INSTITUTO FEDERAL
CEARÁ

Ernani Andrade Leite

ernani@ifce.edu.br

Aula 06 – Estruturas de Dados

- Assunto: **F**ilas.
- Objetivos:

  - Apresentar os conceitos elementares de Filas e sua aplicação no cotidiano;

  - Aplicação de algoritmos em situações do dia-a-dia;

  - Elaborar programas usando Filas.

- Roteiro:
  1. Introdução.
  2. Definições e Conceitos.
  3. Implementação de Filas com Ponteiros
  4. Operando Filas.
  5. Exercícios.

TAD Fila (1/2)

Uma fila é uma lista linear em que todas as inserções são realizadas em um extremo da lista, e todas as retiradas e geralmente os acessos são realizados no outro extremo da lista.

O modelo intuitivo de uma fila é o de uma fila de espera em que as pessoas no início da fila são servidas primeiro e as pessoas que chegam entram no fim da fila. Por esta razão as filas são chamadas de listas **FIFO**, termo formado a partir de "**F**irst-**I**n, **F**irst-**O**ut".

Existe uma ordem linear para filas que é a "**ordem de chegada**".

Filas são utilizadas quando desejamos processar itens de acordo com a ordem "**PRIMEIRO-QUE-CHEGA, PRIMEIRO-ATENDIDO**". Sistemas operacionais utilizam filas para regular a ordem na qual tarefas devem receber processamento e recursos devem ser alocados a processos.

TAD Fila (2/2)

Um possível conjunto de operações, definido sobre um **tipo abstrato de dados** Fila, é definido a seguir.

1. **FazFilaVazia(Fila)**. Faz a fila ficar vazia.
2. **FilaVazia(Fila)**. Esta função retorna 1 (*true*) se a fila está vazia; senão retorna 0 (*false*).
3. **Enfileira(x, Fila)**. Insere o item *x* no final da fila.
4. **Desenfileira(Fila, x)**. Retira o item *x* que está no início da fila.
5. **ImprimeFila(Fila)**. Imprime os itens da fila (obedecendo a ordem de chegada).

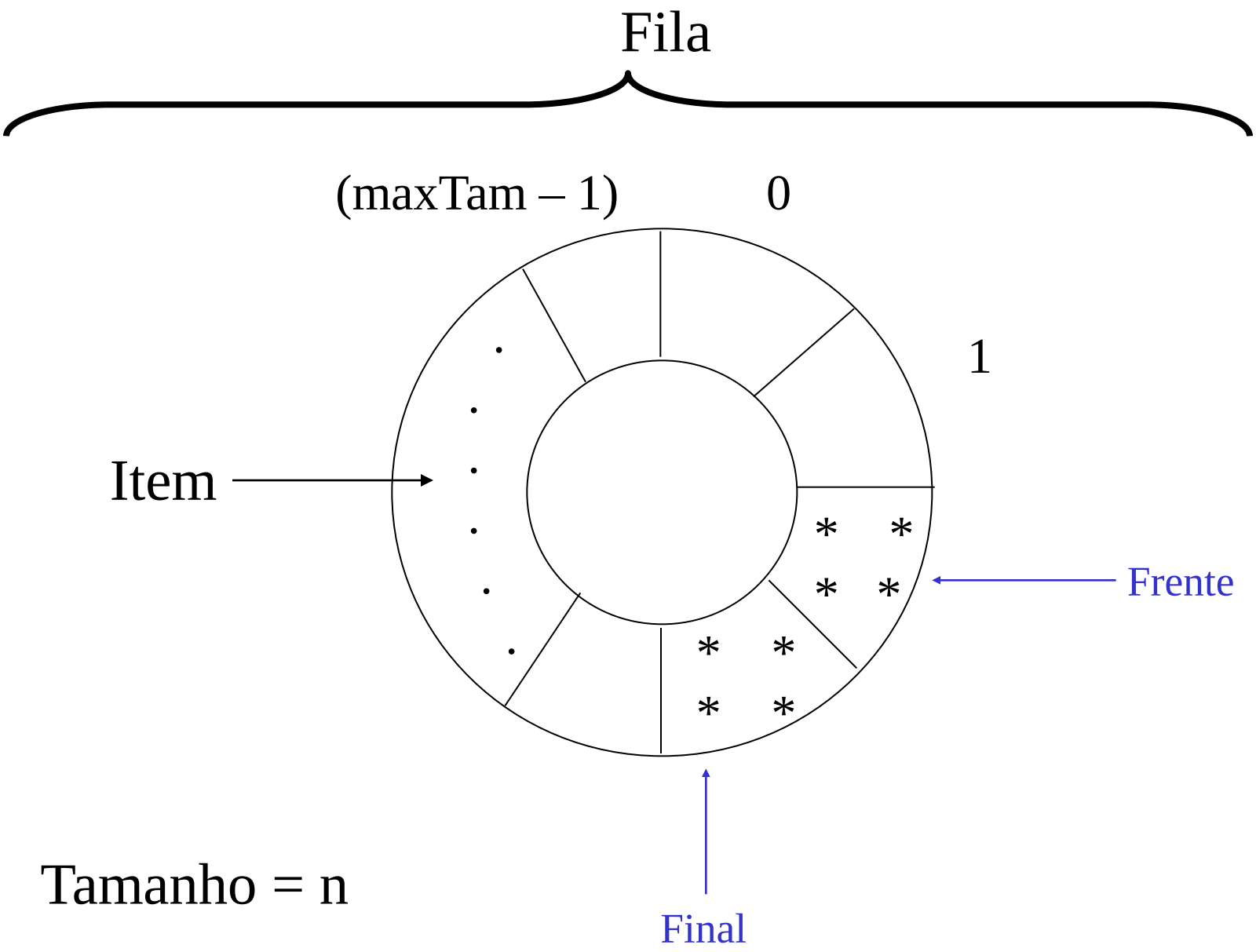
Como no caso do tipo abstrato de dados Lista, existem várias opções de estruturas de dados que podem ser usadas para representar filas. As duas representações mais utilizadas são as implementações através de *arranjos* (*vetores*) e de *apontadores*.

