Fila

26 e 31/8/2008

Representação/Implementação:

Seqüencial estática

Encadeada dinâmica

Exercícios/Aplicações

Operações sobre a fila

- Implementar as 9 operações do TAD FILA:

Cria(F): cria uma fila F vazia

Tornar_Vazia(F): Reinicializa uma fila existente F como uma fila vazia.

Destroi(F): Remove a fila criada da memória

Entra(F,X): X entra no fim da fila F

Sai(F,X): o primeiro elemento da fila F é retirado da fila e atribuído a X

Começo_Fila(F, X): Mostra o começo da fila F sem remover o item.

Y=Vazia(F): verdade se a fila estivar vazia; caso contrário, falso

Y=Cheia(F): verdade se a fila estiver cheia; caso contrário, falso

Y = Tamanho (F): Retorna o tamanho da fila F

- Atenção: considerações sobre TAD
  - Arquivos .c e .h, parâmetros, mensagens de erro

Fila.h

```
#define TamFila 100
typedef char elem;
typedef struct fila Fila;

/* cria uma fila vazia F. Deve ser chamada antes da fila ser usada. Retorna erro
     se não há memória. */
Fila* Cria(int *flagErro);

/* esvazia uma fila F para poder ser reusada. Retorna erro se Fila não existe */
void Esvazia(Fila* F, int *flagErro);

/* remove a fila criada da memória. Retorna erro se a Fila não existe  */
void Destroi(Fila *F,  int *flagErro);
```

Estamos pressupondo que a Fila existe, pois nenhum erro é retornado para estas 3 operações

```
/* retorna o número de elementos de F */
int Tamanho(Fila* F);

/* retorna true se a fila estiver vazia; false caso contrário */
int Vazia(Fila* F);

/* retorna true se a fila estiver cheia; false caso contrário */
int Cheia(Fila* F);
```

MAS, os usuários são imprevisíveis! É melhor checar se a Fila existe em TODAS as funções do TAD !

Retorno de erro!

```
/* retorna o número de elementos de F. Retorna erro (sucesso = 0 e Fila
    inexistente = 1) se a fila não existe; no caso de erro o retorno será -1.  */
int Tamanho(Fila* F,int *flagErro);

/* retorna true se a fila estiver vazia; false caso contrário. Retorna erro
    (sucesso = 0 e Fila inexistente = 1) se a fila não existe; neste caso o retorno
    será true.   */
int Vazia(Fila* F,int *flagErro);

/* retorna true se a fila estiver cheia; false caso contrário. Retorna erro
    (sucesso = 0 e fila inexistente = 1) se a fila não existe; neste caso o retorno
    será false.  */
int Cheia(Fila* F,int *flagErro);
```

/* insere o elemento X no fim da fila F. Se F estiver cheia erro = 1 e se a operação tiver sucesso, erro = 0 */

```
void Entra(Fila* F, elem X, int* erro);
```

/* remove elemento da fila F e retorna em X o valor do elemento que estava no início de F. Se F estiver vazia erro = 1 e se a operação tiver sucesso, erro = 0 */

```
void Sai(Fila* F, elem* X, int* erro);
```

/* acessa o valor do elemento do início de F, sem remover.

Se F estiver vazia erro = 1 e se a operação tiver sucesso, erro = 0 */

```
void Inicio(Fila* F, elem* X, int* erro);
```

