois, inicio != fim e fim < max

- Cadastro:

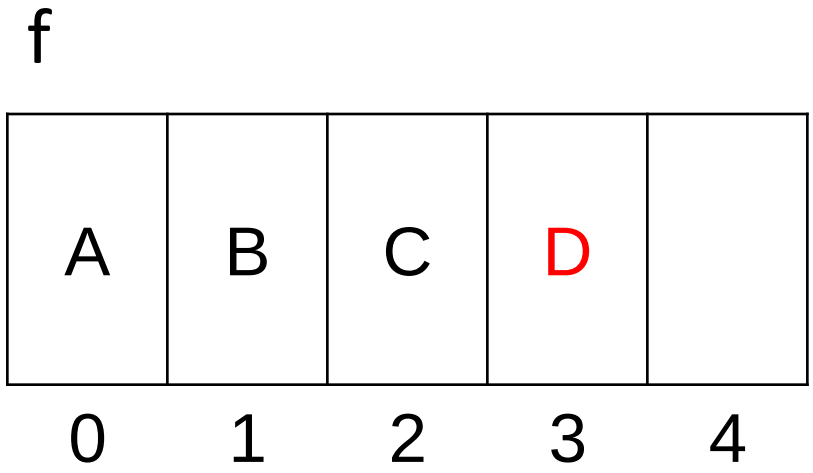


inicio : 0 fim : ~~0 1 2~~ 3

Implementação - Fila Estática (Problema)

- Cadastro:

Do quarto dado



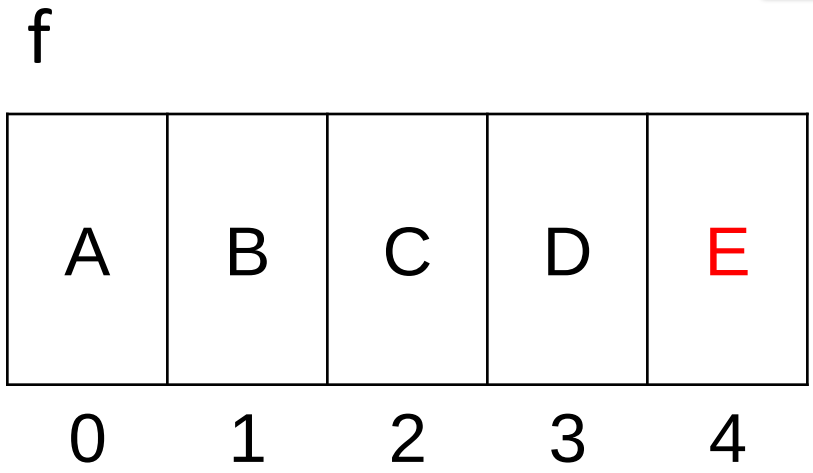
inicio : 0 fim : ~~0 1 2 3~~ 4

Implementação - Fila Estática (Problema)

- Cadastro:

Do quinto dado

A Fila está cheia, pois, fim == max

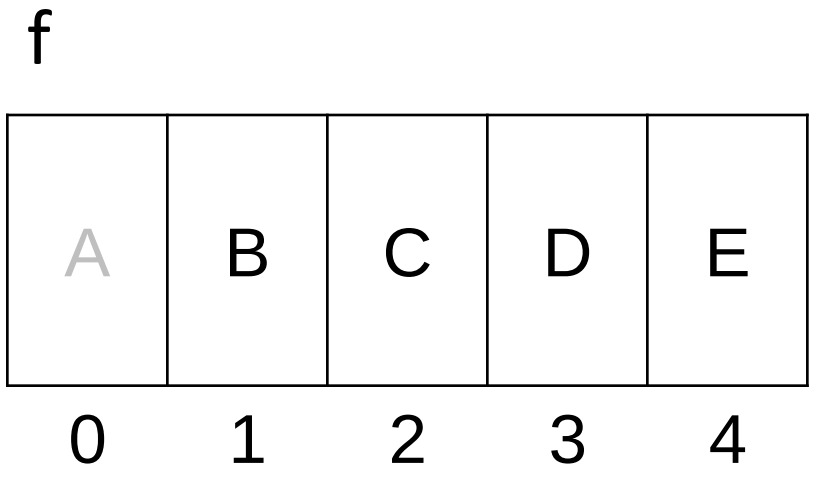


inicio : 0 fim : 0 1 2 3 4 5

Implementação - Fila Estática (Problema)

- Exclusão:

A exclusão de dados é dada na posição indicada pelo campo “inicio”, cujo valor é incrementado em um.



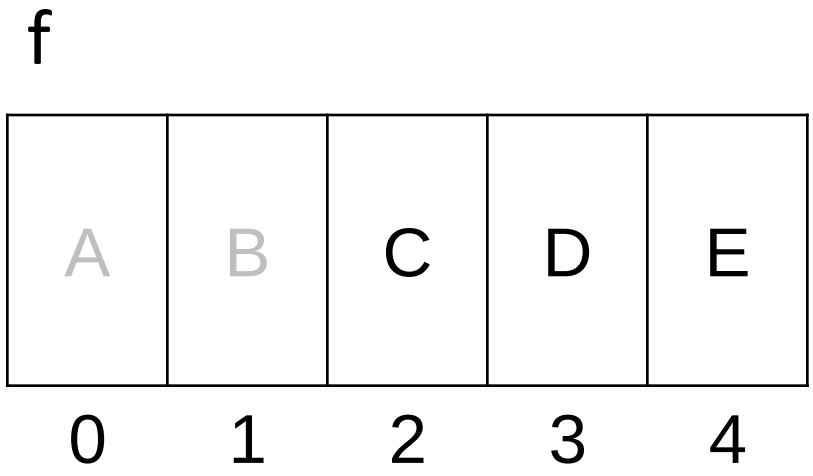
inicio : ~~0~~ 1 fim : 5

Problema: A Fila continua cheia, pois, fim == max, apesar de um elemento ter sido excluído e um espaço liberado.

Implementação - Fila Estática (Problema)

- Exclusão:

Do segundo dado

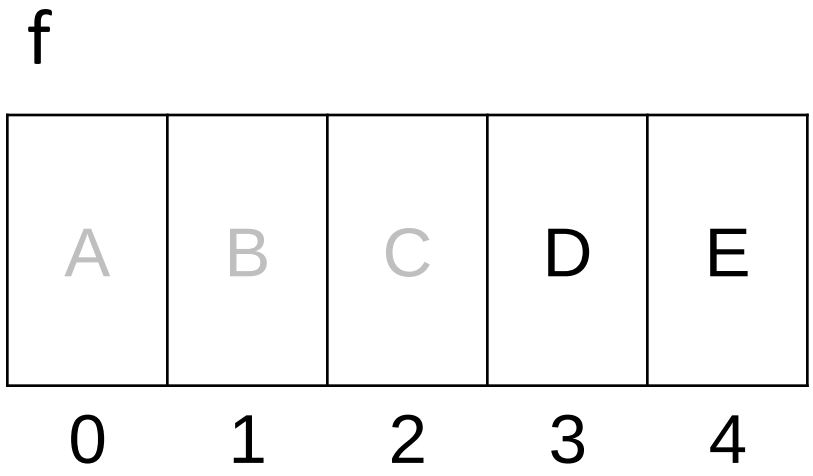


inicio : ~~0~~ ~~1~~ 2 fim : 5

Implementação - Fila Estática (Problema)

- Exclusão:

Do terceiro dado

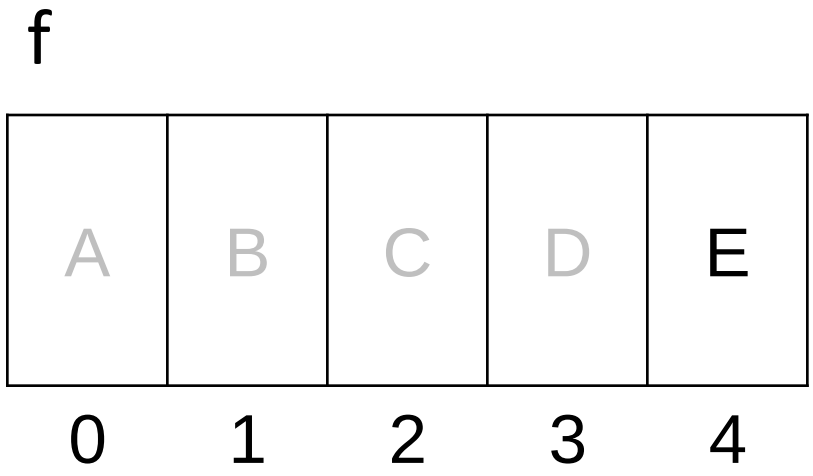


inicio : ~~0 1 2~~ 3 fim : 5

Implementação - Fila Estática (Problema)

- Exclusão:

Do quarto dado

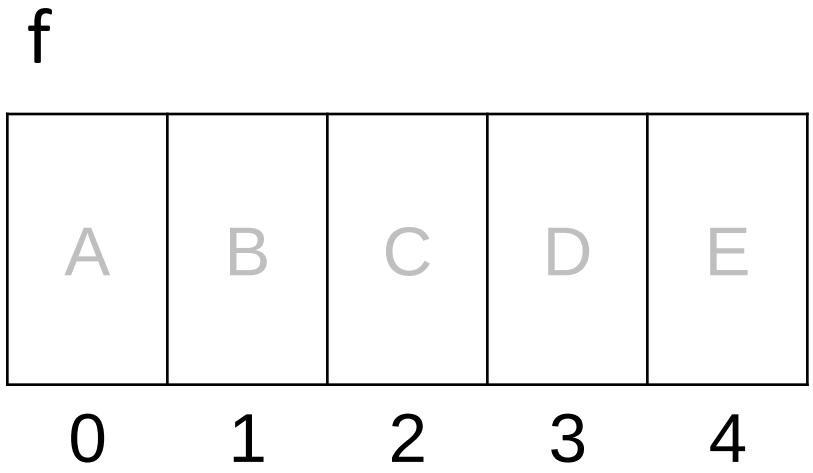


inicio : ~~0 1 2 3~~ 4 fim : 5

Implementação - Fila Estática (Problema)

- Exclusão:

Do quinto dado



inicio : ~~0 1 2 3 4~~ 5 fim : 5

Problema: A Fila está cheia ou vazia? Afinal, inicio == fim (fila vazia) e fim == max (fila cheia)

Implementação - Fila Estática - Soluções

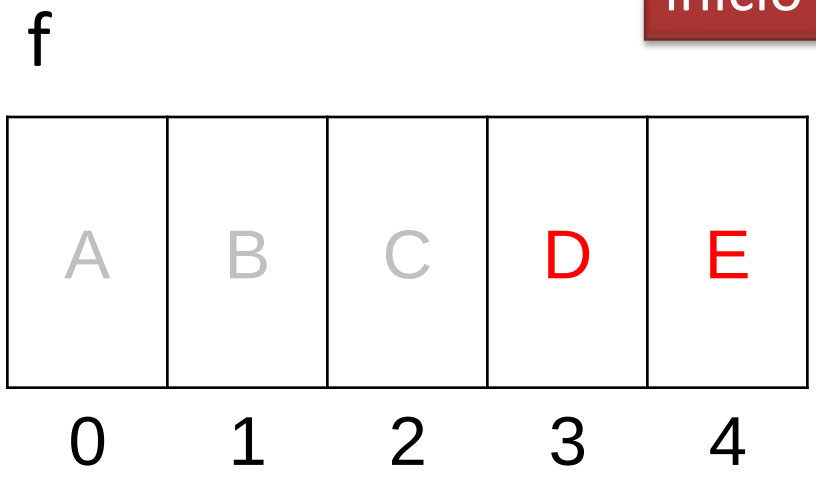
- Para resolver o problema exposto no slide anterior, pode-se utilizar alguns métodos:

1) O deslocamento dos dados para o início do vetor quando houvesse lugar vazio e a necessidade de novas inserções
2) Fila estática circular

Implementação - Fila Estática – Solução 1

- Primeira solução:

Tendo-se a fila neste estado, por exemplo, dois elementos no final do vetor e três espaços vazios no início

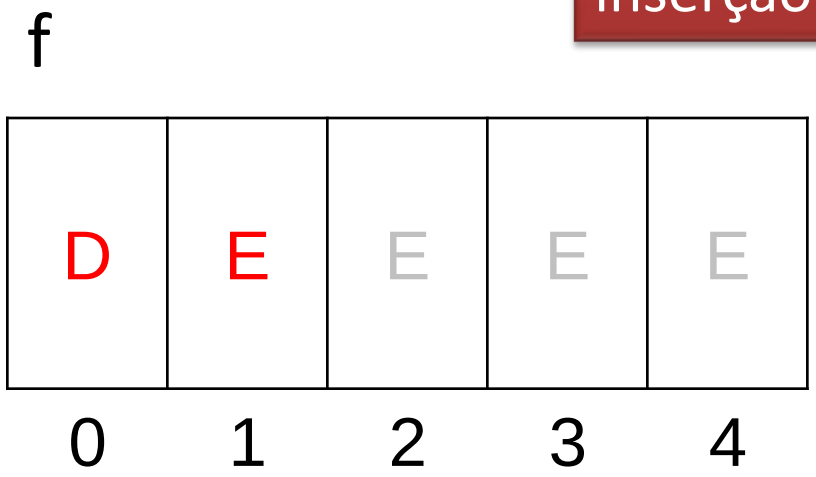


inicio : 3 fim : 5

Implementação - Fila Estática – Solução 1

- Primeira solução:

Os dados são deslocados para o início do vetor, obtendo-se uma fila não cheia (fim < max), podendo assim ser feita nova inserção.



