Pilhas e Filas

Listas, Pilhas e Filas

- Listas:
  Inserção: em qualquer posição
  Remoção: em qualquer posição

- Pilhas:
  Inserção: Topo (primeira posição na lista)
  Remoção: Topo (primeira posição na lista)

- Filas:
  Inserção: Trás (última posição na lista)
  Remoção: Início (primeira posição na lista)

TAD Pilhas

- Tipo Abstrato de dados com a seguinte característica:

O último elemento a ser inserido é o primeiro a ser retirado

(LIFO – *Last In First Out)*

- Analogia: pilha de pratos, livros, etc

- Usos: Chamada de subprogramas, avaliação de expressões aritméticas, etc...

TAD Pilhas

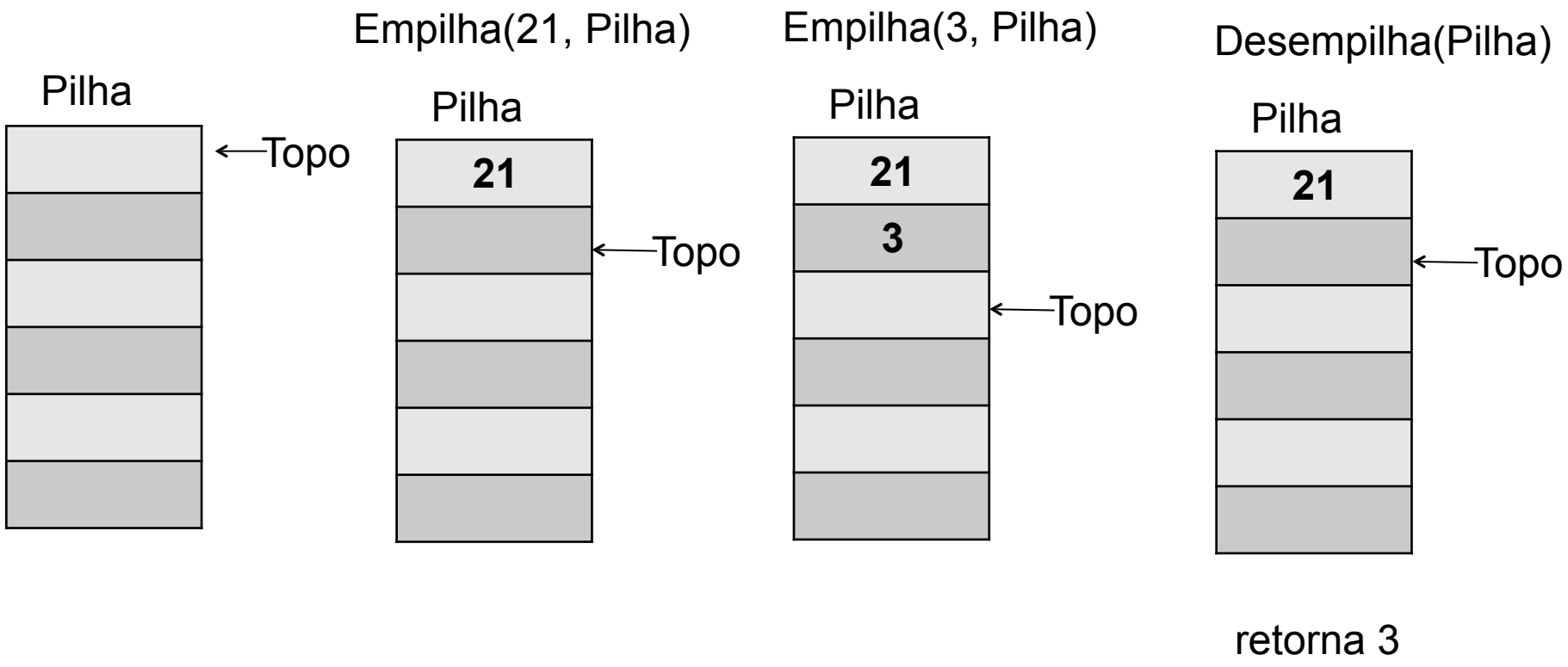
- **Conjunto de operações:**
  1) FPVazia(Pilha). Faz a pilha ficar vazia.
  2) Vazia(Pilha). Retorna true se a pilha está vazia; caso contrário, retorna false.
  3) Empilha(x, Pilha). Insere o item x no topo da pilha.
  4) Desempilha(Pilha). Retorna o item x no topo da pilha, retirando-o da pilha.
  5) Tamanho(Pilha). Esta função retorna o número de itens da pilha.

- **Representações comuns**
  - Alocação sequencial (arranjos) e apontadores

Implementação de Pilhas com alocação Sequencial

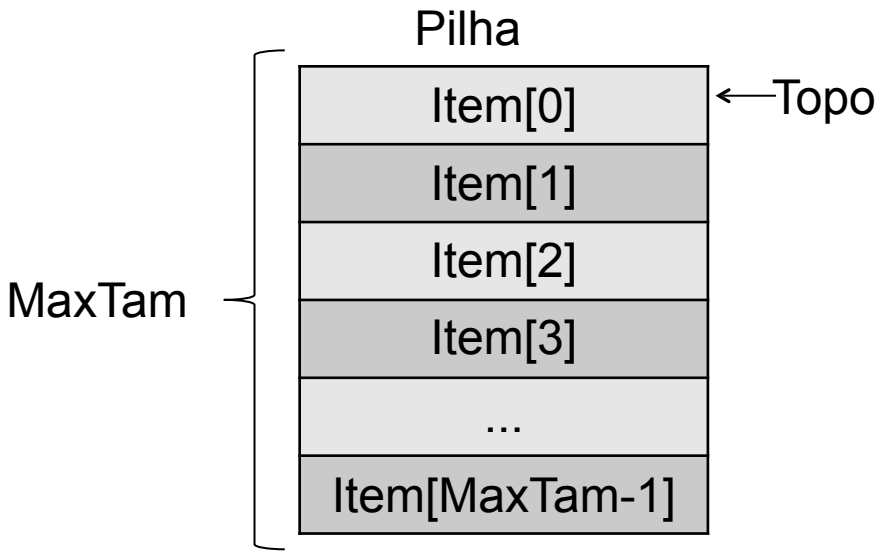
- Os itens da pilha são armazenados em posições contíguas de memória.
- Inserções e retiradas**: apenas no topo**.
- Variável **Topo** é utilizada para controlar a posição do item no topo da pilha.

Implementação de Pilhas com alocação Sequencial



Estrutura de Dados de Pilhas com alocação Sequencial

```
#define MaxTam 1000

typedef int Apontador;
typedef int TipoChave;

typedef struct {
  TipoChave Chave;
  /* outros componentes */
} TipoItem;

typedef struct {
  TipoItem Item[MaxTam];
  Apontador Topo;
} TipoPilha;
```