Implementação de Filas através de Arranjos

Em uma implementação através de arranjos os itens são armazenados em posições contíguas de memória. Devido às características da fila, a operação *Enfileira* faz a parte final da fila expandir-se, e a operação *Desenfileira* faz a parte da frente da fila contrair-se. Conseqüentemente, a fila tende a caminhar pela memória do computador, ocupando espaço na parte final e descartando espaço na parte da frente. Com poucas inserções e retiradas de itens a fila vai de encontro ao limite do espaço da memória alocado para ela.

A solução para o problema de caminhar pelo espaço alocado para uma fila é imaginar o **vetor** com um círculo, onde a primeira posição segue a última. A fila se encontra em posições contíguas de memória, em alguma posição do círculo, delimitada pelos apontadores *Frente* e *Final*. Para enfileirar um item basta mover o apontador *Final* uma posição no sentido horário; para desenfileirar um item basta mover o apontador *Frente* uma posição no sentido horário.

Implementação de Filas através de Arranjos



Implementação de Filas através de Arranjos

```
// modelo matemático (estrutura de dados)
struct TipoItem {    // cada item da fila corresponde a um
  char nome[30];     // registro (TipoItem) composto apenas
};                   // do campo nome

const maxTam = 10;   // tamanho máximo da fila

struct TipoFila {
  TipoItem Item[maxTam];
  int Frente;
  int Final;
  int Tamanho;
};
```

O campo *Item* é o principal componente do registro *TipoFila*. O tamanho máximo do **vetor** circular é definido pela constante *maxTam*. Os outros campos do registro *TipoFila* contêm apontadores para a parte da frente (campo *Frente*) e final (campo *Final*) da fila. O campo *Tamanho* indica a quantidade de itens armazenados na fila.

Implementação de Filas através de Arranjos

Na representação circular da Fila existe um pequeno problema: não há uma forma de distinguir uma fila vazia de um fila cheia, pois nos dois casos os apontadores *Frente* e *Final* apontam para a mesma posição do círculo.

Uma possível saída para este problema é utilizar um campo extra (*Tamanho*) para indicar a quantidade de itens armazenados na fila.

Neste caso:
1) a fila está cheia quando *Tamanho* for igual a *maxTam*, e,
2) a fila está vazia quando *Tamanho* for igual a 0 (zero).