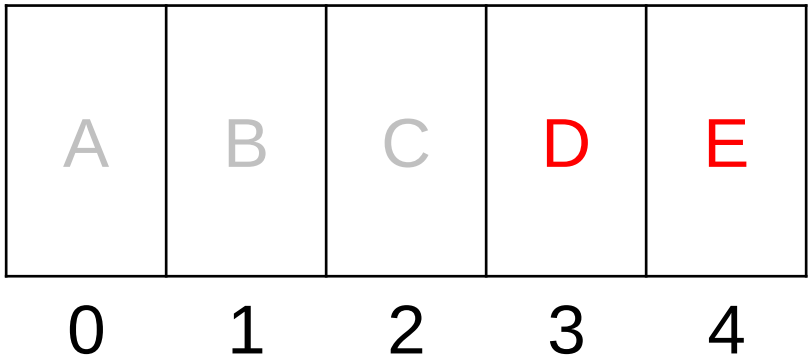
inicio : 0

fim : 2

Implementação - Fila Estática – Solução 2

- Segunda solução:
  - Fila Estática Circular

f



Mais um campo é acrescentado na estrutura: “total”

Onde:

Fila cheia ➔ total = 5

Fila vazia ➔ total = 0

Inicialização:

total é inicializado com 0

No exemplo:

Aqui, com dois dados, total = 2

inicio : 3 fim : 5 total: 2

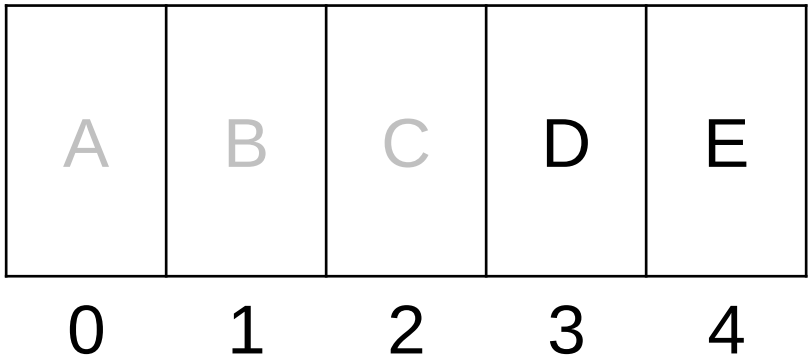
Implementação - Fila Estática – Solução 2

- Segunda solução:
  - Fila Estática Circular

Toda vez que a variável “fim” ou “inicio” igualar a “max” (5), elas recebem o valor 0.

No exemplo deste slide, “fim” passa a receber 0 porque tinha o valor 5 que é igual a “max” (5)

f

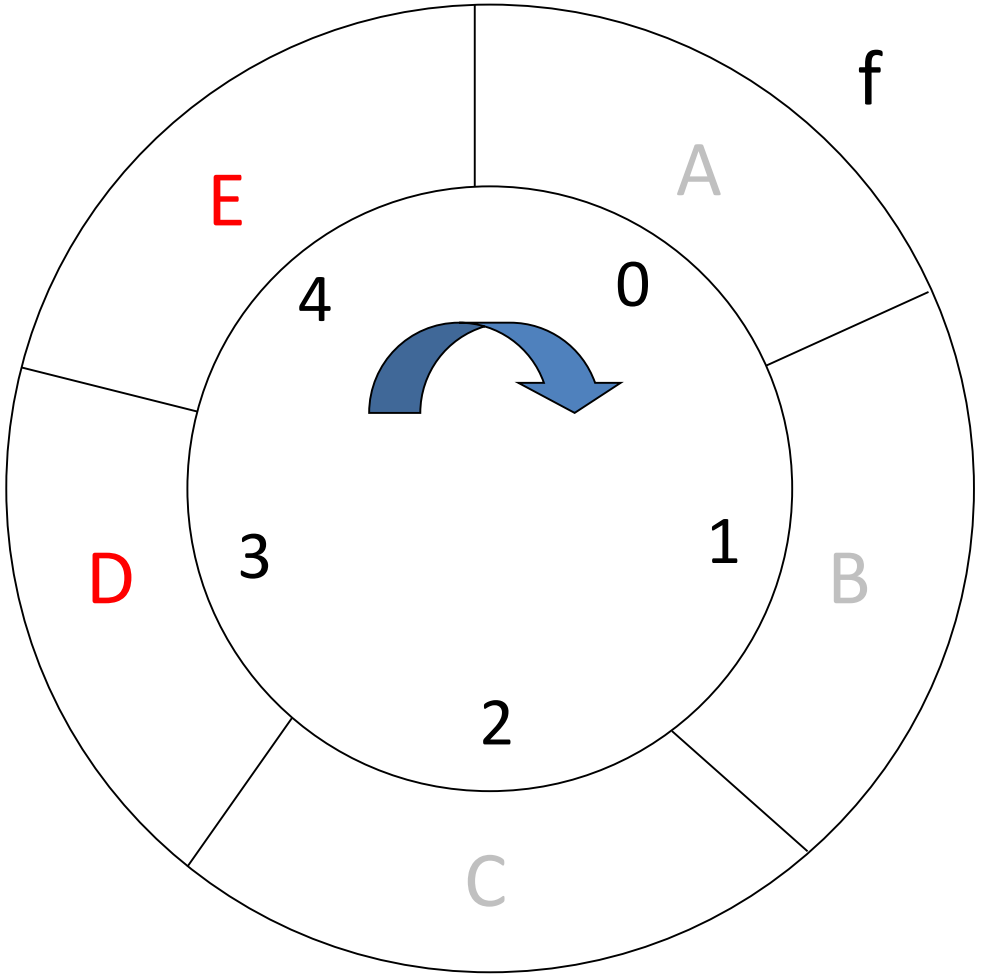


inicio : 3 fim : ~~5~~ 0 total: 2

Implementação - Fila Estática – Solução 2

- Segunda solução:
  - Fila Estática Circular

Outra forma de representar uma fila estática circular. Ela continua sendo um vetor, só mudou a forma de visualização.

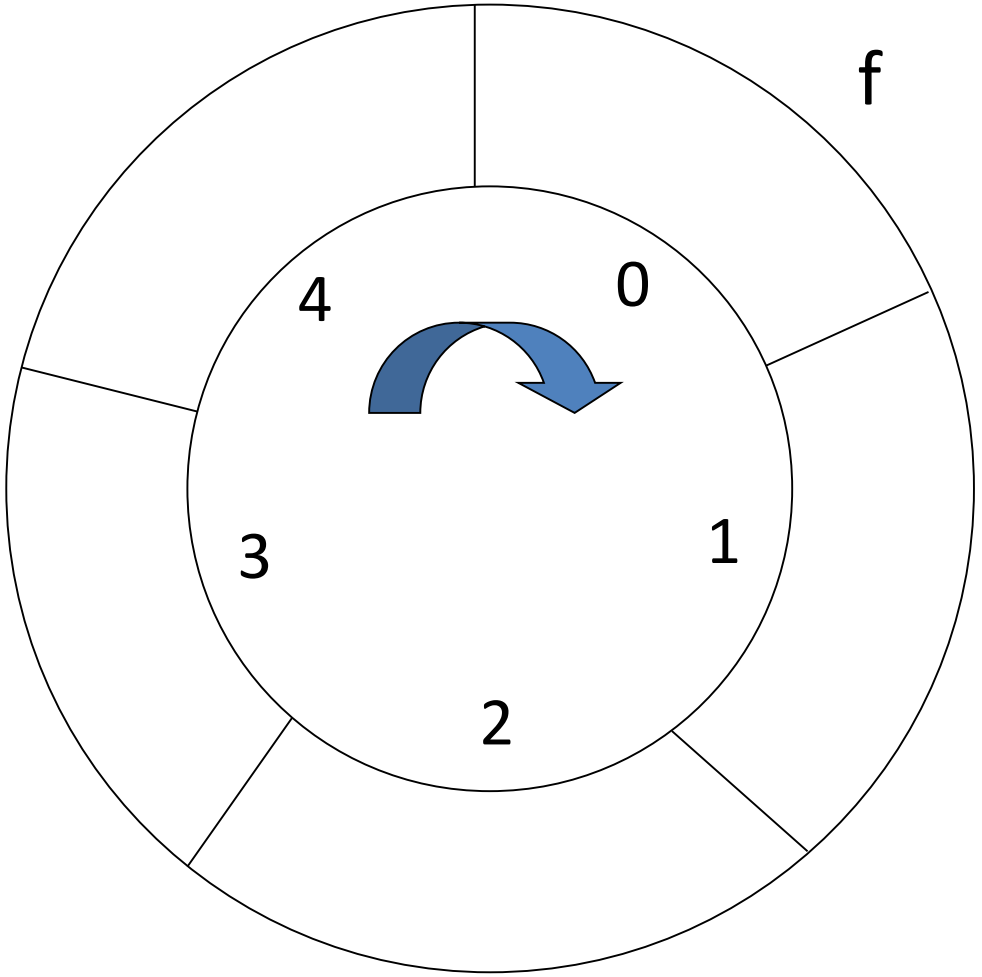


inicio : 3 fim : 0 total: 2

Implementação - Fila Estática Circular

- Inicialização:

```
void inicializa(sFila *fila){
    fila->inicio = 0;
    fila->fim = 0;
    fila->total = 0;
}
```