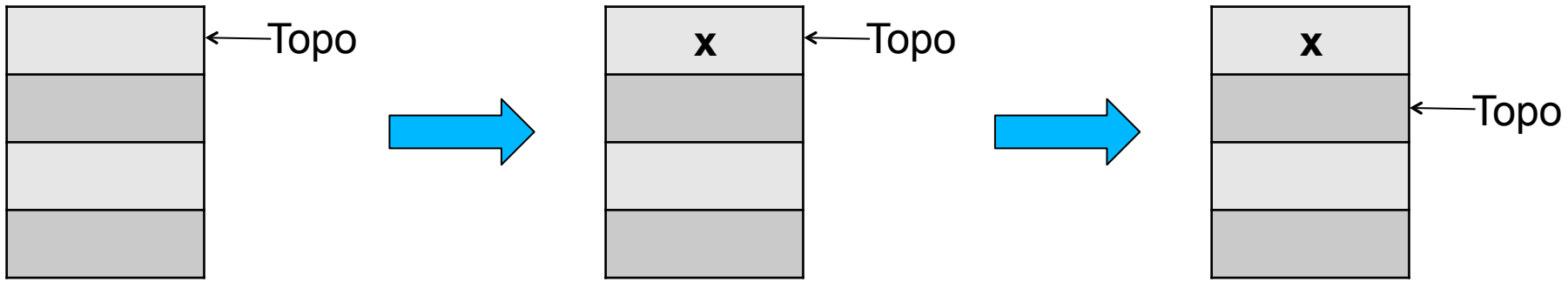


Operações sobre Pilhas com alocação Sequencial

```
void FPVazia(TipoPilha *Pilha)
{
  Pilha->Topo = 0;
}  /* FPVazia */


int Vazia(TipoPilha *Pilha)
{
  return (Pilha->Topo == 0);
}  /* Vazia */
```

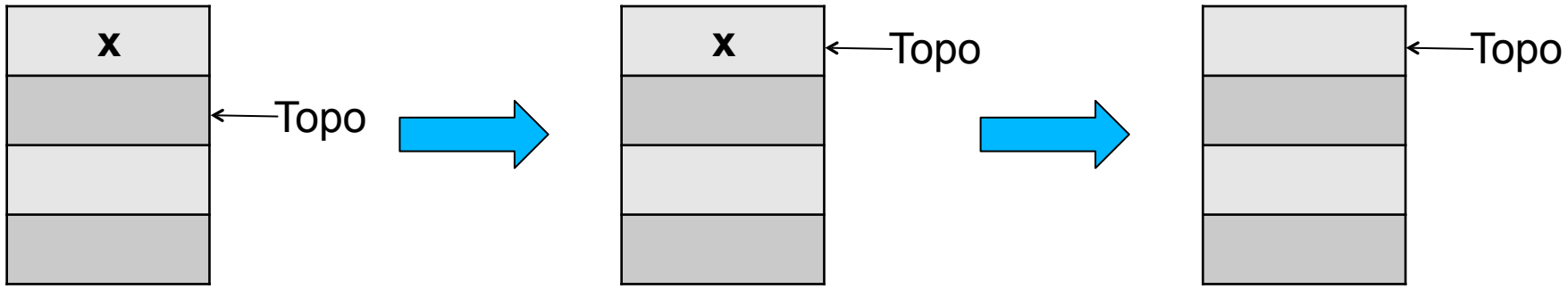
Operações sobre Pilhas com alocação Sequencial

```
void Empilha(TipoItem x, TipoPilha *Pilha)
{
  if (Pilha->Topo == MaxTam)
      printf("Erro: pilha está cheia\n");
  else {
      Pilha->Item[Pilha->Topo] = x;
      Pilha->Topo++;
  }
}
```



Operações sobre Pilhas com alocação Sequencial

```
TipoItem Desempilha(TipoPilha *Pilha)
{
  if (Vazia(Pilha))
      printf("Erro: a pilha está vazia\n");
  else {
       Pilha->Topo--;
      return Pilha->Item[Pilha->Topo];
  }
}
```

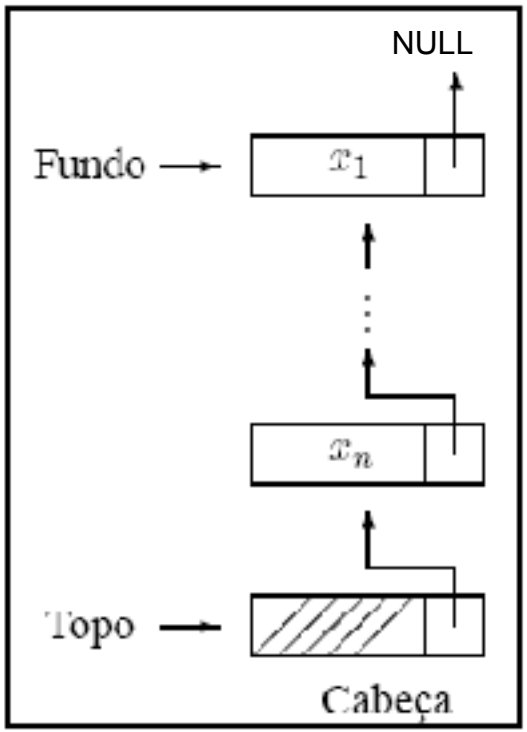


Operações sobre Pilhas Usando Arranjos

```
int Tamanho(TipoPilha *Pilha)
{
  return Pilha->Topo;
}
```

Implementação de Pilhas por meio de Apontadores

- Há uma **célula cabeça** no topo para facilitar a implementação das operações empilha e desempilha quando a pilha está vazia.

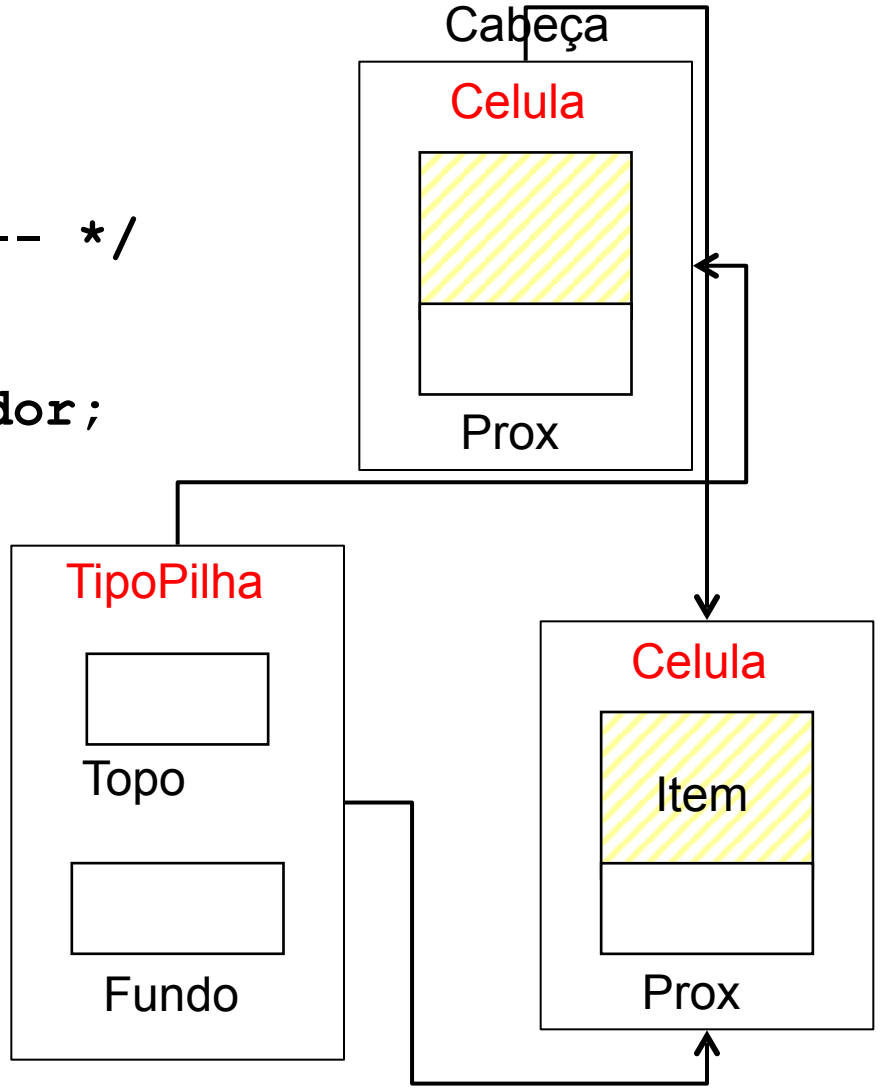


- **Desempilha**: desliga a célula cabeça da lista a próxima célula, que contém o primeiro item, passa a ser a célula cabeça.