Fila.c

```
#include "fila.h"
#include <stdlib.h> /* malloc, free, exit */
#include <stdio.h> /* printf */

struct fila{
                int inicio, fim, total;
                elem itens[TamFila];
};
```

```
Fila* Cria(int *flagErro){
    Fila *F = (Fila*)malloc(sizeof(Fila));
    if (F == NULL) {
            *flagErro = 1; // ERRO_MEMORIA_INSUFICIENTE
        return F;
     }
     else
     *flagErro = 0; // SUCESSO
     F->inicio=0;
     F->fim=0;
     F->total=0;
     return F;
}
```

```
void Esvazia(Fila *F , int *flagErro) {
    if(F != NULL){
      F->inicio=0;
      F->fim=0;
      F->total=0;
      *flagErro = 0; //SUCESSO
    }
    else
       *flagErro = 1; // ERRO_PONTEIRO_NULO
}

void Destroi(Fila *F, int *flagErro) {
   if(F != NULL){
        free(F);
        *flagErro = 0; //SUCESSO
    }
    else
        *flagErro = 1; // ERRO_PONTEIRO_NULO
}
```

```
int Vazia(Fila *F,int *flagErro) {
   if(F != NULL){
     *flagErro = 0; //SUCESSO
     if (F->total==0)
       return 1;
     else return 0;
   }
    else {
      *flagErro = 1; //
     ERRO_PONTEIRO_NULO
      return 1;
    }
}
```

```
int Cheia(Fila *F,int *flagErro) {
   if(F != NULL){
     *flagErro = 0; //SUCESSO
     if (F->total==TamFila-1) return 1;
     else return 0;
   }
    else {
     *flagErro = 1; // ERRO_PONTEIRO_NULO
     return 0;
    }
}
int Tamanho(Fila *F,int *flagErro) {
    if(F != NULL){
     *flagErro = 0; //SUCESSO
     return F->total;
   }
    else {
     *flagErro = 1; // ERRO_PONTEIRO_NULO
     return -1;
    }
}
```

/* insere o elemento X no fim da fila F. Se F estiver cheia erro = 1 e se a operação tiver sucesso, erro = 0 */

```
void Entra(Fila *F, elem X, int *erro) {
    if (!Cheia(F,erro)) {
          if (*erro == 0) {
            *erro=0;
            F->total++;
            F->itens[F->fim]=X;
            if (F->fim==TamFila-1) F->fim=0;
            else F->fim++;
          }
}
     else *erro=1;
}
```

```
void Sai(Fila *F, elem *X, int *erro) {
        if (!Vazia(F,erro)) {
          if (*erro == 0) {
          *erro=0;
          F->total--;
          *X=F->itens[F->inicio];
          if (F->inicio==TamFila-1) F->inicio=0;
          else F->inicio++;
      }
}
      else *erro=1;
}
```

```
void Inicio(Fila* F, elem* X, int* erro) {
    if (!Vazia(F,erro)) {
        if (*erro == 0) {
        *erro=0;
        *X=F->itens[F->inicio];
    }
}
    else *erro=1;
}
```

```
//versão do programa de edição de texto, agora com fila
#include <stdio.h>
#include "fila.h"

int main(void) {
    elem c, x;
    int erro;
    Fila* F = Cria(&erro);
    printf("Digite seu texto: ");
    while ((c=getche())!='\r') {
        if (c=='#') {
           Sai(F,&x,&erro);
           if (erro) printf("(erro) ");
           else printf("(%c retirado) ",x);
        }
        else if (c=='@') {
            Esvazia(F, &erro);
            printf("(fila esvaziada) ");
        }
        else {
            Entra(F,c,&erro);
            if (erro) printf("(erro) ");
        }
    }
    printf("\n\nTirando tudo da fila: ");
    while (!Vazia(F,&erro)) {
        Sai(F,&x,&erro);
        if (erro) printf("erro ");
        else printf("%c ",x);
    }
    system("pause");
    Destroi(F,&erro);
    return 0;
}
```

Exercício

- Faça uma rotina para verificar se os elementos de uma fila estão ordenados de forma crescente

Exercício

- Faça uma rotina que inverta uma fila F1, criando-se uma nova fila F2

Exercício

- Desafio: como criar uma fila “mais genérica” que possa guardar tipos diferentes (inteiros e reais, por exemplo)?
  - TAD ainda melhor!