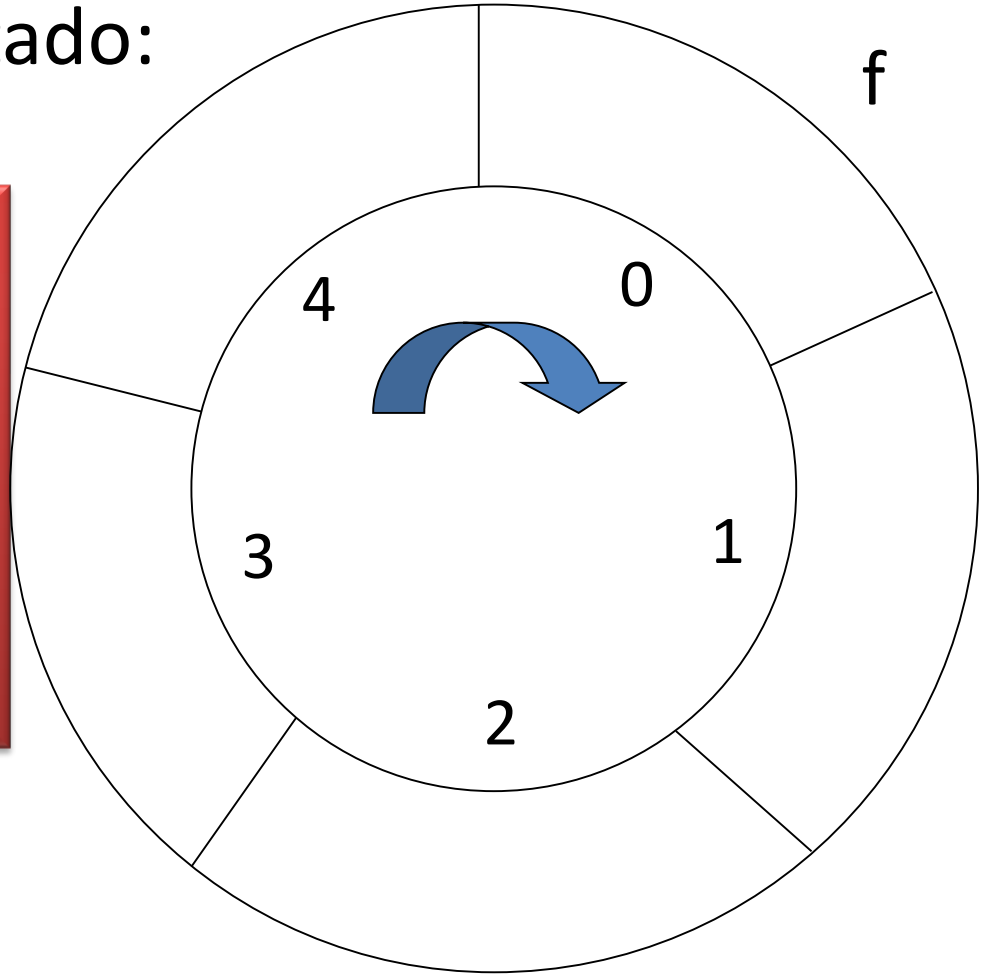
inicio : 0 fim : 0 total: 0

Implementação - Fila Estática Circular

- Verificações do seu estado:

```
int estaVazia(sFila *fila){
    return (fila->total == 0?1:0);
}

int estaCheia(sFila *fila){
    return (fila->total == max?1:0);
}
```

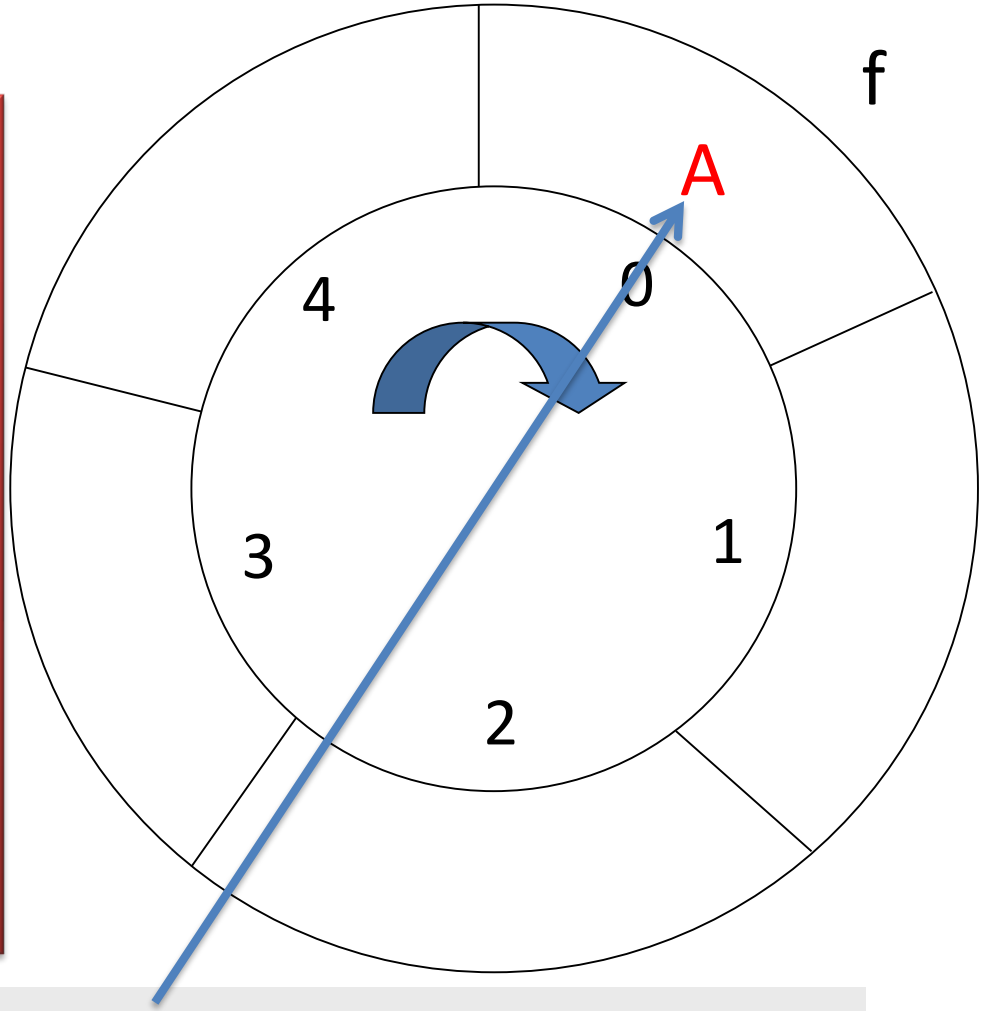


inicio : 0 fim : 0 total: 0

Implementação - Fila Estática Circular

- Inserção:

```
void inserir(sFila *fila, char dado){
    if (estaCheia(fila))
        printf("Fila  Cheia!");
    else{
        fila->f[fila->fim] = toupper(dado);
        fila->fim++;
        fila->total++;
        if (fila->fim == max)
            fila->fim = 0;
        printf("Inserido com sucesso!");
    }
    system("pause");
}
```

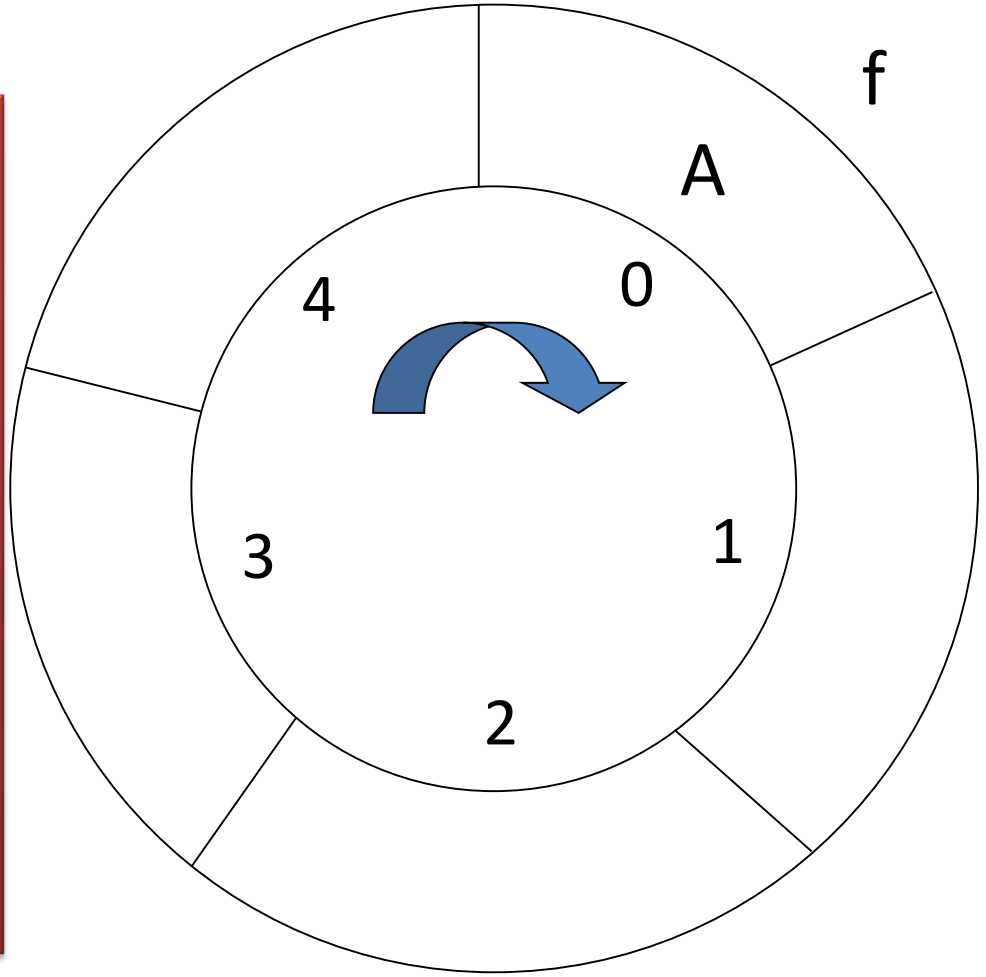


inicio : 0 fim : 0 total: 0

Implementação - Fila Estática Circular

- Inserção:

```
void inserir(sFila *fila, char dado){
    if (estaCheia(fila))
        printf("Fila  Cheia!");
    else{
        fila->f[fila->fim] = toupper(dado);
        fila->fim++;
        fila->total++;
        if (fila->fim == max)
            fila->fim = 0;
        printf("Inserido com sucesso!");
    }
    system("pause");
}
```

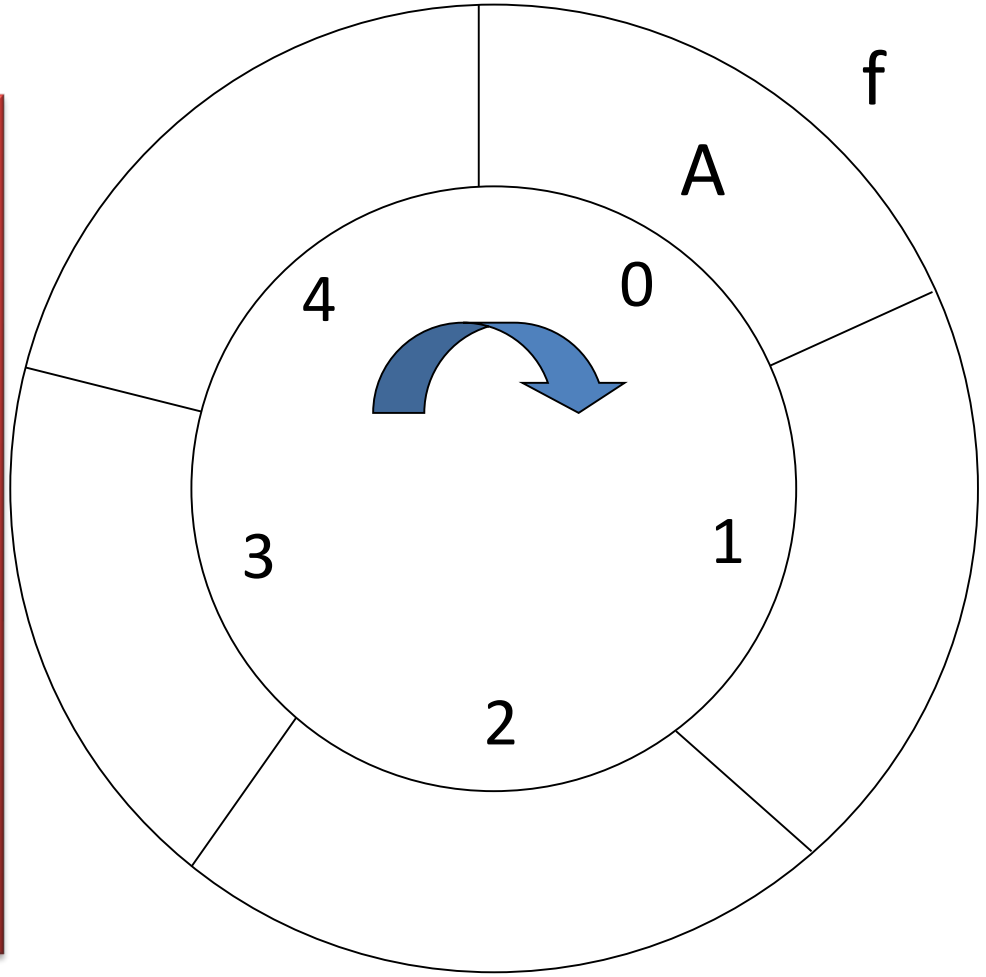


inicio : 0 fim : ~~0~~ 1 total: 0

Implementação - Fila Estática Circular

- Inserção:

```
void inserir(sFila *fila, char dado){
    if (estaCheia(fila))
        printf("Fila  Cheia!");
    else{
        fila->f[fila->fim] = toupper(dado);
        fila->fim++;
        fila->total++;
        if (fila->fim == max)
            fila->fim = 0;
        printf("Inserido com sucesso!");
    }
    system("pause");
}
```