Implementação de Filas através de Arranjos

```
// Faz a 'Fila' ficar vazia
void FazFilaVazia(TipoFila *Fila) {
  Fila->Frente = 0;
  Fila->Final = 0;
  Fila->Tamanho = 0;
}
```

```
// Esta função retorna 1 (true) se a 'Fila'
// está vazia; senão retorna 0 (false)
int FilaVazia(TipoFila *Fila) {
  return(Fila->Tamanho == 0);
}
```

Implementação de Filas através de Arranjos

```
// Insere o item 'x' no 'Final' da 'Fila'.
int Enfileira(TipoItem x, TipoFila *Fila) {
  if (Fila->Tamanho == maxTam)
     return(0);      // Erro: Fila Cheia !
  else {
// item inserido no final da fila
     Fila->Item[Fila->Final] = x;
     Fila->Final = Fila->Final + 1;
// se última posição então volta para a primeira
// posição do vetor (vetor circular)
     if (Fila->Final == maxTam)
        Fila->Final = 0;
     Fila->Tamanho = Fila->Tamanho + 1;
     return(1); // Item inserido com sucesso
  }
}
```

Implementação de Filas através de Arranjos

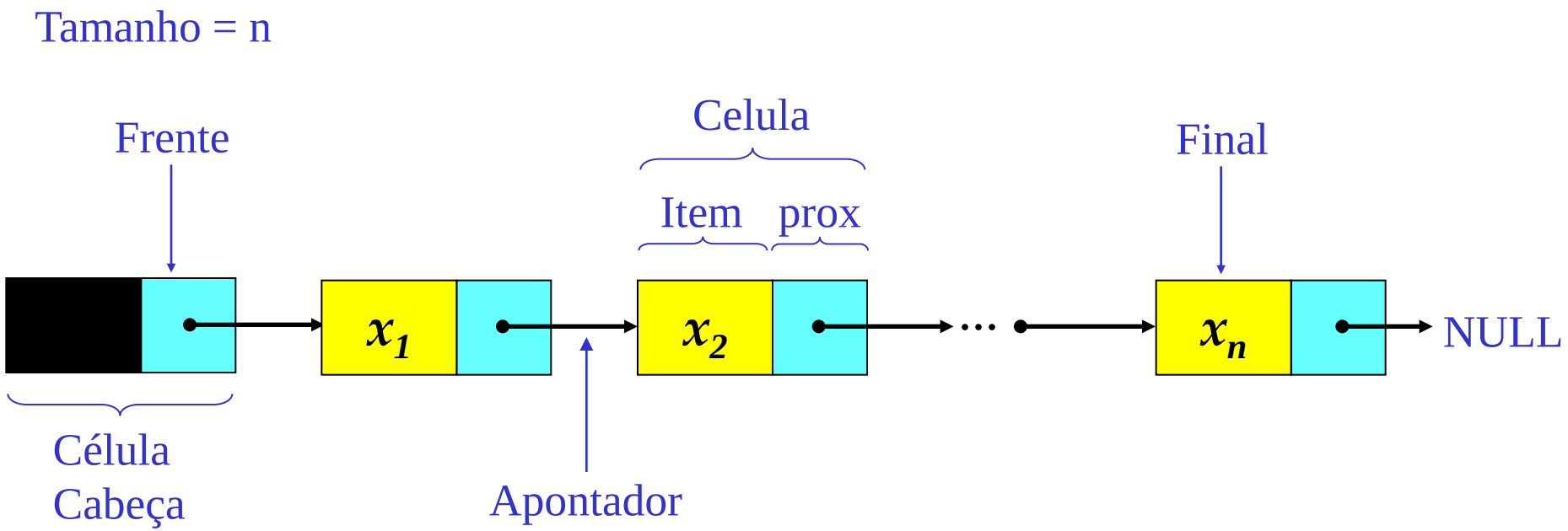
```
// Retira o item 'x' que está na frente da 'Fila'
int Desenfileira(TipoFila *Fila, TipoItem *x) {
   if (FilaVazia(Fila))
      return(0); // Erro: Fila vazia.
   else {
// item retornado
     *x = Fila->Item[Fila->Frente];
     Fila->Frente = Fila->Frente + 1;
// se última posição então volta para a primeira
// posição do vetor (vetor circular)
     if (Fila->Frente == maxTam)
        Fila->Frente = 0;
     Fila->Tamanho = Fila->Tamanho - 1;
     return(1); // Item retirado com sucesso
   }
}
```