- **Empilha**: Cria uma nova célula cabeça e adiciona o item a ser empilhado na antiga célula cabeça.

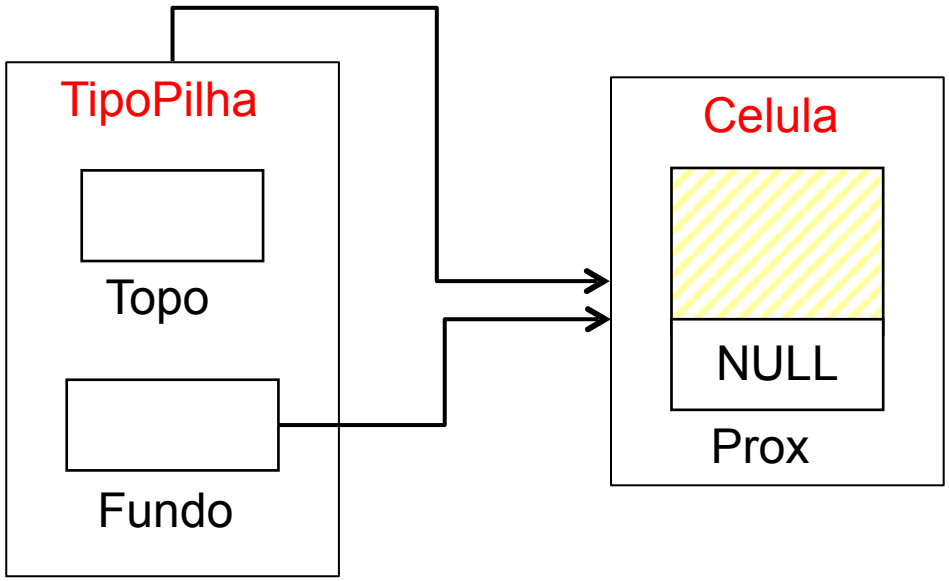
Estrutura da Pilha Usando Apontadores

```
typedef int TipoChave;
typedef struct {
  int Chave;
  /* --- outros componentes --- */
} TipoItem;

typedef struct Celula *Apontador;

typedef struct Celula {
  TipoItem Item;
  Apontador Prox;
} Celula;

typedef struct {
  Apontador Fundo, Topo;
  int Tamanho;
} TipoPilha;
```



Operações sobre Pilhas Usando Apontadores

```
void FPVazia(TipoPilha *Pilha)
{
  Pilha->Topo = (Apontador) malloc(sizeof(Celula));
  Pilha->Fundo = Pilha->Topo;
  Pilha->Topo->Prox = NULL;
  Pilha->Tamanho = 0;
}  /* FPVazia */
```

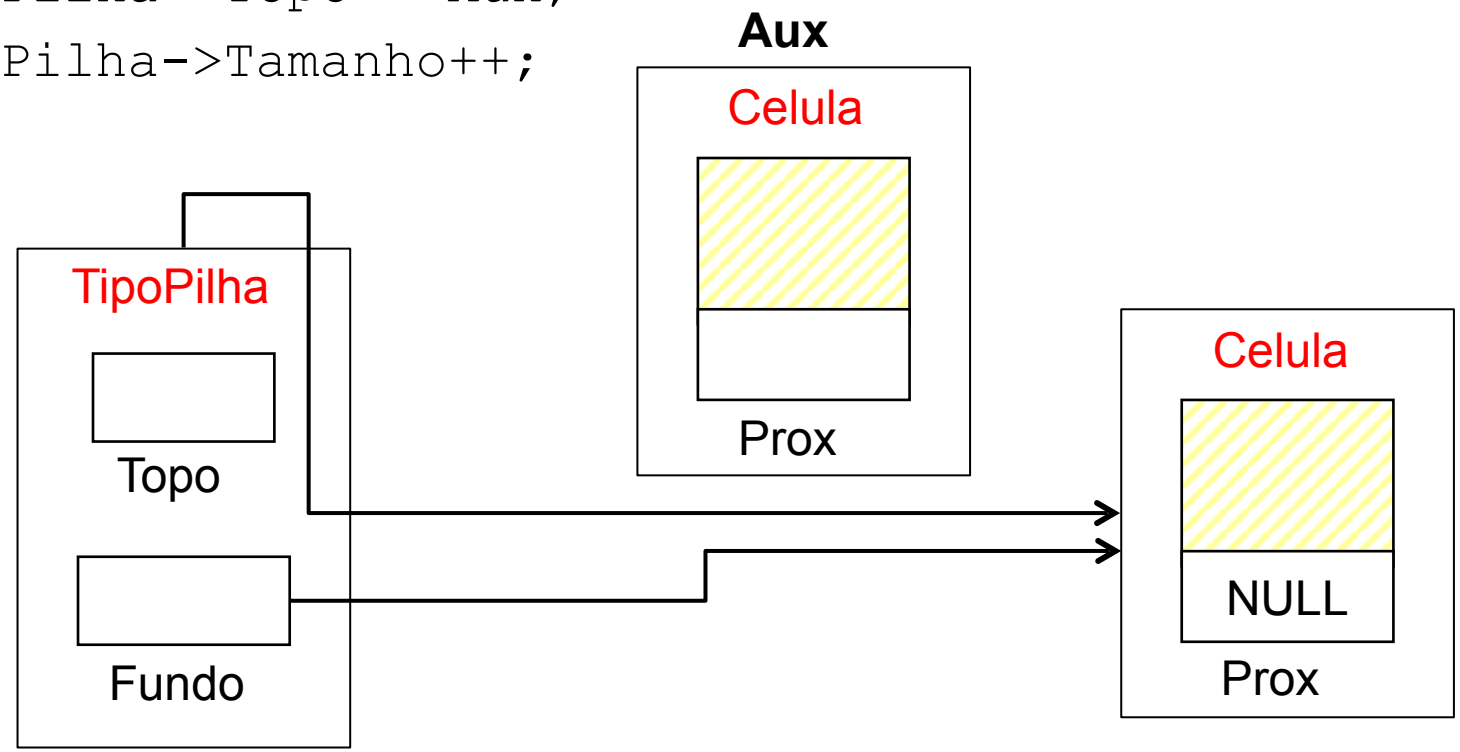


Operações sobre Pilhas Usando Apontadores

```
int Vazia(TipoPilha *Pilha)
{
  return (Pilha->Topo == Pilha->Fundo);
}  /* Vazia */
```