TROCA:
typedef char elem;

```
#define TamFila 100

typedef union {
      int i;
      float r;
} elem;

typedef struct {
                  int inicio, fim, total;
                  elem itens[TamFila];
                  enum tipo {
                                    inteiro, real
                   } Tipo;
} Fila;
```

Cria:

```
void Cria(Fila* F, int *flagErro, int tipo)

        F->inicio=0;
        F->fim=0;
        F->total=0;
        F->Tipo=tipo;
```

Cliente: (0=inteiro|1=real)

```
if (tipo==0)
       Create(&F,&erro, inteiro);
   else Create(&F,&erro, real);
```

Entra

```
if (F->Tipo==inteiro)
                F->itens[F->fim].i=X->i;
          else F->itens[F->fim].r=X->r;
```

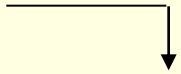
Sai

```
if (F->Tipo==inteiro)
                X->i=F->itens[F->inicio].i;
          else X->r=F->itens[F->inicio].r;
```

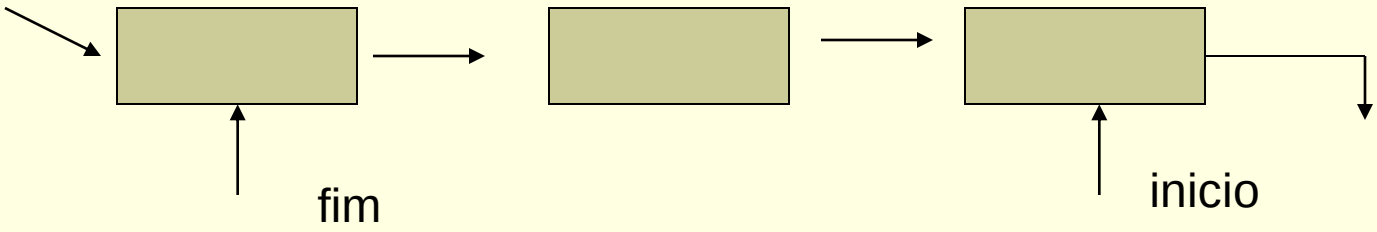
Implementações de Filas: encadeada dinâmica – pensamentos iniciais