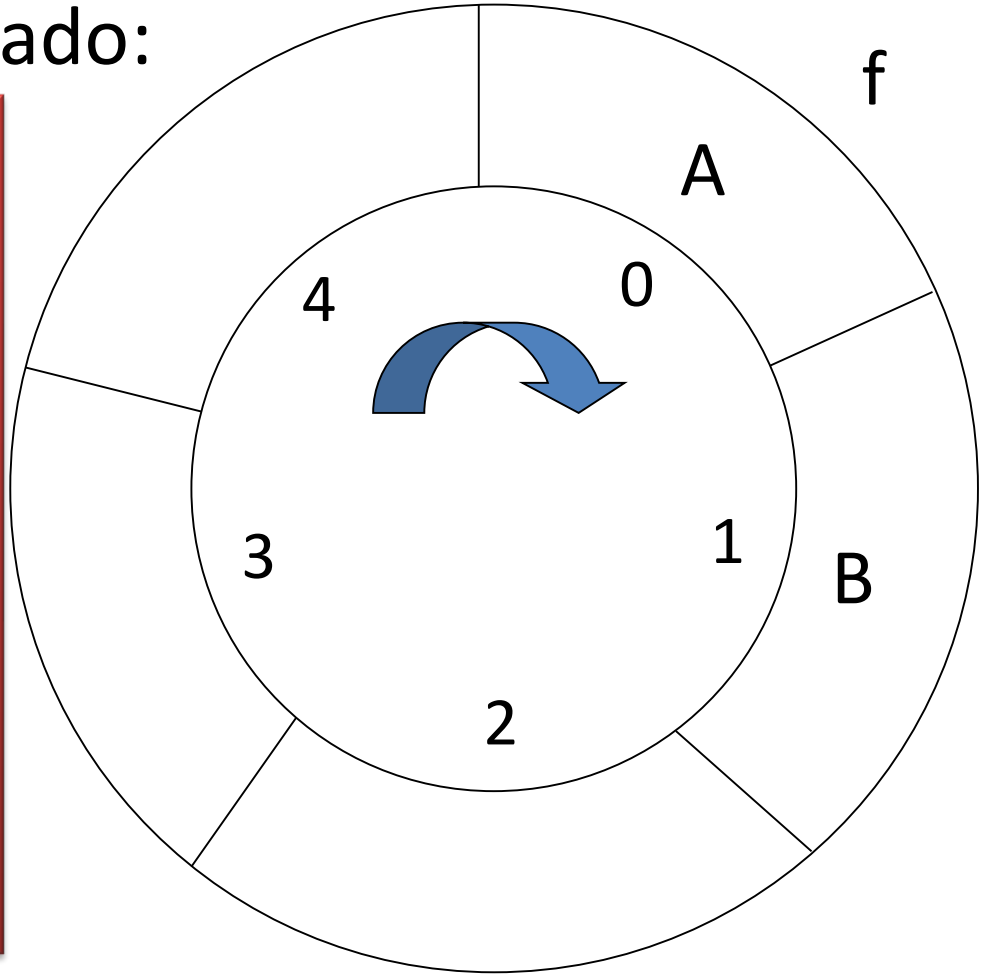


inicio : 0 fim : 1 total: ~~0~~ 1

Implementação - Fila Estática Circular

- Inserção do segundo dado:

```
void inserir(sFila *fila, char dado){
    if (estaCheia(fila))
        printf("Fila  Cheia!");
    else{
        fila->f[fila->fim] = toupper(dado);
        fila->fim++;
        fila->total++;
        if (fila->fim == max)
            fila->fim = 0;
        printf("Inserido com sucesso!");
    }
    system("pause");
}
```



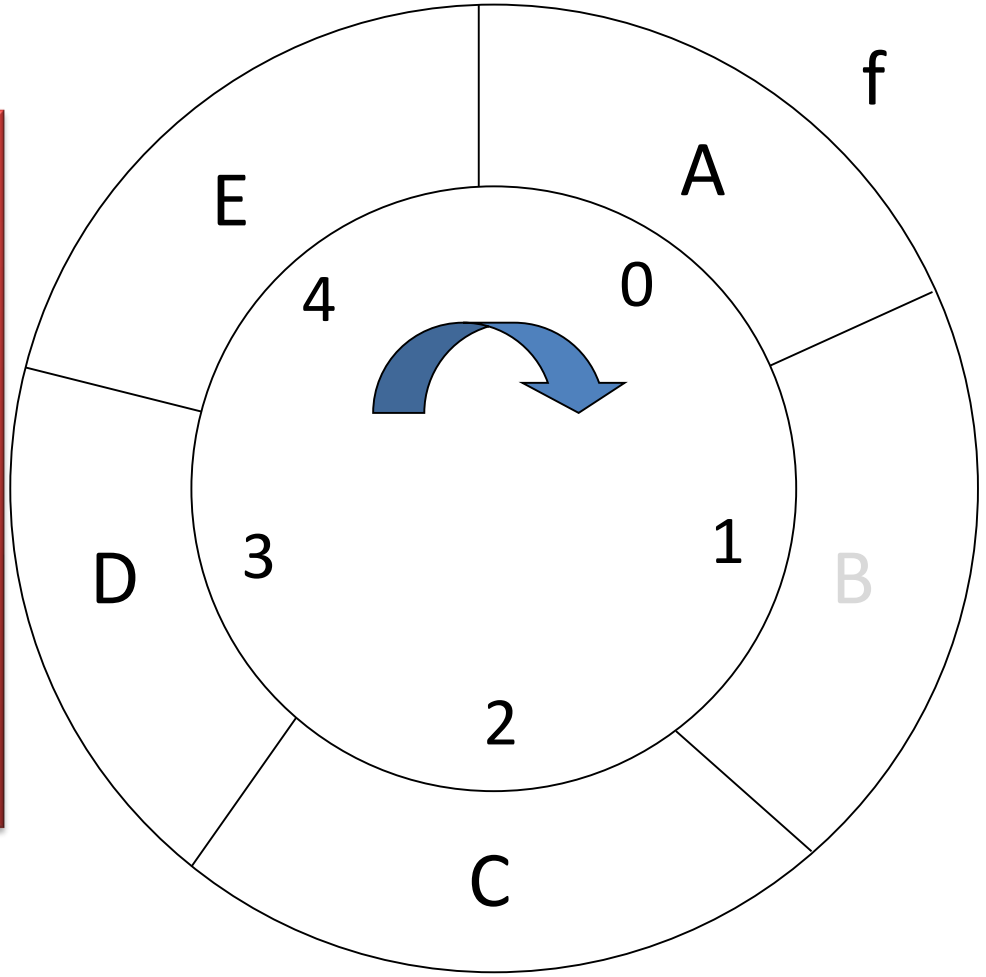
inicio : 0 fim : ~~1~~ 2 total: ~~0~~ ~~1~~ 2

Implementação - Fila Estática Circular

- Listagem:

```
void listar(sFila *fila){
  if (estaVazia(fila))
    printf("\nFila Vazia!\n\n");
  else{
    printf(“inicio : %c\n",
                        fila->f[fila->inicio]);
    printf("fim : %c",fila->f[fila->fim-1]);
  }
}
```

Imprimirá:
inicio : C
fim : A



inicio : 2 fim : 1 total: 4

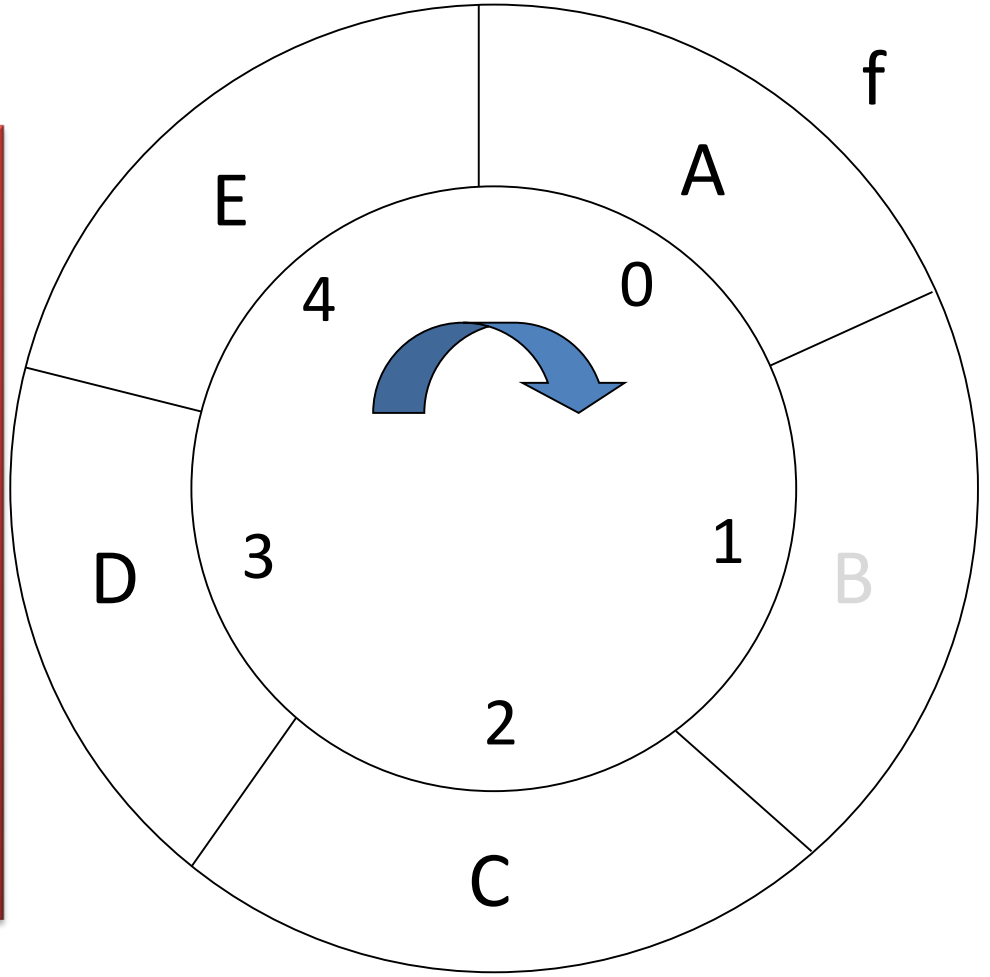
Implementação - Fila Estática Circular

- Exclusão:

Só poderá ser feita exclusão do elemento que se encontra na posição “inicio”.

No caso da fila estática circular, ao lado, poderá ser excluído o “C”.

Este campo é incrementado em um e se após o incremento ficar com valor igual a “max”, ele receberá 0 (zero).