Implementação de Filas através de Arranjos

```
// Imprime os itens da fila obedecendo
// a ordem de chegada.
void ImprimeFila(TipoFila *Fila) {
  TipoItem x;

  clrscr();
  for (int i=1; i<=Fila->Tamanho; i++) {
    Desenfileira(Fila, &x);
    printf("%s\n", x.nome);
    Enfileira(x, Fila);
  }

}
```

Implementação de Filas através de Apontadores

Uma célula cabeça é mantida para facilitar a implementação das operações *Enfileira* e *Desinfileira* quando a fila está vazia. Quando a fila está vazia os Apontadores *Frente* e *Final* apontam para a célula cabeça. Para enfileirar um novo item basta criar uma célula nova, ligá-la após a célula que contém x_n e colocar nela o novo item. Para desenfileirar o item x_1 basta desligar a célula cabeça da lista e a célula que contém x_1 passa a ser a célula cabeça.



Implementação de Filas através de Apontadores

```
// modelo matemático (estrutura de dados)
struct TipoItem {    // cada item da fila corresponde a um
  char nome[30];     // registro (TipoItem) composto apenas
};                   // do campo nome
typedef struct Celula *Apontador;
struct Celula {
   TipoItem Item;
   Apontador prox;
};
struct TipoFila {
  Apontador Frente;
  Apontador Final;
  int Tamanho;
};
```

A fila é implementada através de células, onde cada célula contém um item da fila (campo *Item*) e um "apontador" para outra célula (campo *prox*). O registro *TipoFila* contém um "apontador" para a *Frente* da fila (célula cabeça) e um "apontador" para a parte *Final* da fila. O campo *Tamanho* indica a quantidade de itens armazenados na fila.

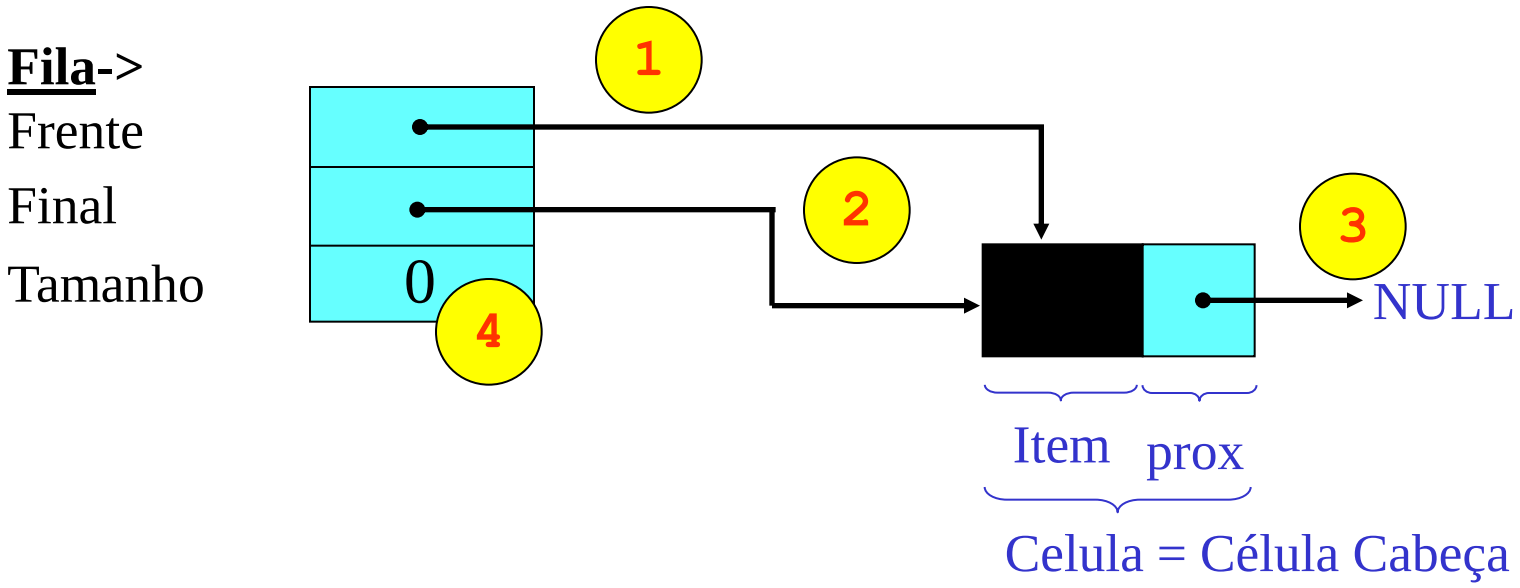
Implementação de Filas através de Apontadores

```
// Faz a 'Fila' ficar vazia criando a célula cabeça
void FazFilaVazia(TipoFila *Fila) {
  Fila->Frente = (Apontador) malloc(sizeof(Celula));
  Fila->Final = Fila->Frente;
  Fila->Frente->prox = NULL;
  Fila->Tamanho = 0;
}
```

```
// Esta função retorna 1 (true) se a 'Fila' está vazia;
// senão retorna 0 (false)
int FilaVazia(TipoFila *Fila) {
  return(Fila->Tamanho == 0);
}
```