Como seria uma operação de inserir e retirar elementos da Fila

Q { fila vazia}

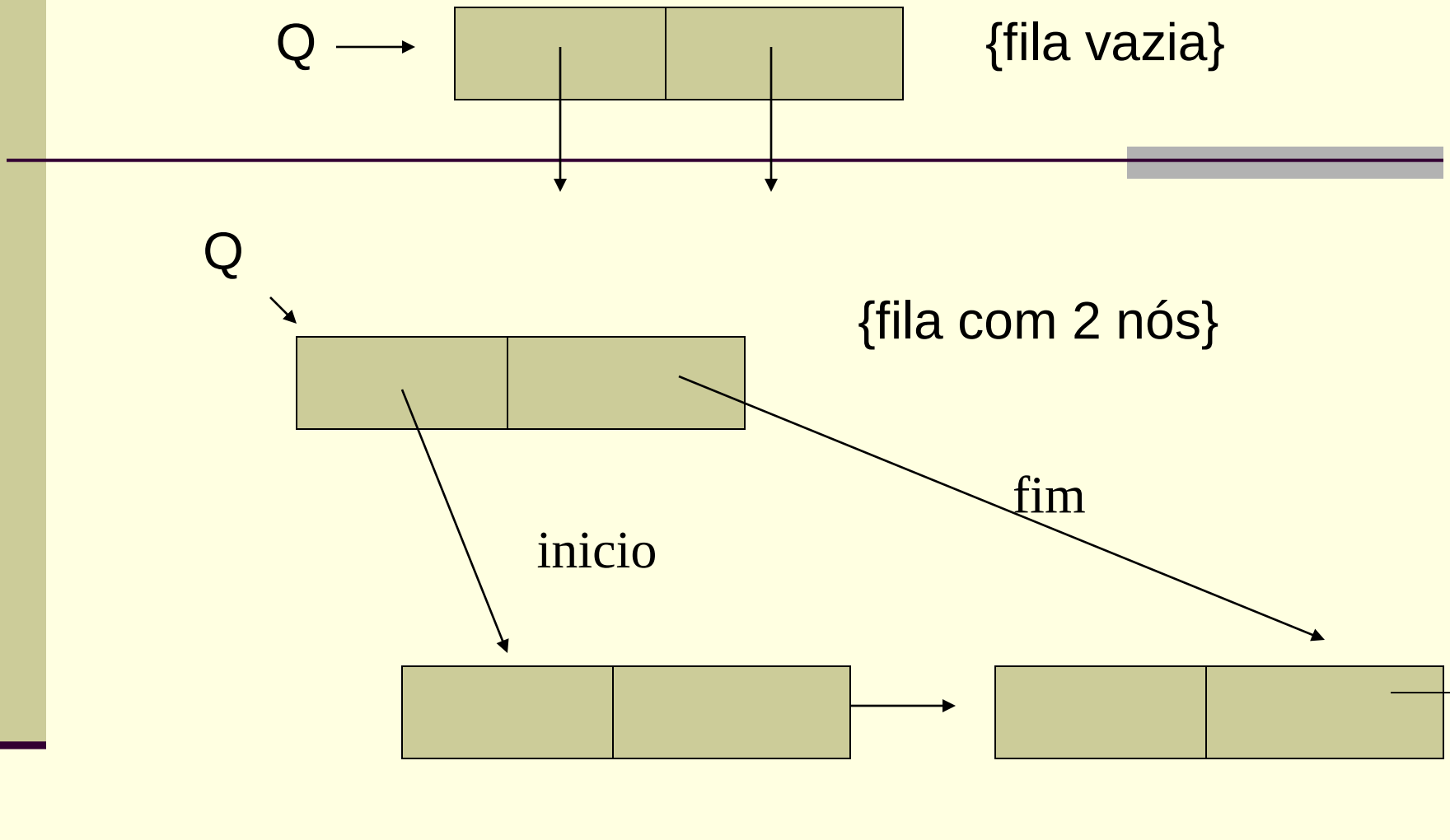


Q



Problema

- Inserção ocorre facilmente, colocando o item no fim da fila.
- PORÉM, esta representação não permite acesso direto ao item do inicio da fila, pois tenho que ajustar o ponteiro do anterior a ele.
  - Temos que percorrer toda a fila para se retirar o item do início da fila.
  - Problema com filas longas.
- Uma posição clara para Pilhas e Filas é que todas as operações menos tornar_vazia devem ser feita em O(1).



Solução: Uso de nó cabeça para guardar os ponteiros de início e fim de fila, crescendo para o fim. Veja como a ordem se inverte entre início e fim.

Implementações de Filas: encadeada dinâmica

```
typedef char elem;
typedef struct fila Fila;
```

Não foi alterado!

Fila.h

```
typedef struct bloco {
        elem info;
        struct bloco *prox;
} no;

struct fila {
        no *inicio, *fim;
        int tam;
};
```

Fila.c

- Cria aloca o nó cabeça e seta inicio e fim com nil

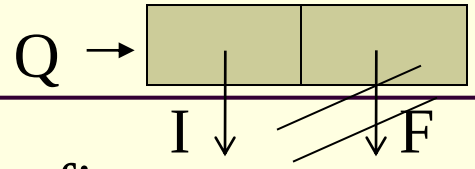
- Entra aloca um nó e coloca o item.
  - Se este é o primeiro então inicio aponta para ele,
  - Se a fila não é vazia devemos ajustar o ponteiro next do último nó
  - e fazer fim apontar para este último nó.