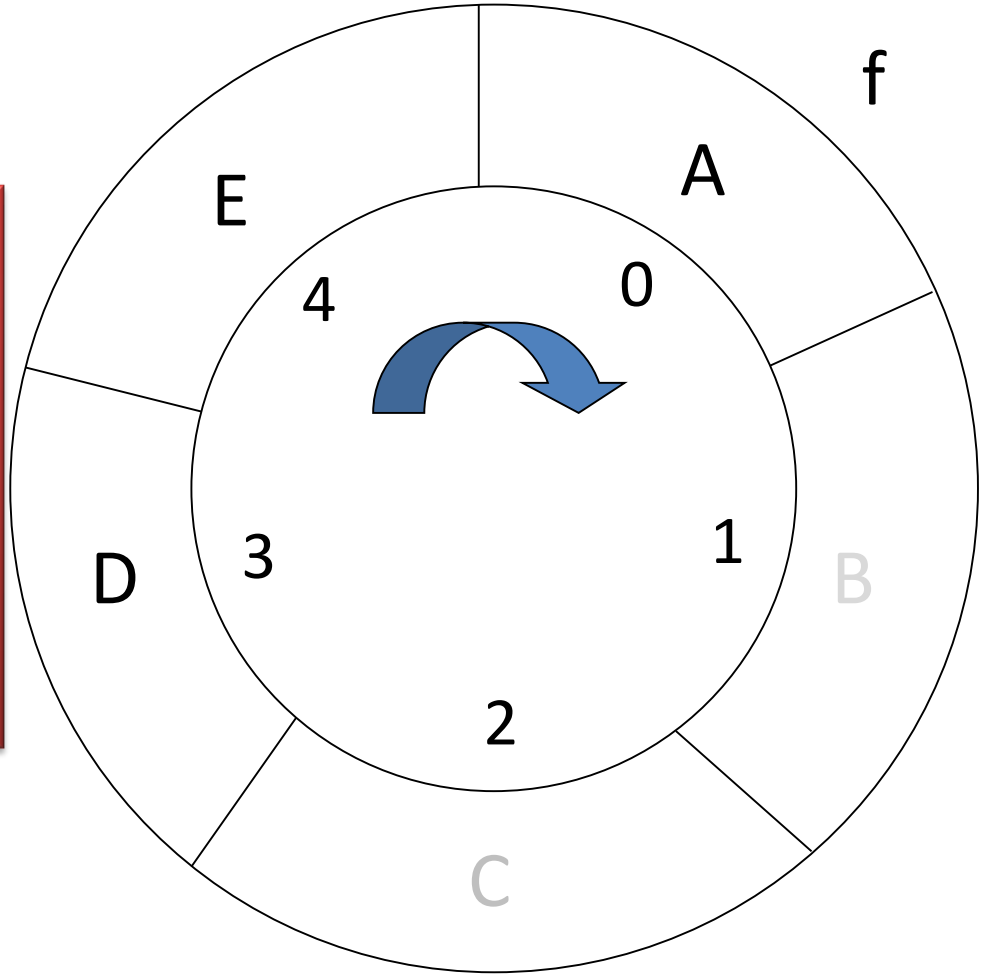


inicio : 2 fim : 1 total: 4

Implementação - Fila Estática Circular

- Exclusão:

```
void excluir(sFila *fila){
      fila->inicio++;
      if (fila->inicio == max)
          fila->inicio = 0;
      fila->total--;
      printf("Dado excluido");
}
```



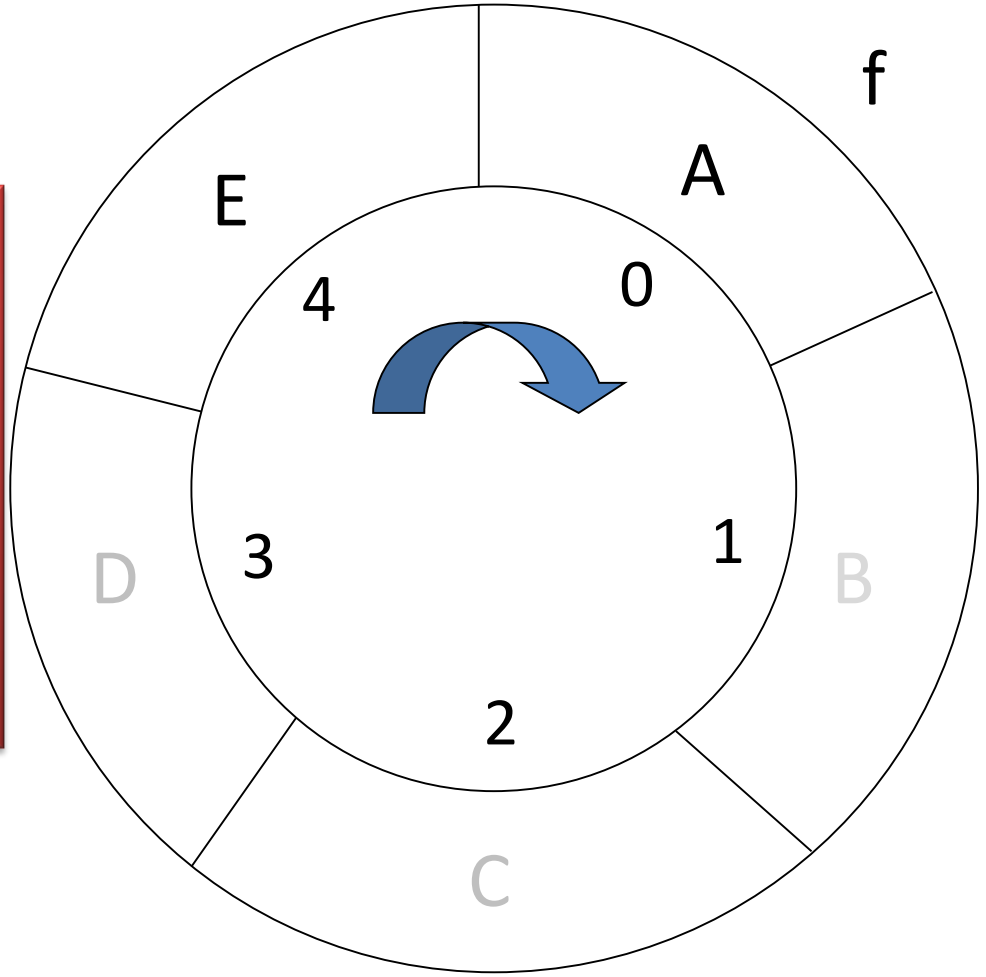
Do primeiro dado

inicio : ~~2~~ 3 fim : 1 total: ~~4~~ 3

Implementação - Fila Estática Circular

- Exclusão:

```
void excluir(sFila *fila){
      fila->inicio++;
      if (fila->inicio == max)
          fila->inicio = 0;
      fila->total--;
      printf("Dado excluido");
}
```



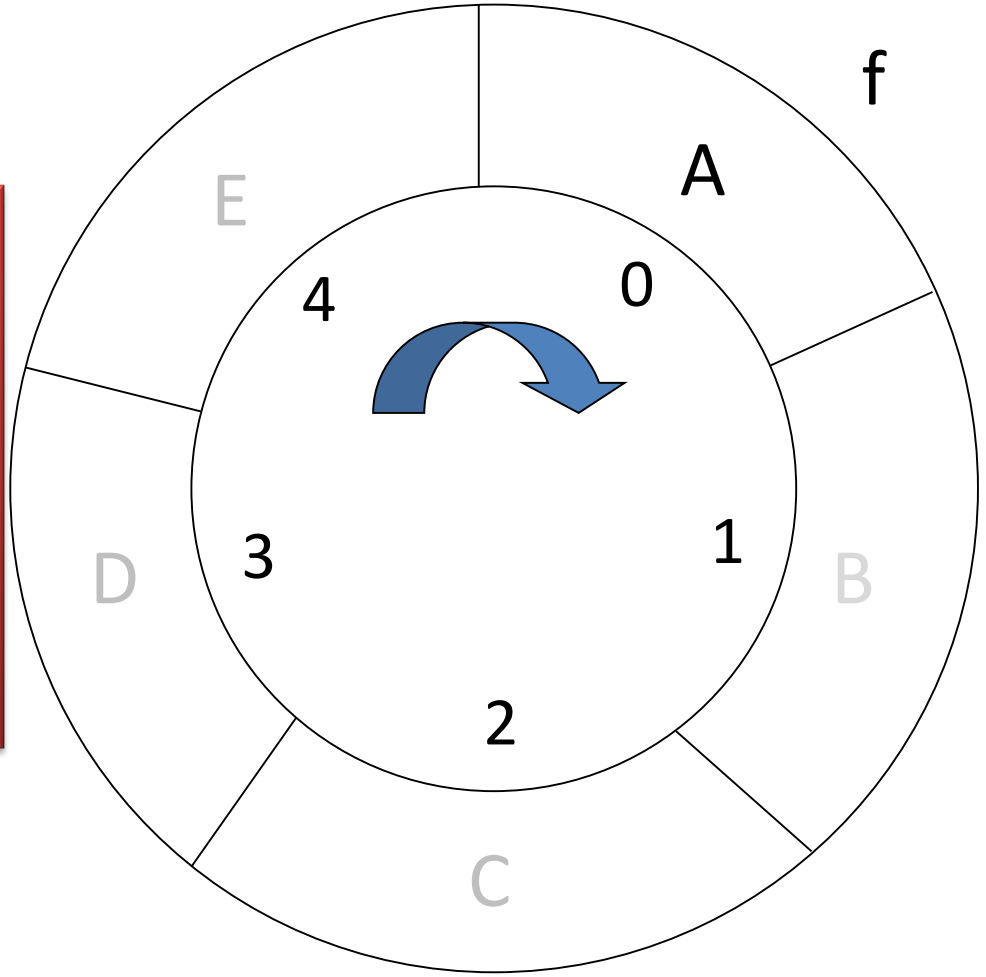
Do segundo dado

inicio : ~~2 3~~ 4 fim : 1 total: ~~4 3~~ 2

Implementação - Fila Estática Circular

- Exclusão:

```
void excluir(sFila *fila){
      fila->inicio++;
      if (fila->inicio == max)
          fila->inicio = 0;
      fila->total--;
      printf("Dado excluido");
}
```



Do terceiro dado

inicio : ~~2 3 4 5~~ 0 fim : 1 total: ~~4 3 2~~ 1

Observe que o valor do campo "inicio" ficou igual a "max" (5), logo, ele receberá o valor 0 (zero)

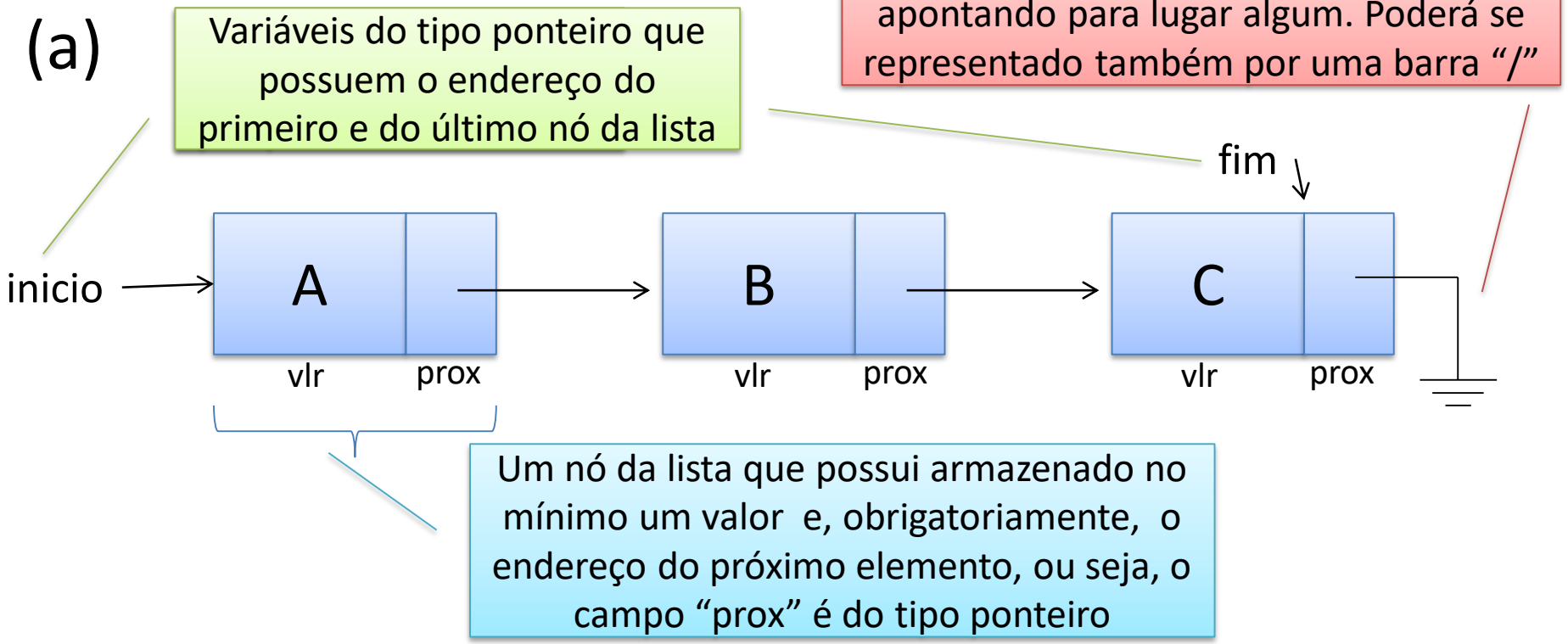
- Fila Dinâmica
  - **Vantagem**: não é necessário saber a quantidade de dados que serão armazenados pois criam-se os espaços necessários dinamicamente, ou seja, em momento de execução. Ocupa espaço estritamente necessário.
  - **Desvantagem**: custos usuais da alocação dinâmica (tempo de alocação, campos de ligação)

Quando usar?