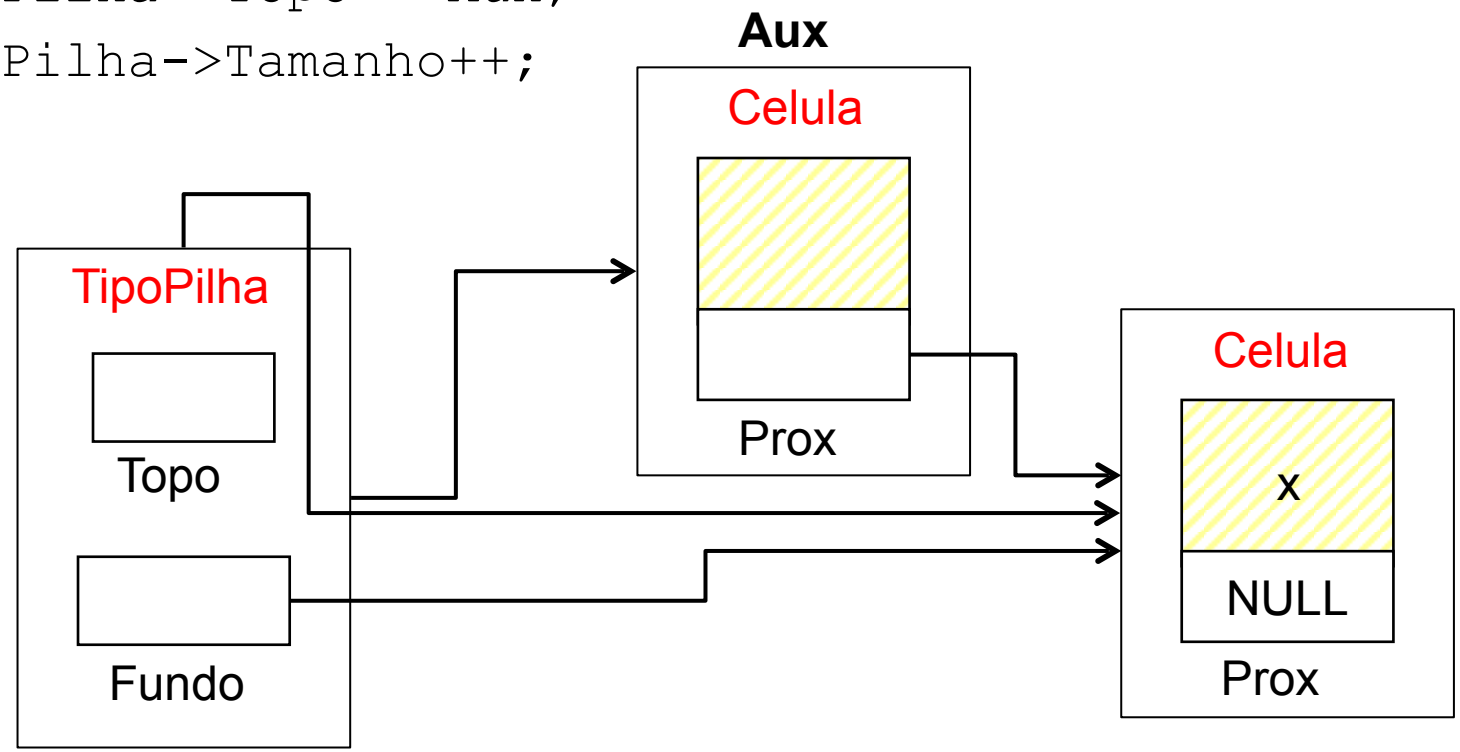
Operações sobre Pilhas Usando Apontadores

```
void Empilha(TipoItem x, TipoPilha *Pilha) {
  Apontador Aux;
  Aux = (Apontador) malloc(sizeof(Celula));
  Pilha->Topo->Item = x;
  Aux->Prox = Pilha->Topo;
  Pilha->Topo = Aux;
  Pilha->Tamanho++;
}
```



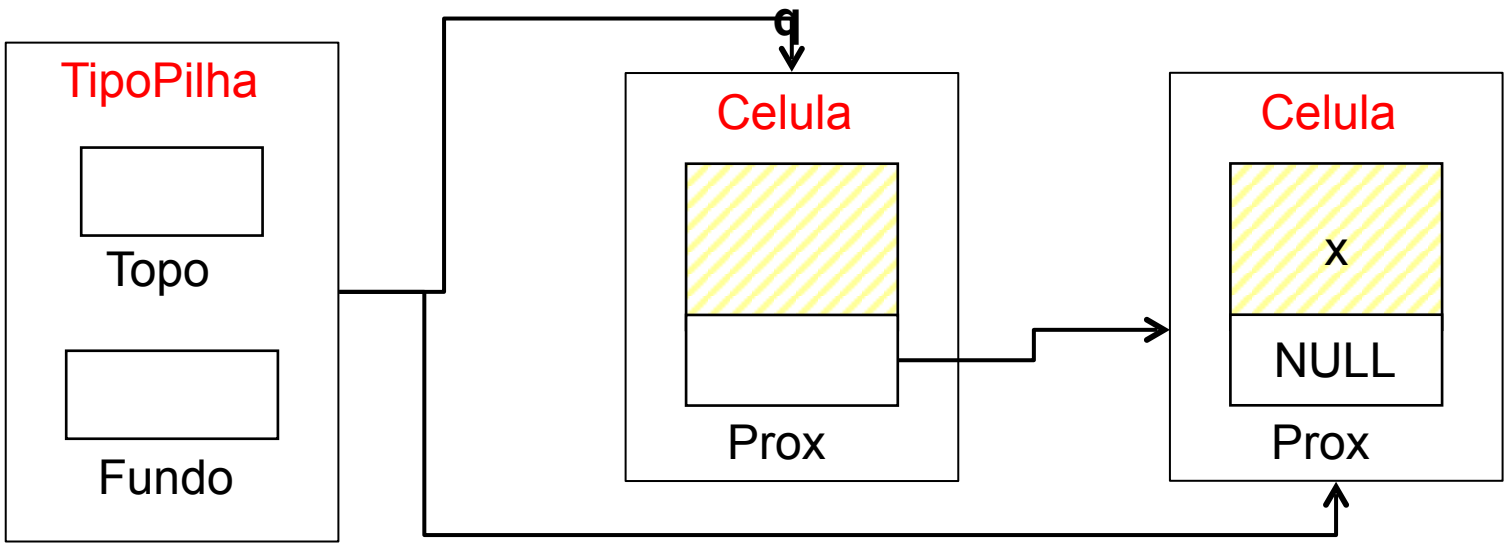
Operações sobre Pilhas Usando Apontadores

```
void Empilha(TipoItem x, TipoPilha *Pilha) {
  Apontador Aux;
  Aux = (Apontador) malloc(sizeof(Celula));
  Pilha->Topo->Item = x;
  Aux->Prox = Pilha->Topo;
  Pilha->Topo = Aux;
  Pilha->Tamanho++;
}
```



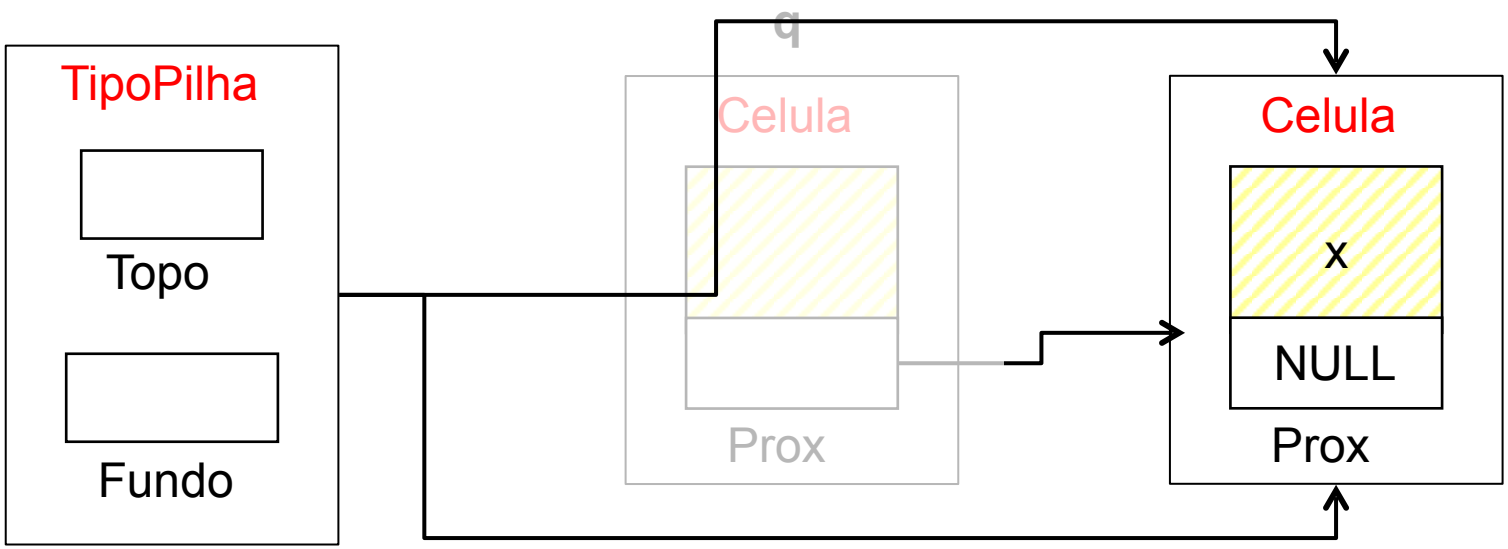
Operações sobre Pilhas Usando Apontadores

```
TipoItem Desempilha(TipoPilha *Pilha){
  Apontador q;
  if (Vazia(Pilha)) {
      printf("Erro: pilha vazia\n"); ERRO;
  }
  q = Pilha->Topo;
  Pilha->Topo = q->Prox;
  free(q);
  Pilha->Tamanho--;
  return Topo->Item;
}
```