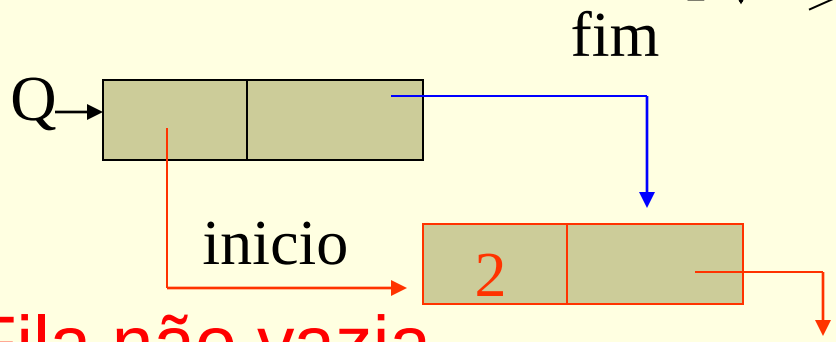
- Sai ajusta o ponteiro de inicio.
  - Se a fila se tornou vazia devemos setar fim para nil.

- Implementar o TAD fila dinâmica.

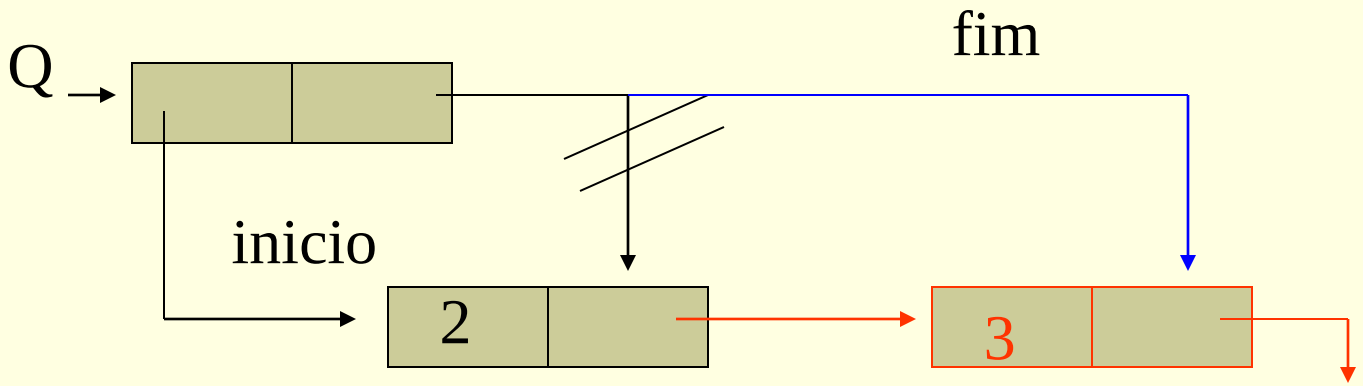
Inserção (Entra)