- Fila Estática Circular
  - quando a fila tiver tamanho pequeno ou seu comportamento for previsível
- Fila Dinâmica
  - Nos demais casos

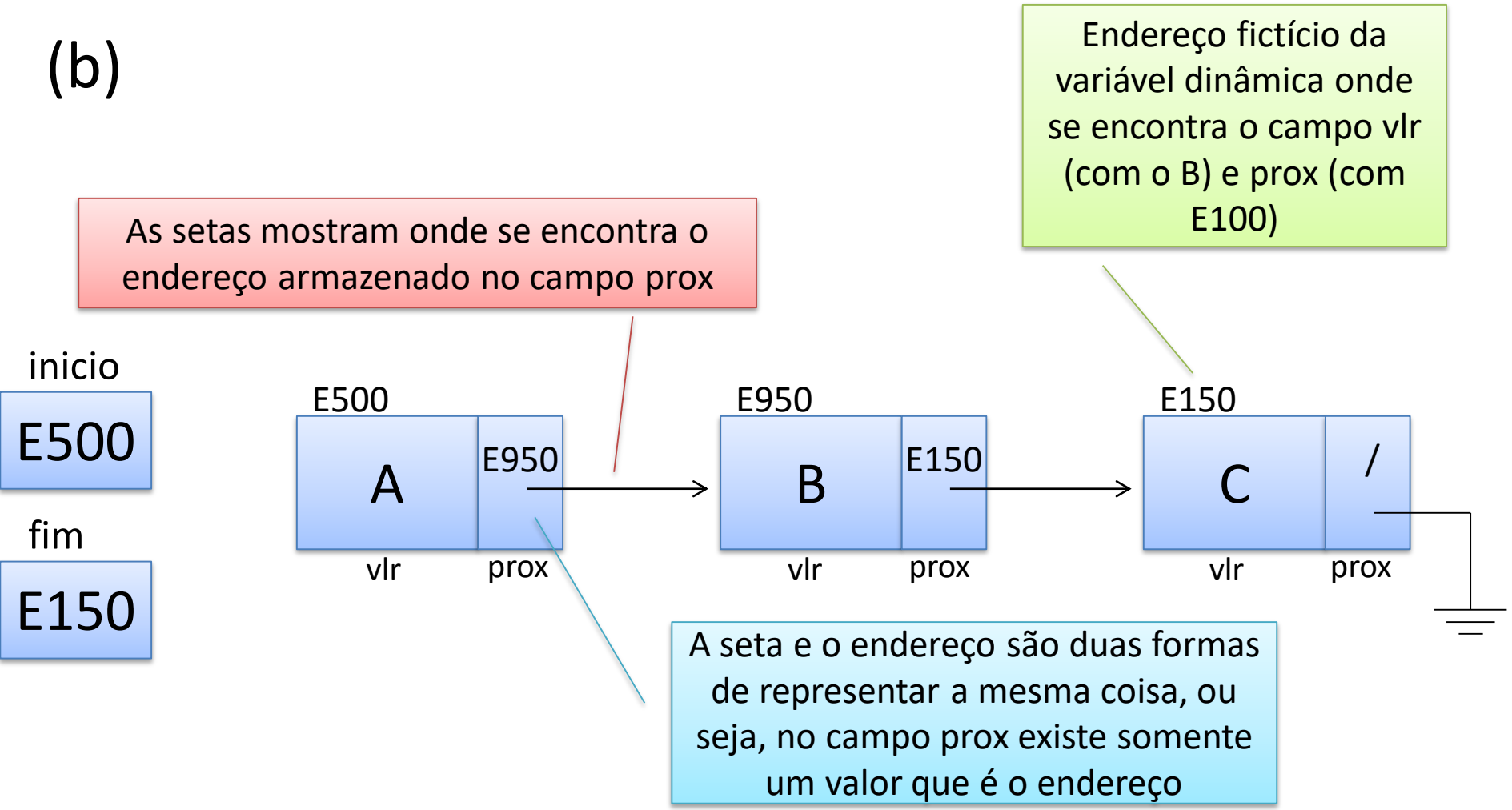
- Um nó é um espaço da lista destinado a armazenar algum valor
- A Fila é normalmente representada pelo desenho abaixo:

(a)



- Variações na representação:

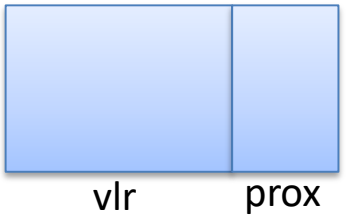
(b)



Implementação – FILA - Criação

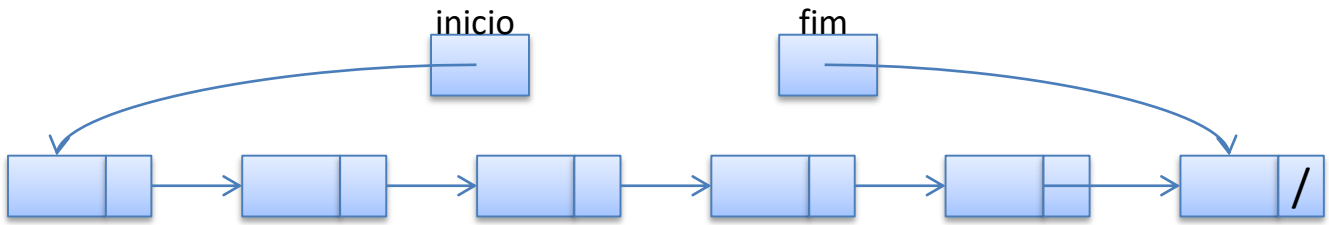
- A estrutura necessária para a criação do nó da fila é:

```
typedef struct noFila{
        char vlr;
        struct noFila *prox;
}sNoFila;
```



- É criada outra estrutura com dois campos que receberão os endereços do nó inicial e final da fila dinâmica.

```
typedef struct Fila{
        sNoFila *inicio,
                *fim;
}sFila;
```



Implementação – FILA - Criação

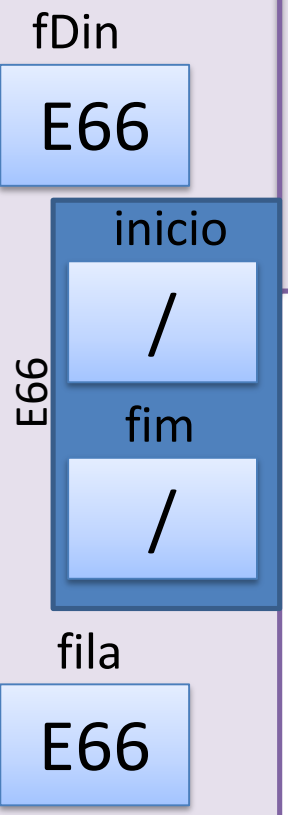
- Para criar a fila é declarada uma variável do tipo ponteiro desta segunda estrutura, que devem ter seus campos inicializados com NULL:

```
sFila *fDin;
fDin = (sFila *) malloc (sizeof(sFila));
if (fDin == NULL)
        exit (1);
inicializa(fDin);
```

```
void inicializa(sFila *fila){
      fila->inicio = NULL;
      fila->fim = NULL;
}
```

Implementação – FILA – Vazia?

Visualização Gráfica

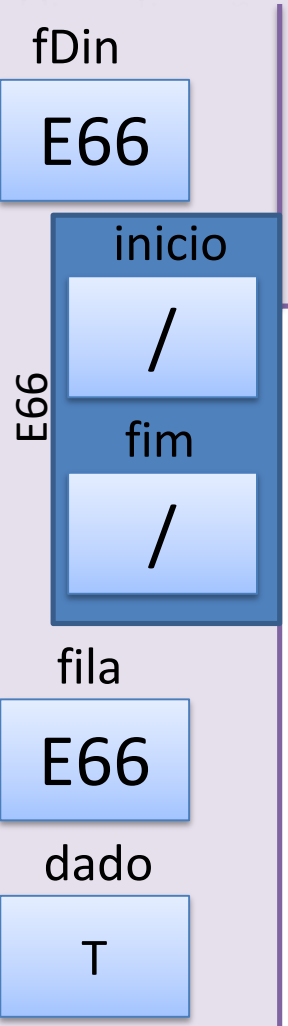


- Quando a fila está vazia, a variável “inicio” do tipo ponteiro terá NULL armazenado dentro dela. A função que verifica esta condição da fila é:

```
int estaVazia(sFila *fila){
     return (fila->inicio == NULL?1:0);
}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica

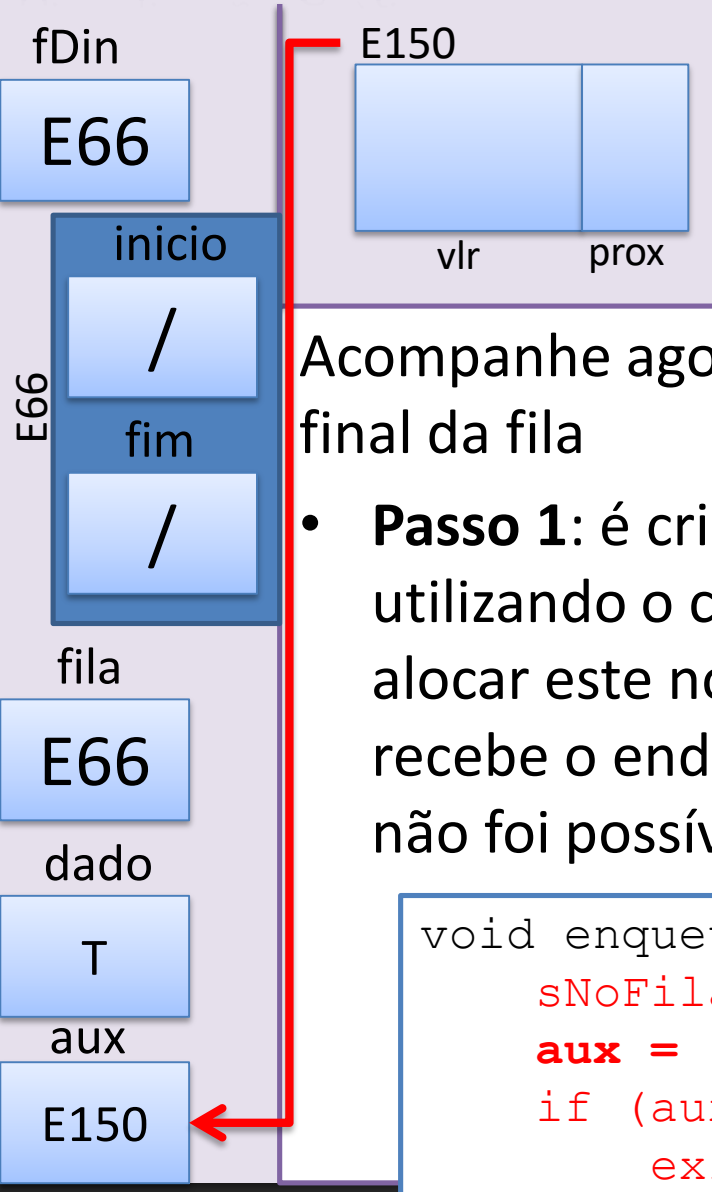


- Para a função que faz somente a inserção de um elemento na fila, são passados dois parâmetros:
  - a estrutura que controla o início e fim da fila (lista)
  - o dado a ser inserido
- O dado poderá ser inserido somente no final da fila.
- Veremos a seguir a forma como são inseridos os valores na fila dinâmica:

```
void enqueue(sFila *fila, char dado){...}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica



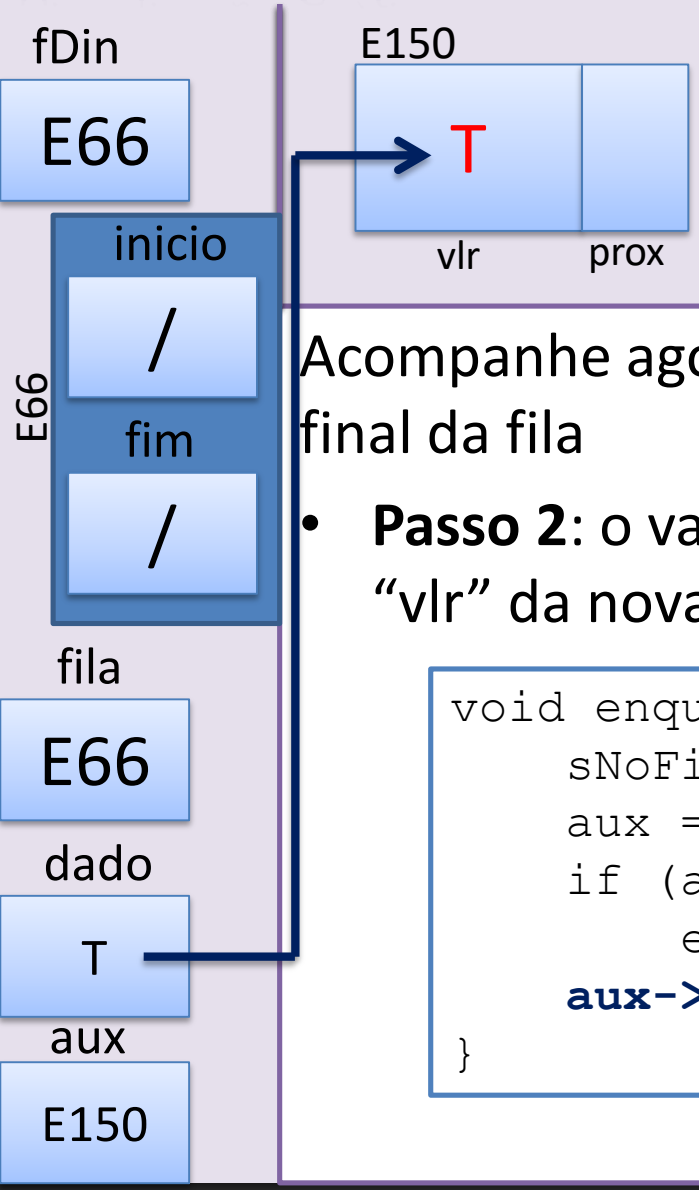
Acompanhe agora os passos para inserir um elemento no final da fila

- **Passo 1**: é criado um novo nó (variável dinâmica) , utilizando o comando malloc, e verificado se foi possível alocar este novo espaço na memória, ou seja, se aux, que recebe o endereço do novo espaço, tiver NULL é porque não foi possível.

```
void enqueue(sFila *fila, char dado){
     sNoFila *aux;
     aux = (sNoFila *) malloc (sizeof(sNoFila));
     if (aux == NULL)
          exit (1);
}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica



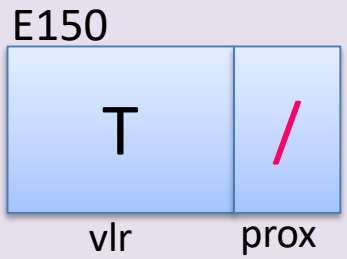
Acompanhe agora os passos para inserir um elemento no final da fila

- **Passo 2**: o valor da variável "dado" é inserido no campo "vlr" da nova variável dinâmica.

```
void enqueue(sFila *fila, char dado){
     sNoFila *aux;
     aux = (sNoFila *) malloc (sizeof(sNoFila));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica



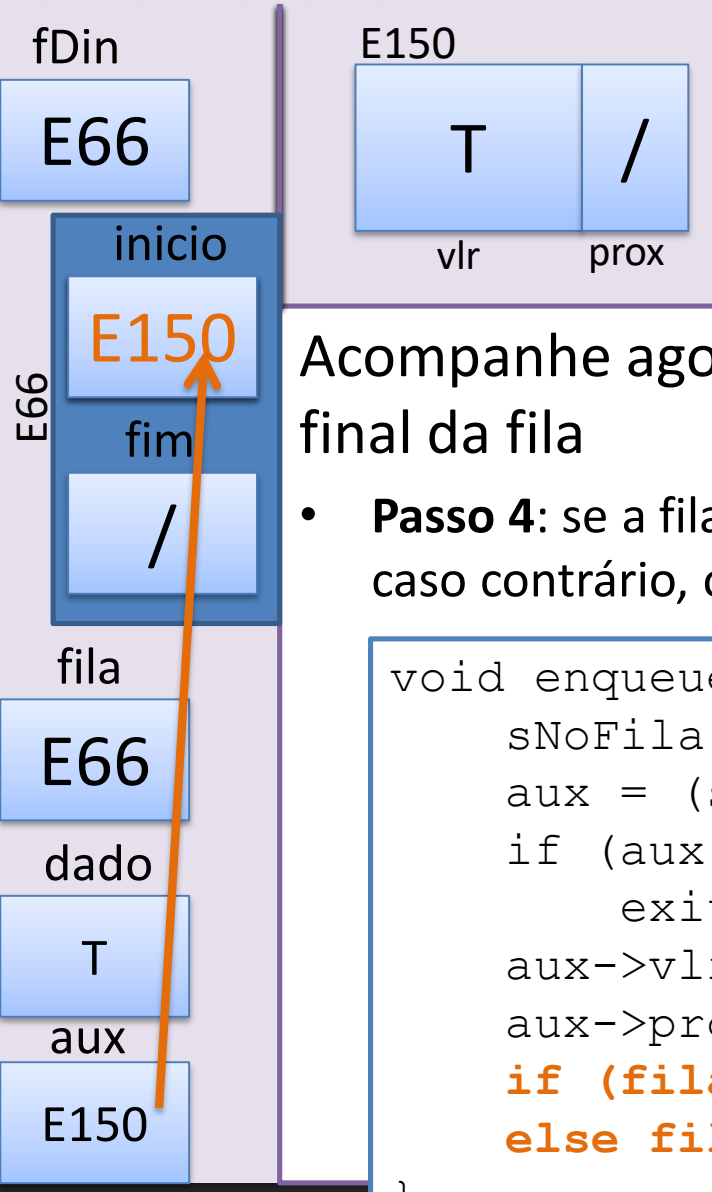
Acompanhe agora os passos para inserir um elemento no final da fila

- **Passo 3**: como este novo dado será o último da fila, o campo "prox" receberá NULL

```
void enqueue(sFila *fila, char dado){
     sNoFila *aux;
     aux = (sNoFila *) malloc (sizeof(sNoFila));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica



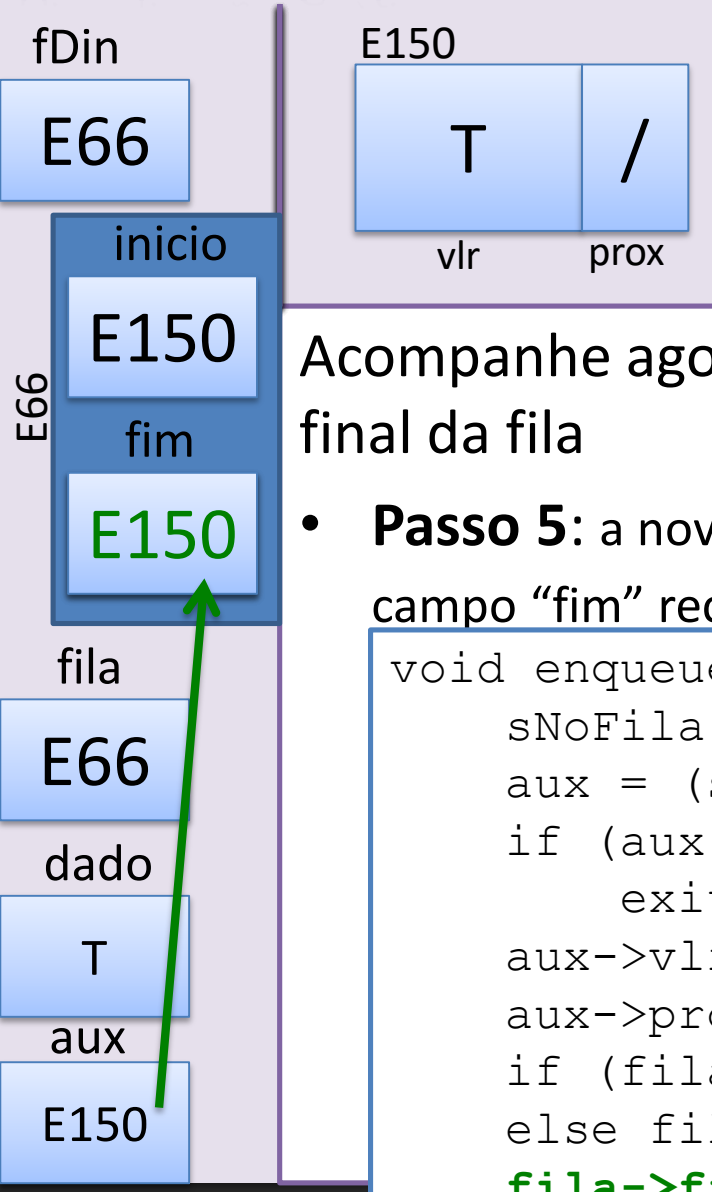
Acompanhe agora os passos para inserir um elemento no final da fila

- **Passo 4**: se a fila estiver vazia, "inicio" receberá o endereço de "aux", caso contrário, o campo "prox" apontado por "fim" receberá "aux"

```
void enqueue(sFila *fila, char dado){
     sNoFila *aux;
     aux = (sNoFila *) malloc (sizeof(sNoFila));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
     if (fila->inicio == NULL) fila->inicio = aux;
     else fila->fim->prox = aux;
}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica



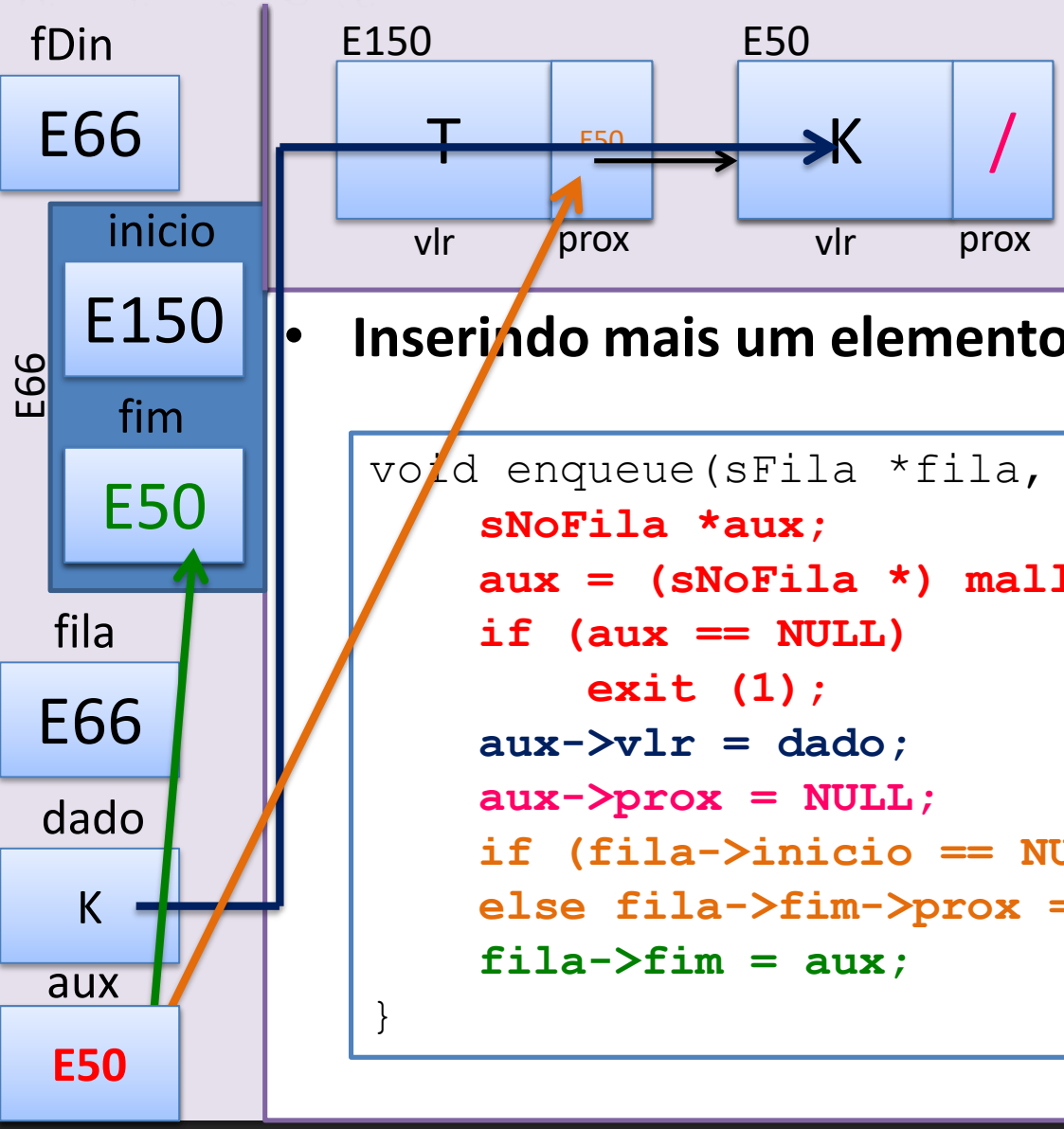
Acompanhe agora os passos para inserir um elemento no final da fila