Implementação de Filas através de Apontadores

```
void FazFilaVazia(TipoFila *Fila) {
1. Fila->Frente = (Apontador) malloc(sizeof(Celula));
2. Fila->Final = Fila->Frente;
3. Fila->Frente->prox = NULL;
4. Fila->Tamanho = 0;
}
```

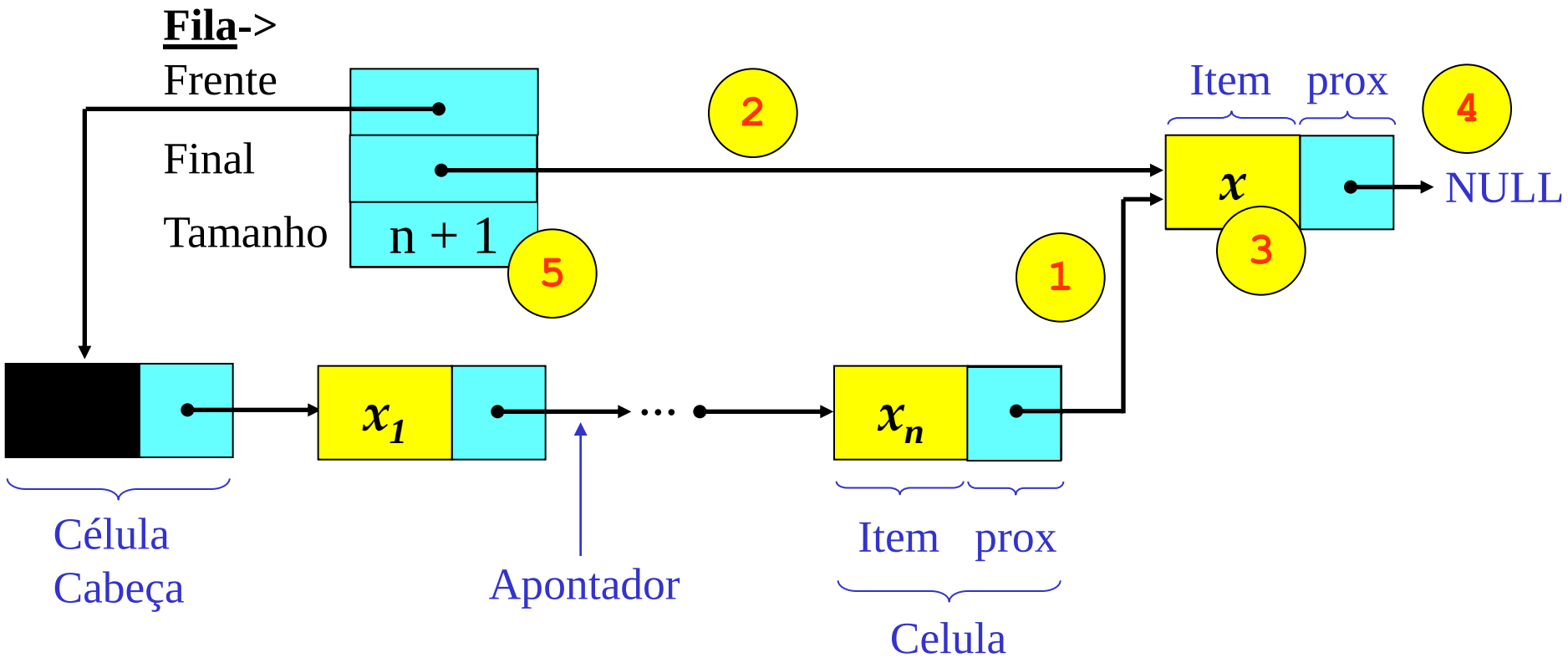


Implementação de Filas através de Apontadores

```
// Insere o item 'x' no 'Final' da 'Fila'.
void Enfileira(TipoItem x, TipoFila *Fila) {

// cria uma nova célula no final da fila
  Fila->Final->prox = (Apontador) malloc(sizeof(Celula));
  Fila->Final = Fila->Final->prox;

// coloca o item 'x' na nova célula
  Fila->Final->Item = x;
  Fila->Final->prox = NULL;

// atualiza o tamanho da fila
  Fila->Tamanho = Fila->Tamanho + 1;

}
```

Implementação de Filas através de Apontadores

```
void Enfileira(TipoItem x, TipoFila *Fila) {
1. Fila->Final->prox = (Apontador) malloc(sizeof(Celula));
2. Fila->Final = Fila->Final->prox;
3. Fila->Final->Item = x;
4. Fila->Final->prox = NULL;
5. Fila->Tamanho = Fila->Tamanho + 1;
}
```