- Fila vazia



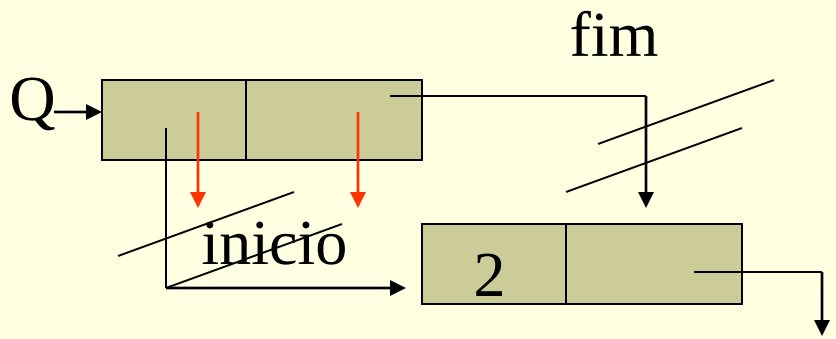


- Fila não vazia

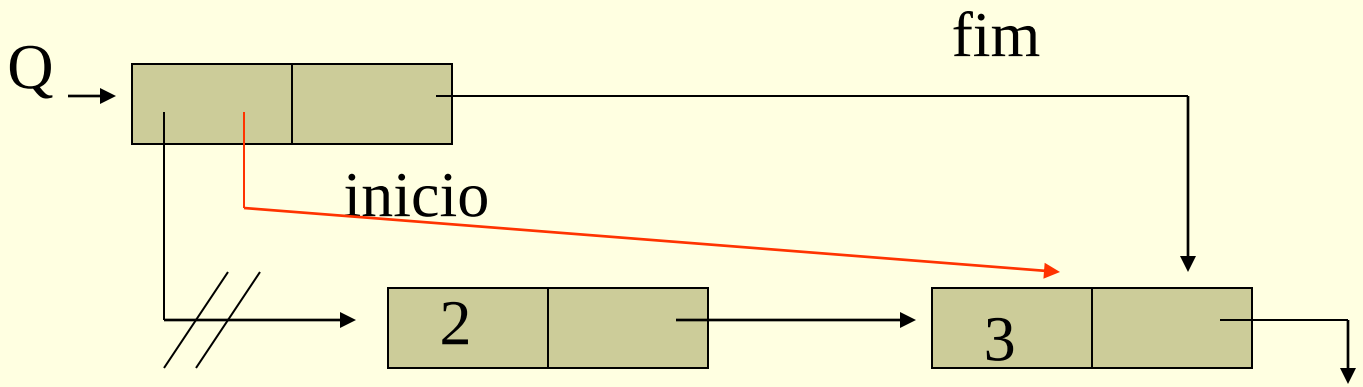


Remoção (Sai)

- Se a fila se tornar vazia



- Se ainda existem elementos depois da remoção



Implementações de Filas: Encadeada Dinâmica

```
Fila* Cria(int *flagErro) {
    Fila *F = (Fila*)malloc(sizeof(Fila));
    if (F == NULL) {
            *flagErro = 1; // ERRO_MEMORIA_INSUFICIENTE
        return F;
     }
     else {
     *flagErro = 0; // SUCESSO
     F->inicio=NULL;
     F->fim=NULL;
     F->total=0;
     return F;
    }
}
```

```
void Esvazia(Fila *F , int *flagErro) {
    no *ndel, *nextno;

    if(F != NULL){
      nextno = F->inicio;
      while (nextno !=NULL){
      ndel = nextno;
      nextno = nextno->prox;
      free(ndel);
      }
      F->inicio=NULL;
      F->fim=NULL;
      F->total=0;
      *flagErro = 0; //SUCESSO
    }
    else
       *flagErro = 1; // ERRO_PONTEIRO_NULO
}
```

```
void Destroi(Fila *F, int *flagErro) {
   if(F != NULL){
        Esvazia(F , flagErro);
        free(F); //SUCESSO
        // *flagErro = 0;
    }
    else
        *flagErro = 1; // ERRO_PONTEIRO_NULO
}
```

```
void Entra(Fila *F, elem X, int *erro) {
     no *p =(no*) malloc(sizeof(no));
    if (F==NULL)
       *erro=1;
    else {
     if (p==NULL)
        *erro=1;
     else {
          *erro=0;
          F->total++;
          p->info=X;
          p->prox=NULL;
          if (F->inicio==NULL) //Se é o primeiro, inicio aponta para o nó
             F->inicio=p;
          else F->fim->prox=p; // senão ajusta o campo prox do último nó
          F->fim=p;            // fim aponta para o nó
     }
}
}
```

```
void Sai(Fila *F, elem *X, int *erro) {
    no *p;
    if (!Vazia(F,erro)) {
    if (*erro == 0) {
        *erro=0;
        F->total--;
        p=F->inicio;
        *X=p->info;
        F->inicio=F->inicio->prox;
        if (F->inicio == NULL) //Se ficou vazia ajusta fim
           F->fim = NULL;
        free(p);
    }
}
     else *erro=1;
}
```

```
void Inicio(Fila* F, elem* X, int* erro) {

    if (!Vazia(F,erro)){
        if (*erro == 0) {
         *erro=0;
         *X=F->inicio->info;
    }
    else *erro=1;
}
}
int Vazia(Fila *F,int *flagErro) {
   if(F != NULL){
    *flagErro = 0; //SUCESSO
    if (F->total==0)
      return 1;
    else return 0;
   }
    else {
     *flagErro = 1; // ERRO_PONTEIRO_NULO
      return 1;
    }
}
```