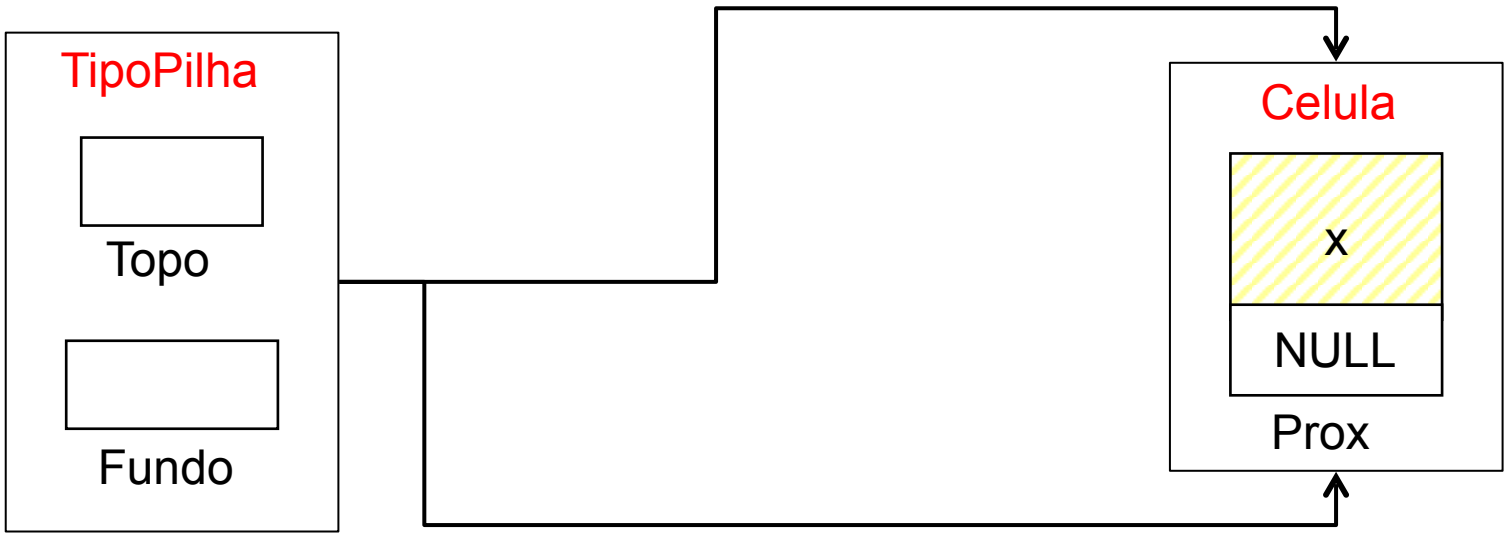
Operações sobre Pilhas Usando Apontadores

```
TipoItem Desempilha(TipoPilha *Pilha){
  Apontador q;
  if (Vazia(Pilha)) {
      printf("Erro: pilha vazia\n"); ERRO;
  }
  q = Pilha->Topo;
  Pilha->Topo = q->Prox;
  free(q);
  Pilha->Tamanho--;
  return Topo->Item;
}
```



Operações sobre Pilhas Usando Apontadores

```
TipoItem Desempilha(TipoPilha *Pilha){
  Apontador q;
  if (Vazia(Pilha)) {
      printf("Erro: pilha vazia\n"); ERRO;
  }
  q = Pilha->Topo;
  Pilha->Topo = q->Prox;
  free(q);
  Pilha->Tamanho--;
  return Topo->Item;
}
```

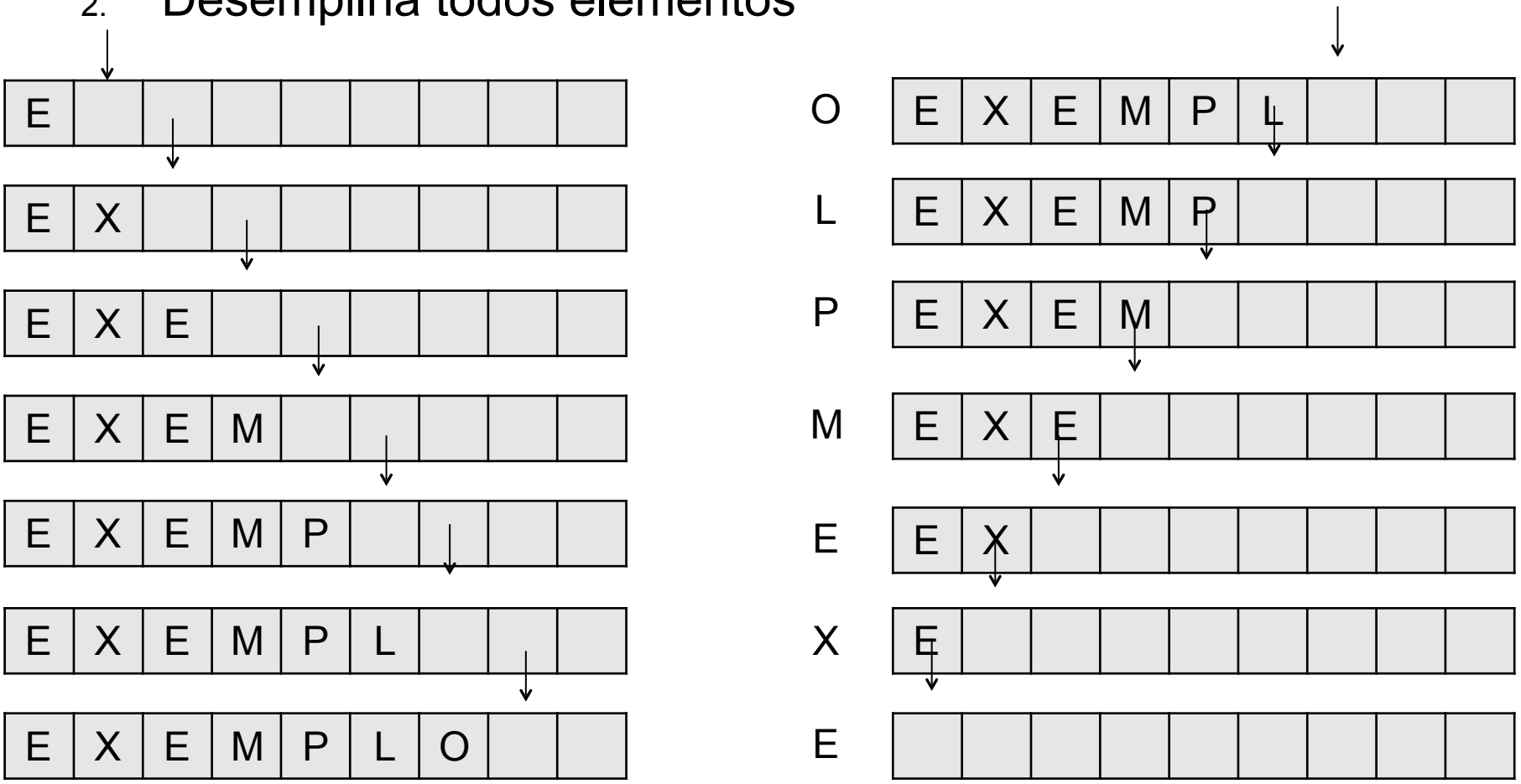


Operações sobre Pilhas Usando Apontadores

```
int Tamanho(TipoPilha *Pilha)
{
  return (Pilha->Tamanho);
}  /* Tamanho */
```