- **Passo 5**: a nova variável dinâmica passa a ser o final da fila, logo, o campo "fim" recebe "aux"

```
void enqueue(sFila *fila, char dado){
     sNoFila *aux;
     aux = (sNoFila *) malloc (sizeof(sNoFila));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
     if (fila->inicio == NULL) fila->inicio = aux;
     else fila->fim->prox = aux;
     fila->fim = aux;
}
```

Implementação – FILA – Inserir (enqueue)

Visualização Gráfica

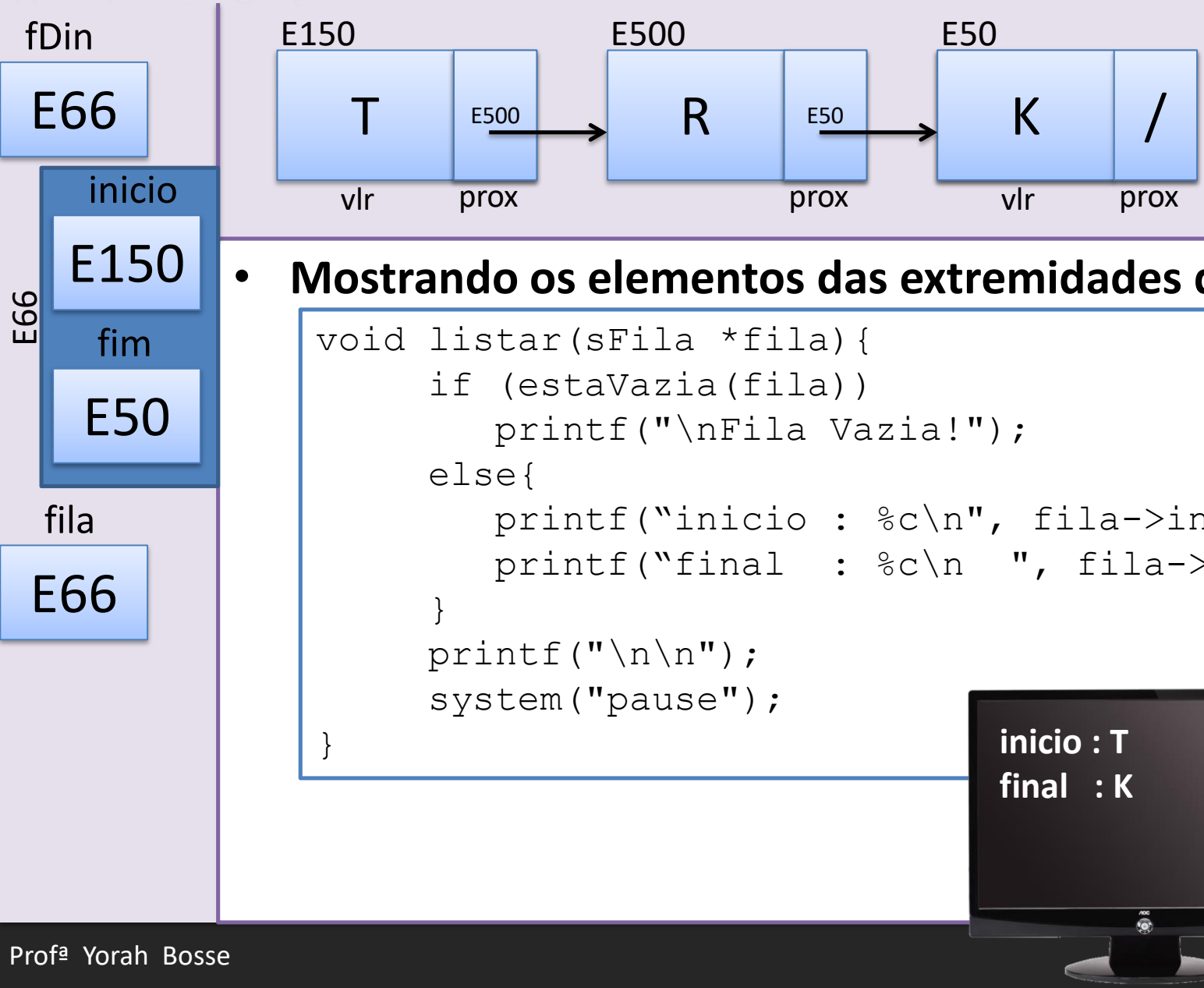


- **Inserindo mais um elemento no final da fila**

```
void enqueue(sFila *fila, char dado){
     sNoFila *aux;
     aux = (sNoFila *) malloc (sizeof(sNoFila));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = NULL;
     if (fila->inicio == NULL) fila->inicio = aux;
     else fila->fim->prox = aux;
     fila->fim = aux;
}
```

Impl. – FILA – Mostrar Extremidades

Visualização Gráfica

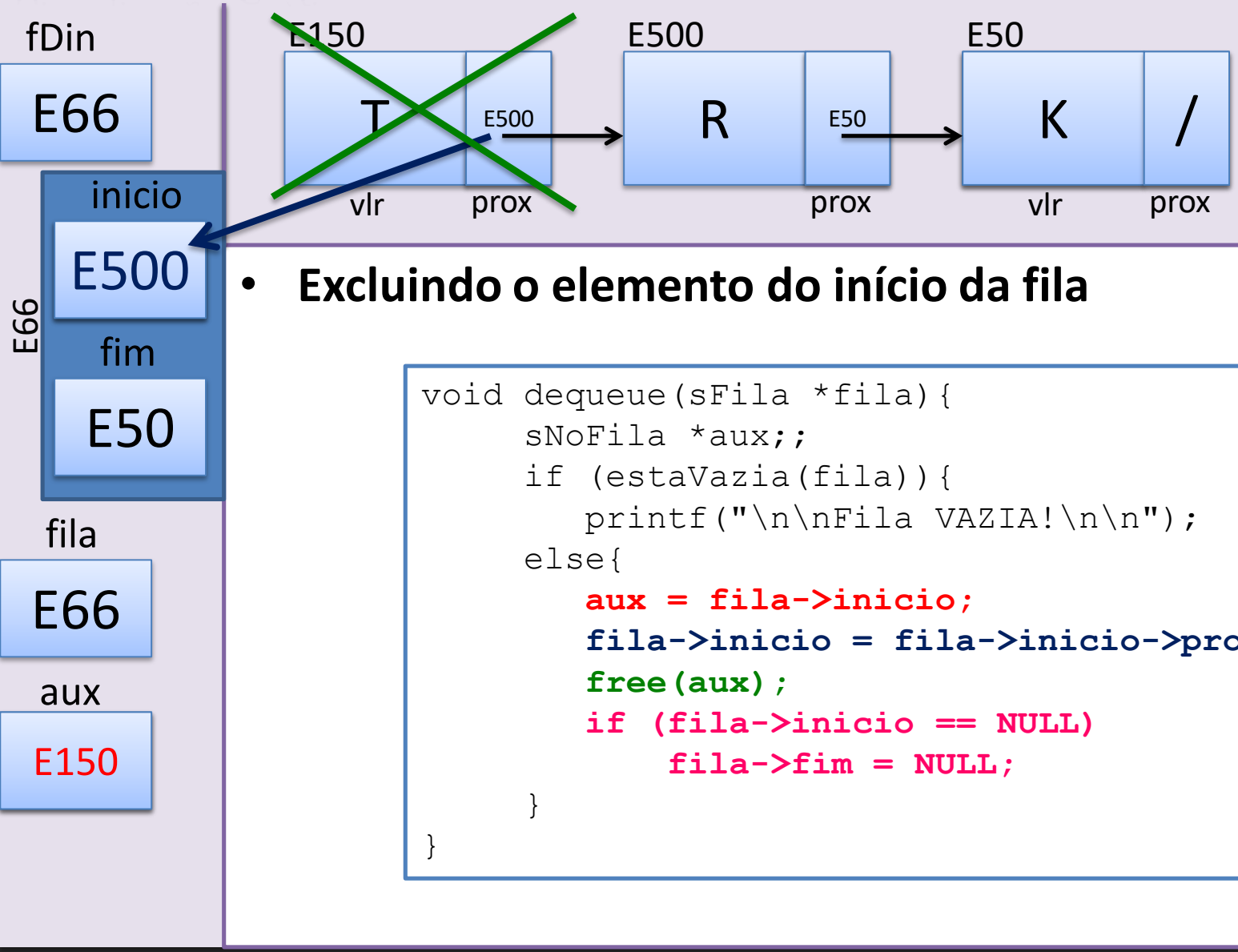


- **Mostrando os elementos das extremidades da fila**

```
void listar(sFila *fila){
     if (estaVazia(fila))
        printf("\nFila Vazia!");
     else{
        printf("inicio : %c\n", fila->inicio->vlr);
        printf("final  : %c\n  ", fila->fim->vlr);
     }
     printf("\n\n");
     system("pause");
}
```

Implementação – FILA – Excluir (dequeue)

Visualização Gráfica



- **Excluindo o elemento do início da fila**

```
void dequeue(sFila *fila){
     sNoFila *aux;;
     if (estaVazia(fila)){
        printf("\n\nFila VAZIA!\n\n");
     else{
        aux = fila->inicio;
        fila->inicio = fila->inicio->prox;
        free(aux);
        if (fila->inicio == NULL)
            fila->fim = NULL;
     }
}
```

Filas com prioridades

- Problema: encontrar uma implementação eficiente que permita uma rápida colocação e uma rápida retirada da fila.
  - Correio: uma pessoa com incapacidade física pode ter prioridades sobre outras
  - Cabines de pedágio: veículos como “carros de polícia”, “ambulância”, “carros de bombeiro” podem passar imediatamente, mesmo sem pagamento.

- Em filas com prioridades, os elementos são retirados da fila de acordo com suas prioridades e suas posições correntes na fila.

Fonte: slide elaborado pelos professores Marcelo Zorzan e Rachel Reis

Aplicações

- **Exemplos de aplicações:**
  - **Fila de impressão**
  - **Fila de comandos a serem executados por determinado sistema**

Leitura Recomendada

- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005.
  - Pág 130 (Fila)
  - Pág 138 (Filas com Prioridades)

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.
  - Pág 209 (Fila)

- FEOFILOFF, Paulo. Algoritmos em Linguagem C. Editora Campus, 2009.
  - Pág 31 (Fila)