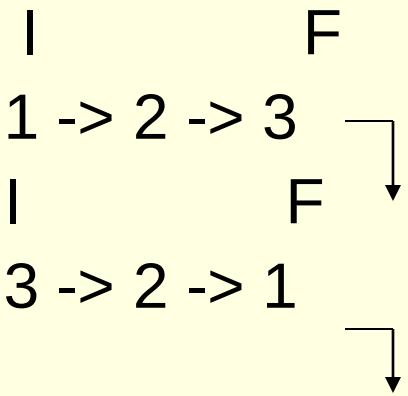
```
int Cheia(Fila *F, int *flagErro) {
    if(F != NULL){
     *flagErro = 0; //SUCESSO
     return 0;
    }
    else {
      *flagErro = 1; // ERRO_PONTEIRO_NULO
      return 0;
    }
}
int Tamanho(Fila *F,int *flagErro) {
    if(F != NULL){
     *flagErro = 0; //SUCESSO
     return F->total;
}
    else {
     *flagErro = 1; // ERRO_PONTEIRO_NULO
     return -1;
    }
}
```

```c
#include <stdio.h>
#include "lista_como_fila.h"

int main(void) {
    int erro;
    elem x;
    int tam;

    Fila* F = Cria(&erro);

    x='a'; Entra(F,x,&erro);
    x='b'; Entra(F,x,&erro);
    x='c'; Entra(F,x,&erro);
    tam = Tamanho(F,&erro);
    printf("O numero de elementos eh %d\n",tam);
    Sai(F,&x,&erro);
    if (!erro) printf("O elemento desenfileirado eh %c\n",x);
    else printf("erro\n");
    Inicio(F, &x, &erro);
    if (!erro) printf("O elemento do topo eh %c\n",x);
    else printf("erro\n");
    Sai(F,&x,&erro);
    if (!erro) printf("O elemento desenfileirado eh %c\n",x);
    else printf("erro\n");
    system("pause");
    Destroi(F,&erro);
    return 0;
}

```

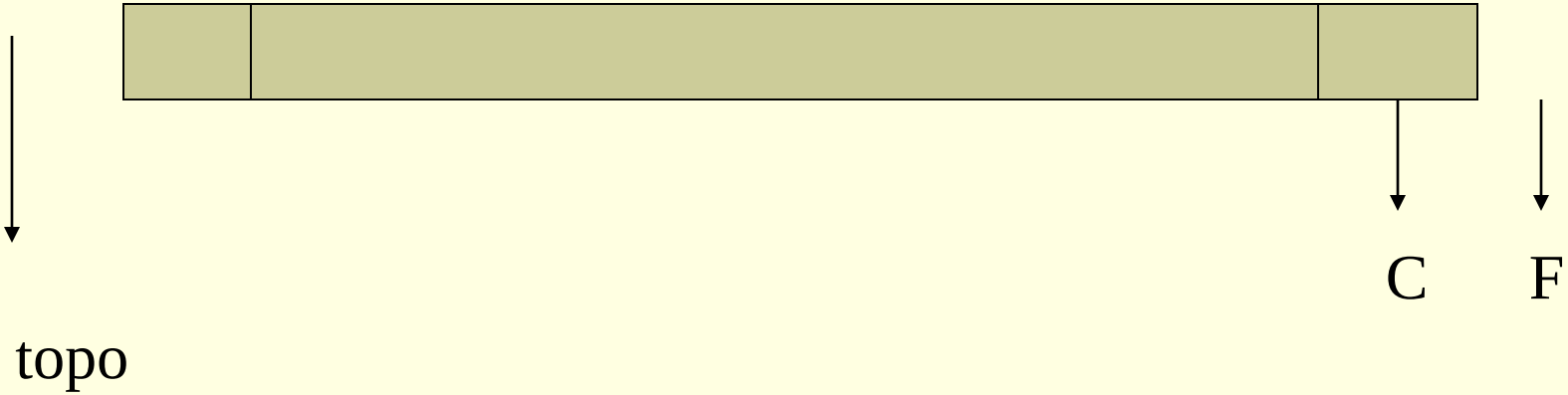
Exercícios:

1) Existe algum problema em se trocar os ponteiros da cabeça da fila?