Exemplos: Pilhas
Inversão de strings

- Inverter a string “Exemplo” usando uma pilha.
  1. Empilha cada caractere em uma pilha vazia
  2. Desempilha todos elementos



Exemplos: Pilhas
Conversão notação infixada p/ pós-fixada

- Infixada: (`5 * (((9+8) * (4*6)) + 7))`
- Pós-fixada: `5 9 8 + 4 6 * * 7 + *` (tão logo encontre um operador, efetua a operação)
- Utilizar uma pilha para converter de infixada para pós-fixada.

```
typedef char TipoItem;
Converte(char *exp, TipoPilha *pilha){

  for (i = 0; i < N; i++) {
    if (exp[i] == ')')
      printf("%c ", desempilha(pilha))
    if (exp[i] == '+' || exp[i] == '*')
      empilha(exp[i], pilha);
    if (exp[i] >= '0' && exp[i] <= '9')
      printf("%c ", exp[i]);
}
```

Exemplos: Pilhas
Conversão notação infixada p/ pós-fixada

- entrada: (5 * (((9+8) * (4*6)) + 7))

Entrada	Pilha	saída
(		
5		5
*	*	
(((		
9		9
+	* +	
8		8
)	*	+
*	* *	
(		
4		4
*	* * *	
6		6
)	* *	*
)	*	*
+	* +	
7		7
)	*	+
)		*

saída: 5 9 8 + 4 6 * * 7 + *

TAD Filas

- Tipo Abstrato de dados com a seguinte característica:

O primeiro elemento a ser inserido é o primeiro a ser retirado

(FIFO – *First In First Out)*

- Analogia: fila bancária, fila do cinema

- Usos: Sistemas operacionais: fila de impressão, processamento

TAD Filas

- **Conjunto de operações:**
  1) FFVazia(Fila). Faz a fila ficar vazia.
  2) Enfileira(x, Fila). Insere o item x no final da fila.
  3) Desenfileira(Fila). Retorna o item x no início da fila, retirando-o da fila.
  4) Vazia(Fila). Esta função retorna true se a fila está vazia; senão retorna false.

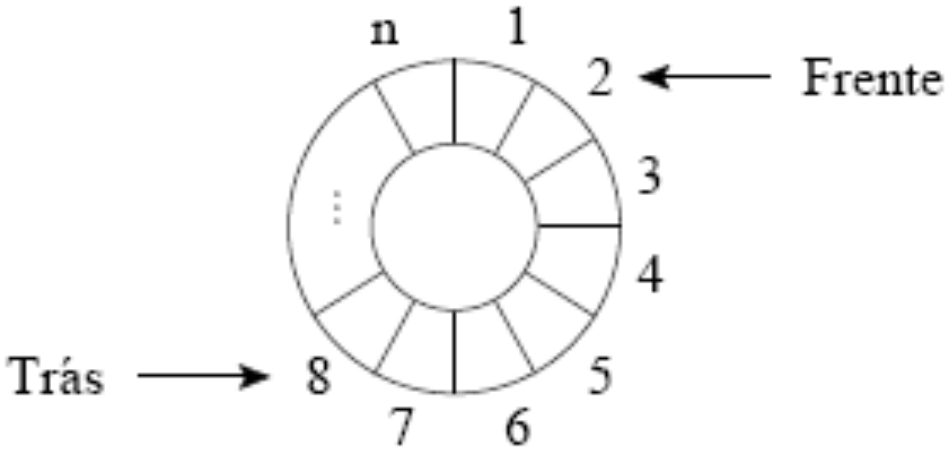
Implementação de Filas com alocação sequencial Arranjos

- Os itens são armazenados em **posições contíguas** de memória.
- **Enfileira:** faz a parte de trás da fila expandir-se.
- **Desenfileira:** faz a parte da frente da fila contrair-se.
- A fila tende a se movimentar pela memória do computador, **ocupando espaço na parte de trás e descartando espaço na parte da frente.**

Implementação de Filas com alocação sequencial

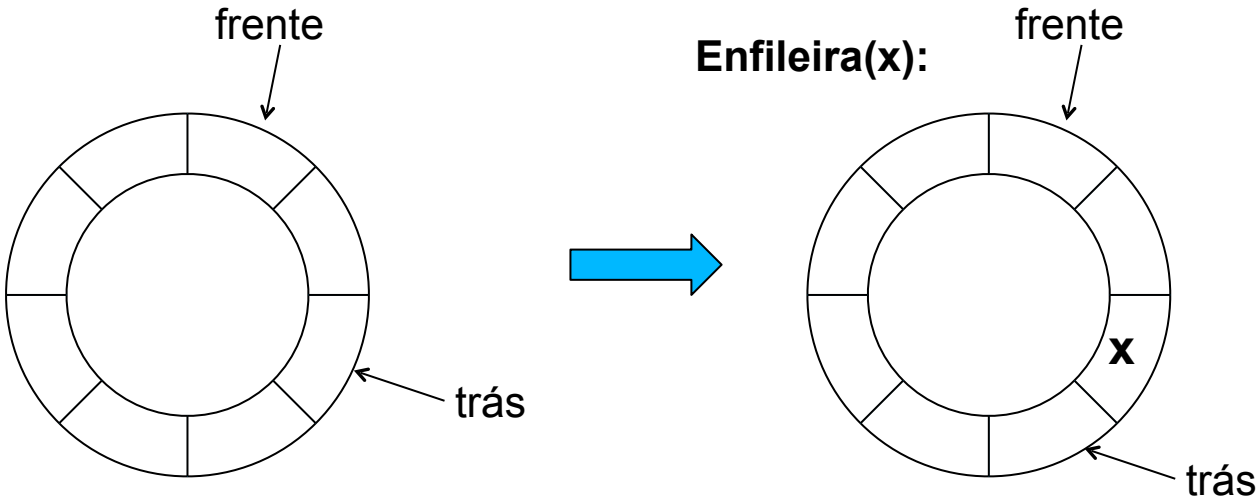
- Com poucas inserções e retiradas, a fila vai ao encontro do limite do espaço da memória alocado para ela.

- **Solução**: imaginar o array como um círculo. A primeira posição segue a última.