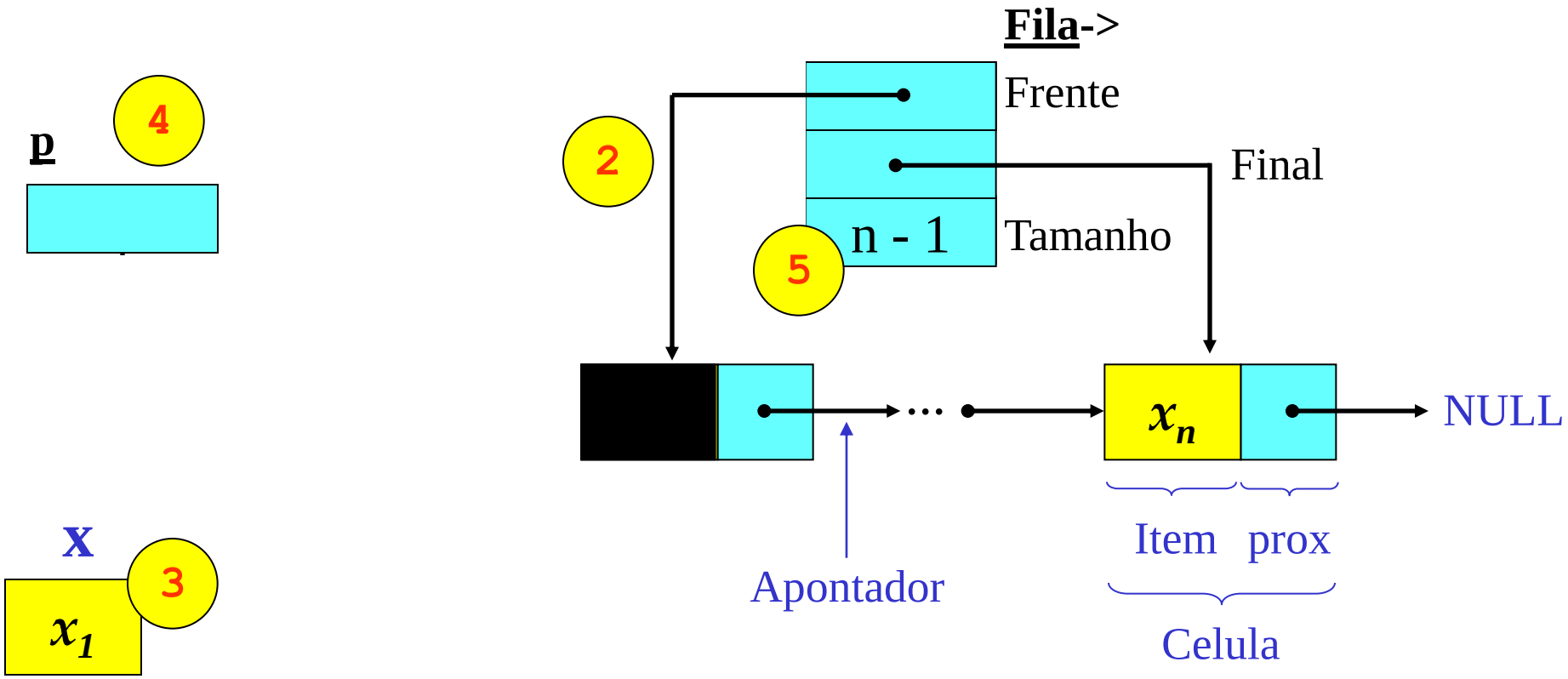


Implementação de Filas através de Apontadores

```
// Retira o item 'x' que está na frente da 'Fila'
int Desenfileira(TipoFila *Fila, TipoItem *x) {
  if (FilaVazia(Fila))
     return(0); // Erro: Fila vazia.
  else {
// retira o item x1 da frente da fila e desliga
// a célula cabeça, a célula que contém x1
// torna-se a nova célula cabeça
     Apontador p;
     p = Fila->Frente;
     Fila->Frente = Fila->Frente->prox;
// item retornado
     *x = Fila->Frente->Item;
     free(p);
// atualiza o tamanho da fila
     Fila->Tamanho = Fila->Tamanho - 1;
     return(1); // Item retirado com sucesso