2) Implemente um procedimento reverso que reposiciona os elementos na fila de forma que o início se torne fim e vice-versa. Use uma pilha.



3) Obtenha uma representação mapeando uma pilha P e uma fila F em um único array V[1..n]. Escreva algoritmos para inserir e eliminar elementos destes 2 objetos de dados. O que você pode dizer sobre a conveniência de sua representação?



Deques

- Um deque é uma fila em que os itens podem ser inseridos/removidos em qualquer lado. (inicio/fim).
- Desta forma, falamos em lado esquerdo ou lado direito do deque.
- As 4 operações básicas são: remove-esq, remove_dir, insere_esq, insere_dir e as outras....

- Um deque de entrada restrita permite a inserção somente em um dos lados sendo que a remoção pode ser feita nos dois.

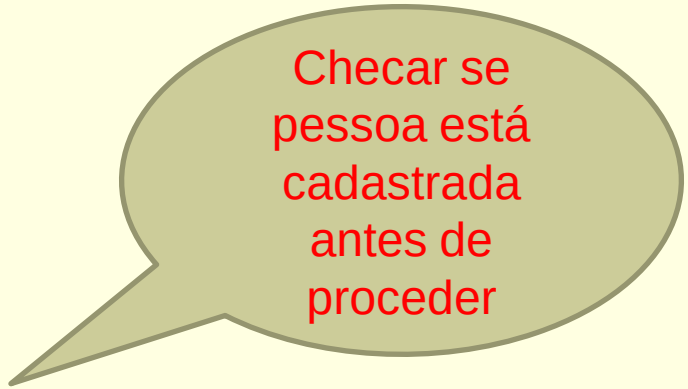
- Um deque de saída restrita permite a remoção somente em um dos lados sendo que a inserção pode ser feita nos dois.

Exercício

- Implemente o sistema para a biblioteca
  - Cada livro deve ser representado por um registro
    - Nome do livro, disponibilidade, fila de espera
  - Ao requisitar um livro, a pessoa entra na fila de espera se o livro não estiver disponível
  - Quando um livro fica disponível, o primeiro da fila de espera do livro deve receber o livro
- Considere prontas as operações sobre a fila

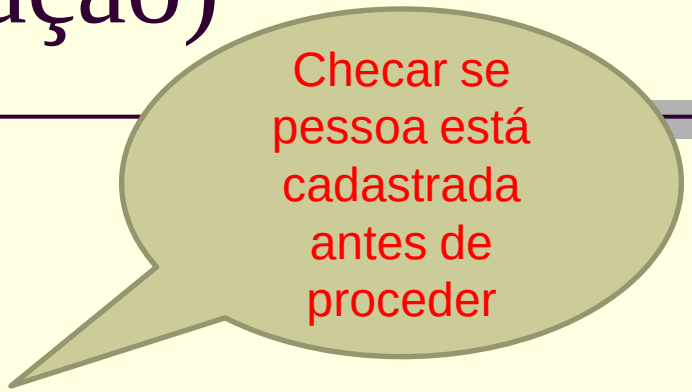
Exercício

```
program biblioteca;
const nroLivros: 1000;
type reg=           record
                    nome: string;
                    disponível: boolean;
                    fila: Fila;
                    end;
var livros: array[1..nroLivros] of reg;
begin
    se livro requisitado
            procure livro no vetor de livros
            se livro disponível
                    dar livro para pessoa
                    tornar livro indisponível
            senão entra(fila,pessoa)

(continua)
```



Exercício (continuação)

```
    senão se livro devolvido
        procure livro no vetor de livros
        se há lista de espera
                sai(fila,pessoa)
                dar livro para pessoa
        senão tornar livro disponível
end.
```



Agradecimentos

- Material montado com exemplos do Prof. Thiago A. S. Pardo