

# ESTRUTURA DE DADOS

Prof.<sup>a</sup> Priscilla Abreu

[priscilla.braz@rj.senac.br](mailto:priscilla.braz@rj.senac.br)



# Estrutura de dados



## Roteiro de Aula

- Objetivo da aula
- Listas
  - Fila sequencial

# Estrutura de dados



## Objetivo da aula

Manipular listas lineares com restrições de acesso.



## **Competência:**

Desenvolver estruturas de dados lineares e não lineares.

# REVISANDO...

# Estrutura de dados



## LISTA LINEAR

### Listas lineares

#### Listas lineares gerais

SEM restrição de inserção e remoção de elementos

#### Listas particulares

COM restrição de inserção e remoção de elementos

# Estrutura de dados



## LISTA LINEAR

### Casos particulares:

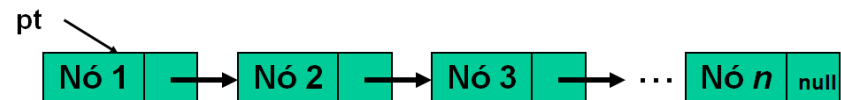
- Deque  
Inserção e remoção apenas nas extremidades;
- Pilha  
Inserção e remoção apenas em um extremo
- Fila  
Inserção em um extremo e remoção em outro extremo;

## LISTA LINEAR: TIPO DE ARMAZENAMENTO

O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória (contígua ou não) de cada dois nós consecutivos na lista.

Existem dois tipos de listas:

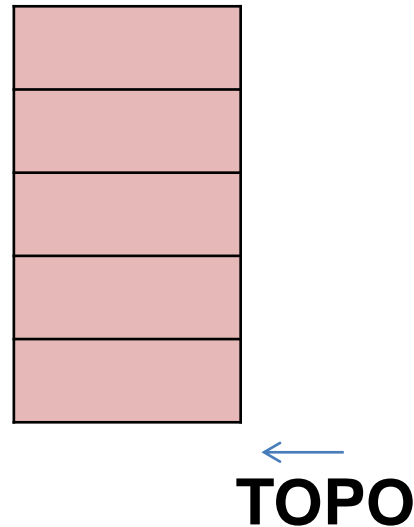
- Lista sequencial
- Lista encadeada



# Estrutura de dados



## PILHA SEQUENCIAL



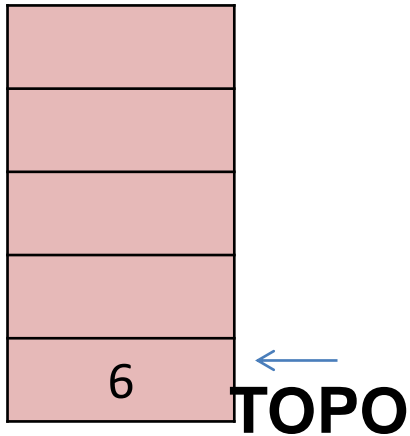


# Estrutura de dados

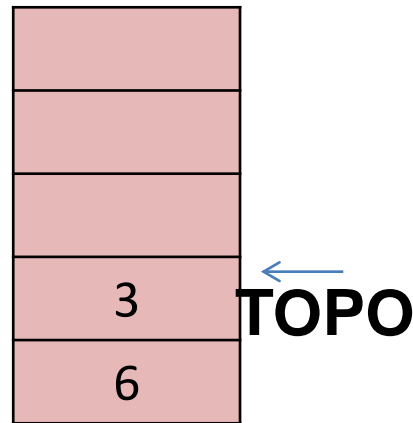


## PILHA SEQUENCIAL

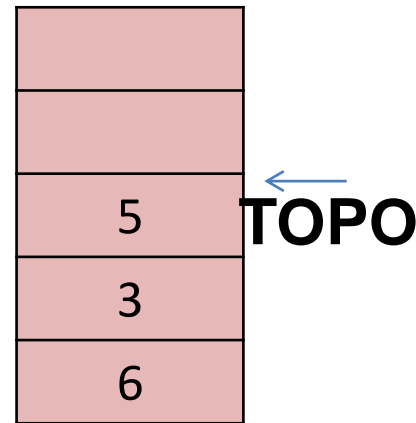
**EMPILHA (6)**



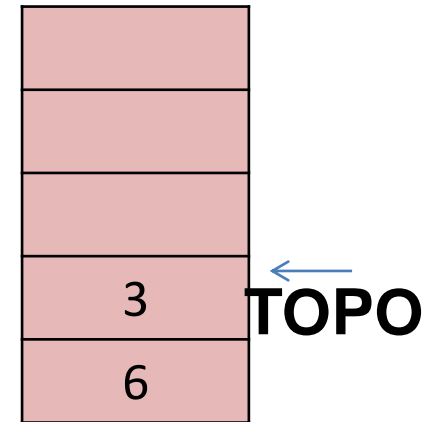
**EMPILHA (3)**



**EMPILHA (5)**



**DESEMPILHA ()**



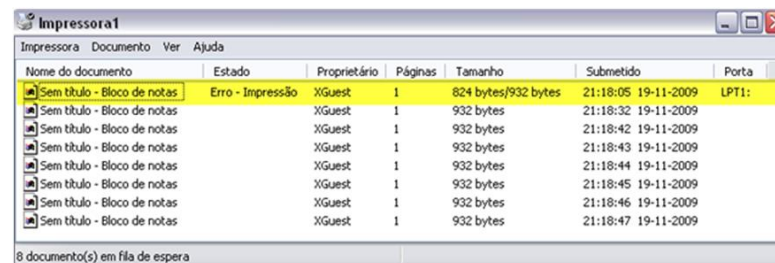
# FILAS SEQUENCIAIS

# Estrutura de dados



## FILAS

- São listas em que todas as inserções ocorrem em uma extremidade e as remoções em outra extremidade;
- Estruturas de dados do tipo FIFO (first-in first-out): o primeiro elemento a ser inserido, será o primeiro a ser removido.
- Exemplos: filas de banco, supermercado, fila de impressão de arquivos , etc.



Nome do documento	Estado	Proprietário	Páginas	Tamanho	Submetido	Porta
Sem título - Bloco de notas	Erro - Impressão	XGuest	1	824 bytes/932 bytes	21:18:05 19-11-2009	LPT1:
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:32 19-11-2009	
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:42 19-11-2009	
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:43 19-11-2009	
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:44 19-11-2009	
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:45 19-11-2009	
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:46 19-11-2009	
Sem título - Bloco de notas		XGuest	1	932 bytes	21:18:47 19-11-2009	

8 documento(s) em fila de espera



# Estrutura de dados



## FILAS



**INÍCIO**



**FIM**

Início da fila: extremidade onde ocorrem as remoções.

Final da fila: extremidade onde ocorrem as inserções.

# Estrutura de dados



## FILAS – APLICAÇÕES

- Fila de arquivos para impressão;
- Atendimento de processos requisitados ao um sistema operacional;
- Processos de reserva e compra online;
- Buffer para gravação de dados em mídia;
- Processos de comunicação em redes de computadores.

# Estrutura de dados



## FILAS – OPERAÇÕES

- Alocação sequencial:
  - Uso de vetores;
  - Variáveis controladoras para inicio e fim da fila:
    - Inicio e fim.
- Operações básicas:
  - Inserção
  - Remoção
- Situações extremas:
  - Fila cheia
  - Fila vazia

# Estrutura de dados



## FILA CIRCULAR

$i \Rightarrow$  posição do elemento que está no início da fila;

$f \Rightarrow$  posição do último elemento inserido na fila;

$n \Rightarrow$  número de elementos na fila.

$M \Rightarrow$  tamanho máximo do vetor que representa a fila.

$\text{fila} \Rightarrow$  vetor que representa a fila.

# Estrutura de dados



## FILA – IMPLEMENTAÇÃO

```
int main(){  
    int fila[M];  
    int i, f, n;  
    i = -1; f= -1, n=0;  
  
    ...  
}
```