  }
}
```

Implementação de Filas através de Apontadores

```
int Desenfileira(TipoFila *Fila, TipoItem *x) {
Apontador p;
1. p = Fila->Frente;
2. Fila->Frente = Fila->Frente->prox;
3. *x = Fila->Frente->Item;
4. free(p);
5. Fila->Tamanho = Fila->Tamanho - 1;
```



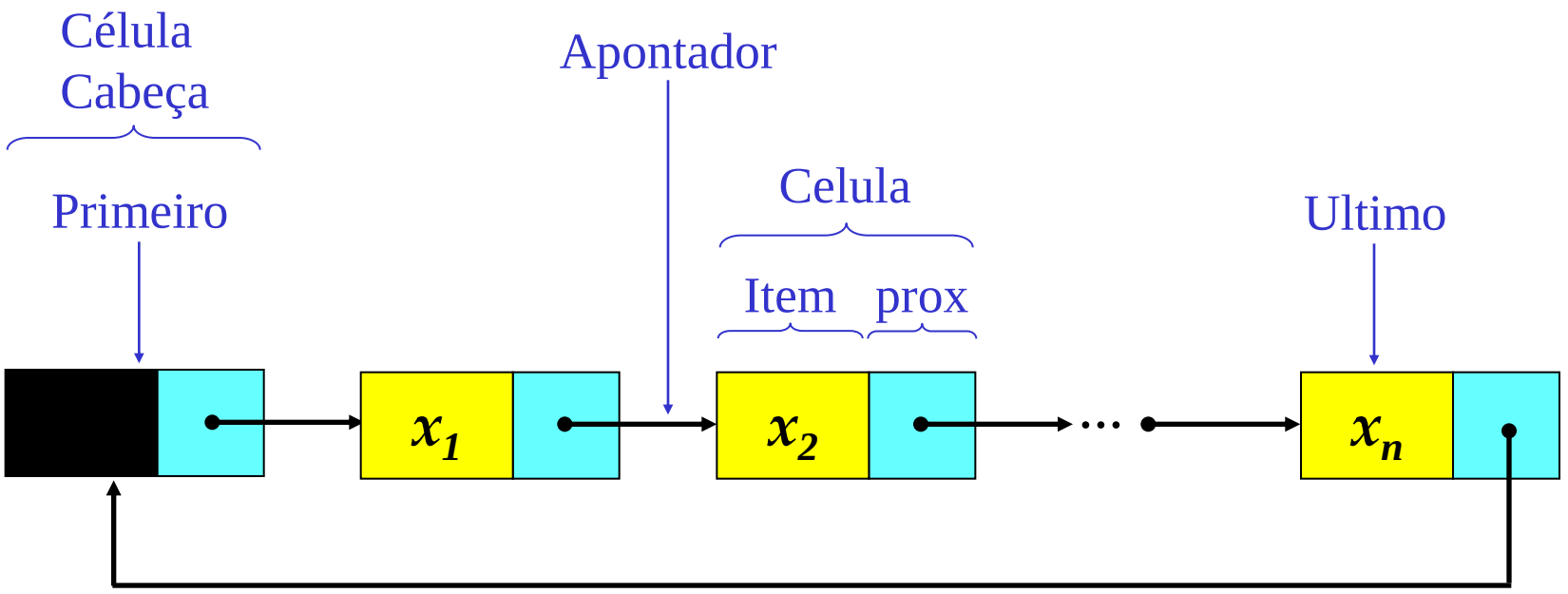
Implementação de Filas através de Arranjos

```
// Imprime os itens da fila obedecendo
// a ordem de chegada.
void ImprimeFila(TipoFila *Fila) {
  TipoItem x;
  int n = Fila->Tamanho;

  clrscr();
  for (int i=1; i<=n; i++) {
    Desenfileira(Fila, &x);
    printf("%s\n", x.nome);
    Enfileira(x, Fila);
  }

}
```