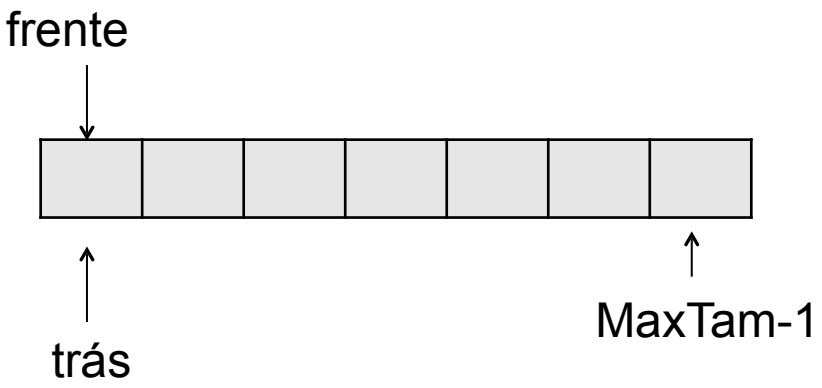
Implementação de Filas com alocação sequencial

- A fila se encontra em posições contíguas de memória, em alguma posição do círculo, delimitada pelos apontadores Frente e Trás. (**Frente**: posição do primeiro elemento, **trás**: a primeira posição vazia)
- **Enfileirar:** mover o apontador Trás uma posição no sentido horário.
- **Desenfileirar:** mover o apontador Frente uma posição no sentido horário.



Estrutura da Fila com Alocação Sequencial

```
#define MaxTam  1000

typedef int Apontador;
typedef int TipoChave;

typedef struct {
  TipoChave Chave;
  /* outros componentes */
} TipoItem;

typedef struct {
  TipoItem Item[MaxTam];
  Apontador Frente, Tras;
} TipoFila;
```



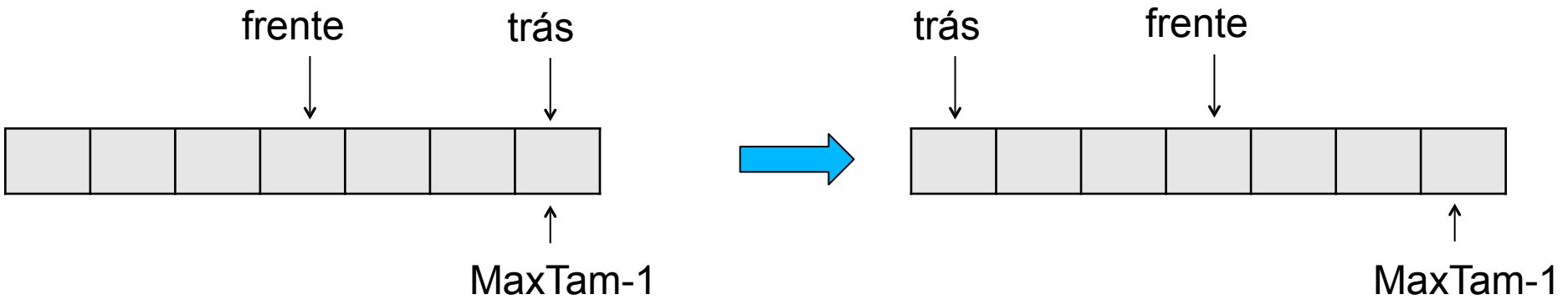
Operações sobre Filas com Alocação Sequencial

- Nos casos de fila cheia e fila vazia, os apontadores Frente e Trás apontam para a mesma posição do círculo.
- Uma saída para distinguir as duas situações é deixar uma posição vazia no array.
- Neste caso, a fila está cheia quando Trás+1 for igual a Frente.

```
void FFVazia(TipoFila *Fila)
{
  Fila->Frente = 0;
  Fila->Tras = Fila->Frente;
}  /* FFVazia */

int Vazia(TipoFila *Fila)
{
  return (Fila->Frente == Fila->Tras);
}  /* Vazia */
```

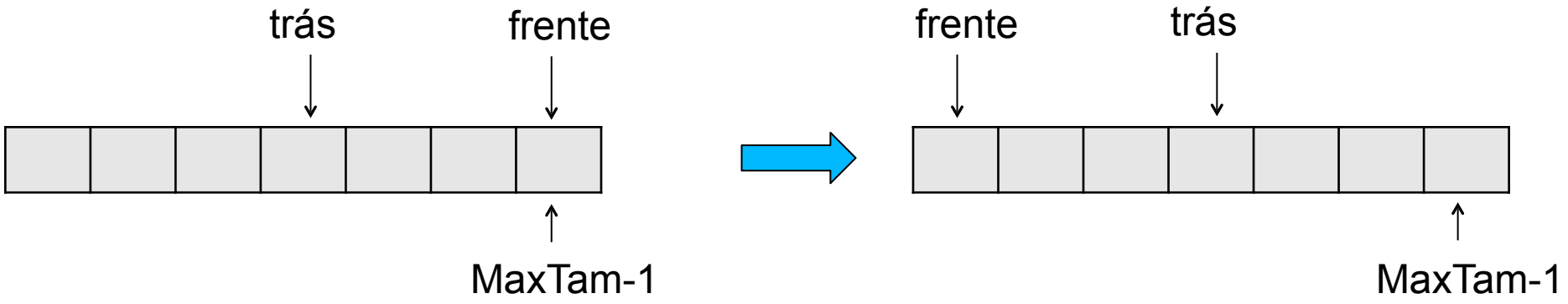
Operações sobre Filas com Alocação Sequencial

```
int Enfileira(TipoItem x, TipoFila *Fila) {
  if ((Fila->Tras + 1) % MaxTam == Fila->Frente){
     printf("Erro: fila está cheia\n"); return 0;
  }
  else {
     Fila->Item[Fila->Tras] = x;
     Fila->Tras = (Fila->Tras + 1) % MaxTam;
  }
  return 1;
}
```



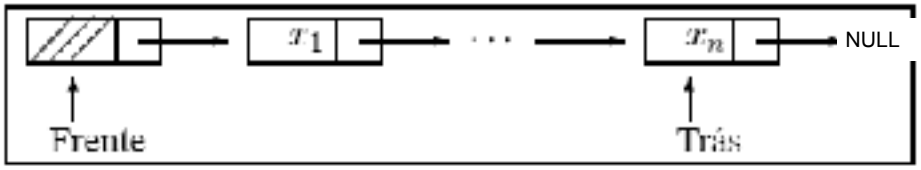
Operações sobre Filas com Alocação Sequencial

```
TipoItem Desenfileira(TipoFila *Fila) {
  if (Vazia(Fila)){
      printf("Erro: fila está vazia\n"); ERRO;
  }
  else {
      int idx = Fila->Frente;
      Fila->Frente = (Fila->Frente + 1) % MaxTam;
      return Fila->Item[idx];
  }
}
```