Casos Especiais de Listas Encadeadas (1/2)

Lista Circular Encadeada

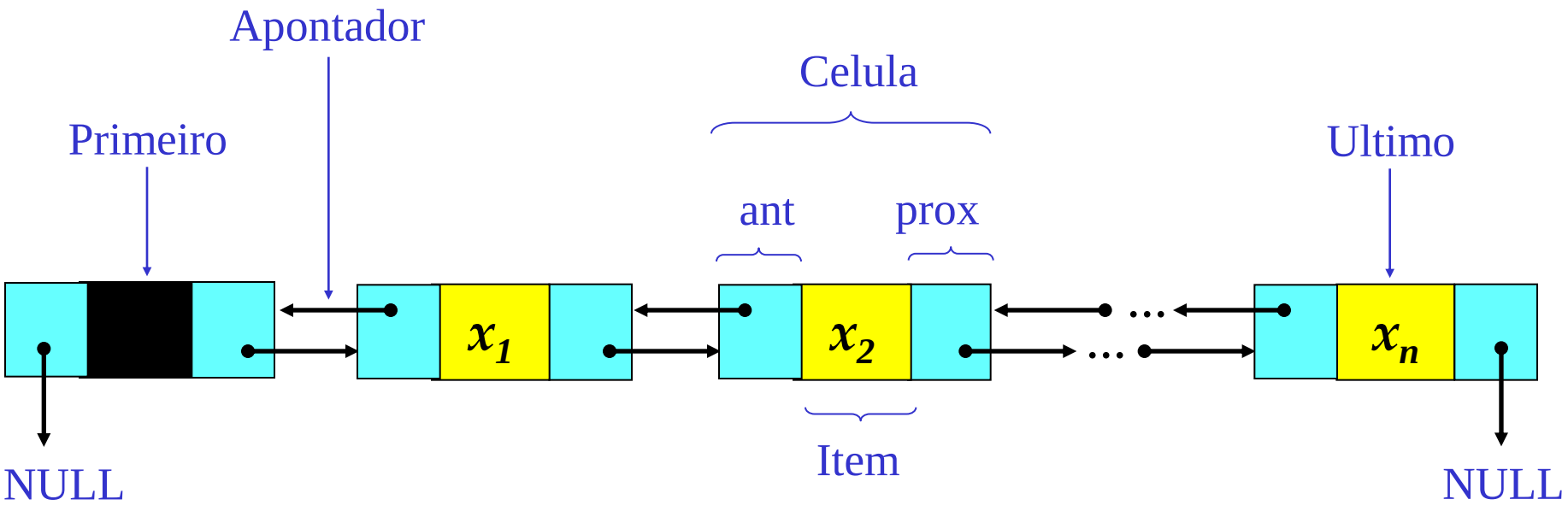
Uma pequena modificação na estrutura física da lista pode ser de grande auxílio: obrigar o último nó da lista a apontar para o nó cabeça, criando assim uma *lista circular encadeada*.



Casos Especiais de Listas Encadeadas (2/2)

Lista Duplamente Encadeada

A estrutura da célula de uma lista duplamente encadeada, com os campos *prox*- aponta para o próximo nó, ou sucessor e *ant*- aponta para o nó anterior, ou antecessor; é possível percorrer a lista nos dois sentidos indiferentemente.



Referências

- Projeto de Algoritmos: com implementações em C e Pascal.
  - Nivio Ziviani.
  - 4ª ed. - São Paulo: Pioneira - 1999.
- Estrutura de Dados Fundamentais: conceitos e aplicações.
  - Silvio do Lago Pereira.
  - 2ª ed. - São Paulo: Érica, 1996.