Implementação de Filas por meio de Apontadores

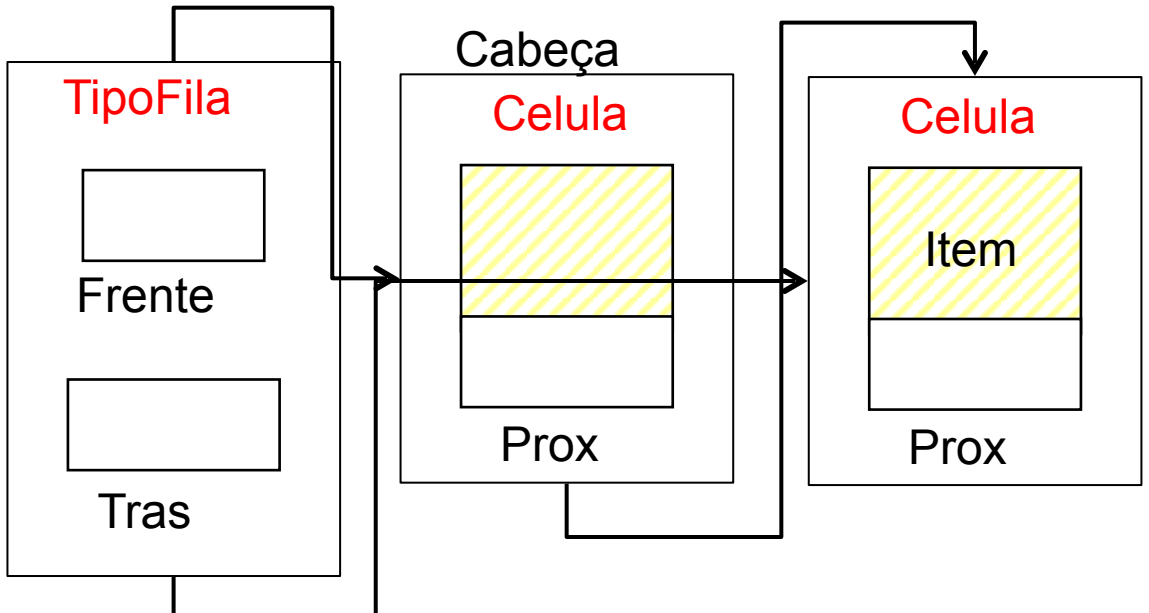
- Utiliza célula cabeça para facilitar a implementação das operações Enfileira e Desenfileira quando a fila está vazia.
- **Quando vazia**: apontadores Frente e Trás apontam para a célula cabeça.
- **Enfileirar novo item**: criar uma célula nova, ligá-la após a célula contendo x_n e colocar nela o novo item.
- **Desenfileirar**: desligar a célula cabeça da lista e a célula que contém x_1 passa a ser a célula cabeça.



Estrutura da Fila Usando Apontadores

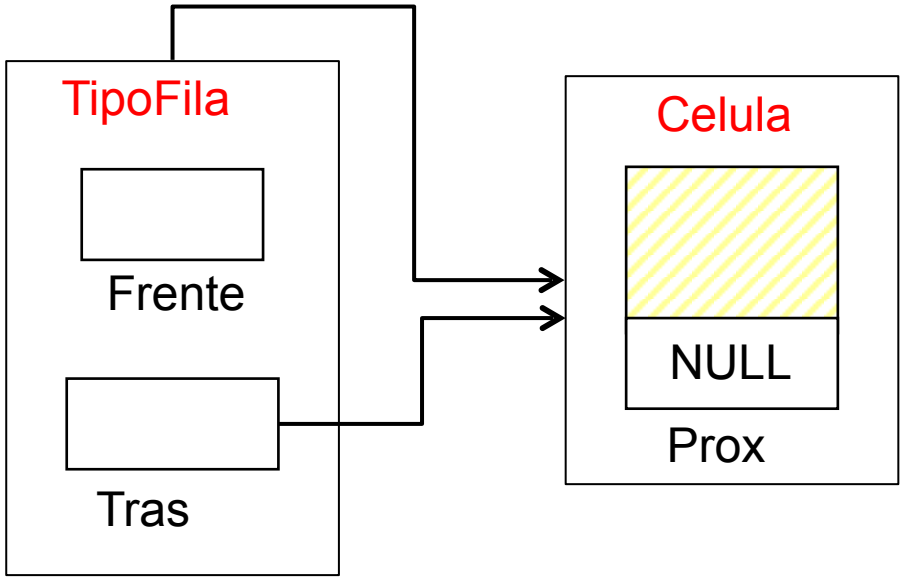
```
typedef int TipoChave;

typedef struct TipoItem {
  TipoChave Chave;
  /* outros componentes */
} TipoItem;

typedef struct Celula *Apontador;

typedef struct Celula {
  TipoItem Item;
  Apontador Prox;
} Celula;

typedef struct TipoFila {
  Apontador Frente, Tras;
} TipoFila;
```

- A fila é implementada por meio de células.
- Cada célula contém um item da fila e um apontador para outra célula.
- A estrutura TipoFila contém um apontador para a frente da fila (célula cabeça) e um apontador para a parte de trás da fila.



Operações sobre Filas Usando Apontadores

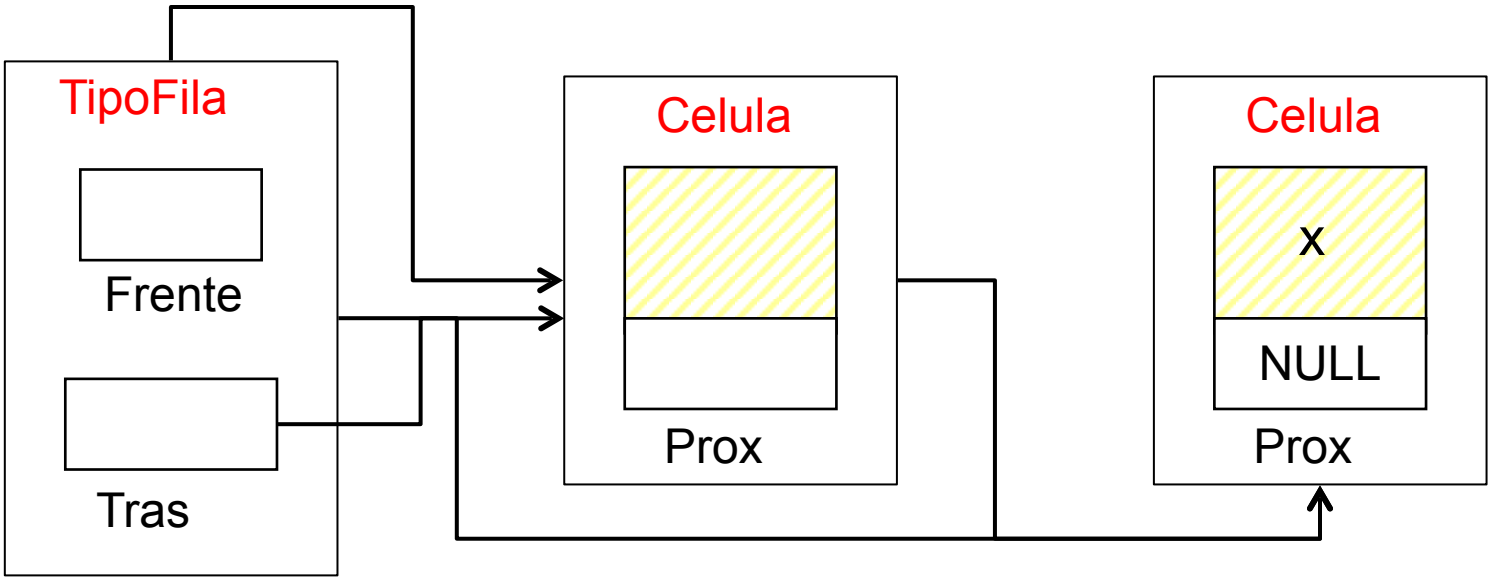
```
void FFVazia(TipoFila *Fila)
{
  Fila->Frente = (Apontador) malloc(sizeof(Celula));
  Fila->Tras = Fila->Frente;
  Fila->Frente->Prox = NULL;
}  /* FFVazia */
```




```
int Vazia(TipoFila *Fila)
{
  return (Fila->Frente == Fila->Tras);
}  /* Vazia */
```

Operações sobre Filas Usando Apontadores

```
void Enfileira(TipoItem x, TipoFila *Fila)
{
  Fila->Tras->Prox = (Apontador) malloc(sizeof(Celula));
  Fila->Tras = Fila->Tras->Prox;
  Fila->Tras->Item = x;
  Fila->Tras->Prox = NULL;
}
```



Operações sobre Filas Usando Apontadores

```
TipoItem Desenfileira(TipoFila *Fila){
  Apontador q;
  if (Vazia(Fila)) {
    printf("Erro: fila está vazia\n"); ERRO;
  }
  q = Fila->Frente;
  Fila->Frente = Fila->Frente->Prox;
  free(q);
  return Fila->Frente->Item;
}
```