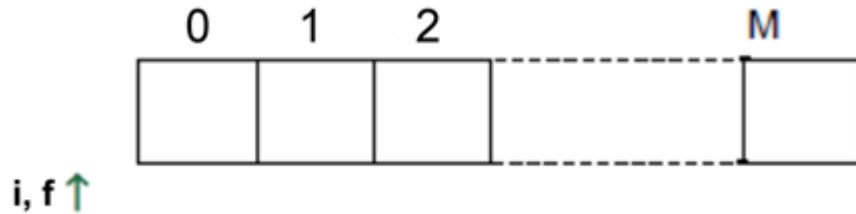


# Estrutura de dados



## FILAS – SIMULAÇÃO

### Situação 1: Fila vazia



**Fila: vetor de tamanho M**

**i: início da fila**

**f: final da fila**

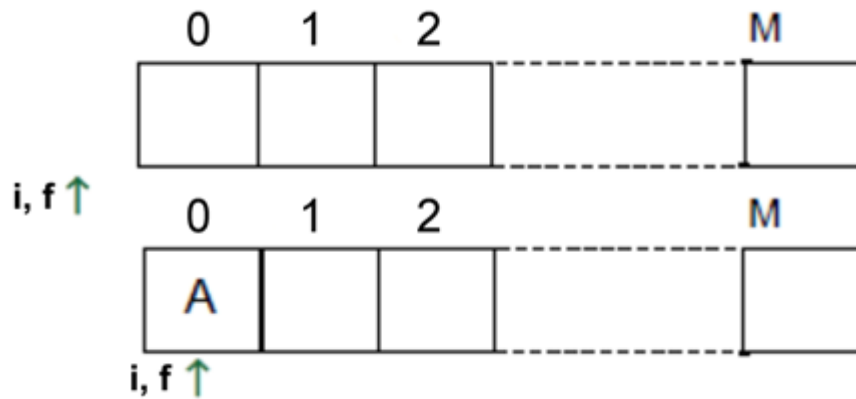
**$i = f = -1$**

# Estrutura de dados



## FILAS – SIMULAÇÃO

### Situação 2: Insere(A)



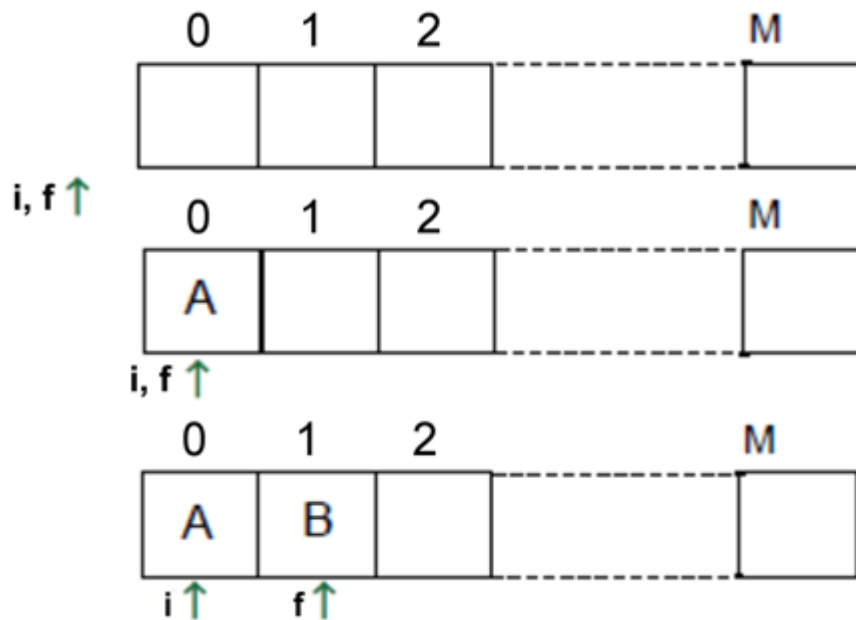
**Fila:** vetor de tamanho M  
**i:** início da fila  
**f:** final da fila

# Estrutura de dados



## FILAS – SIMULAÇÃO

### Situação 3: Insere(B)



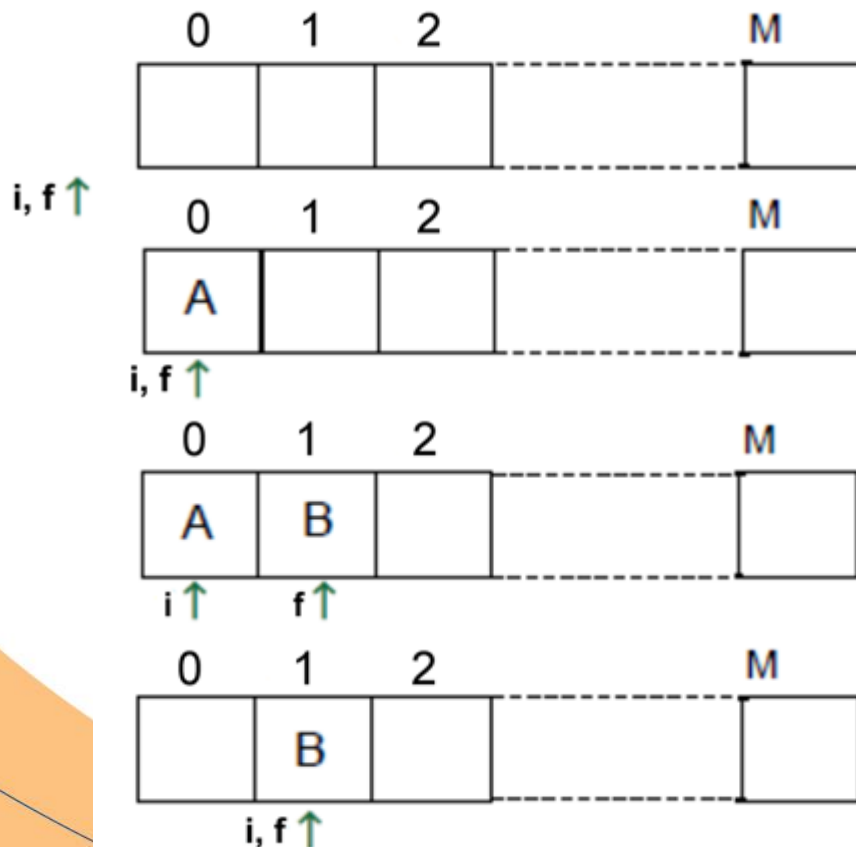
Fila: vetor de tamanho M  
i: início da fila  
f: final da fila

# Estrutura de dados



## FILAS – SIMULAÇÃO

### Situação 4: Remove()



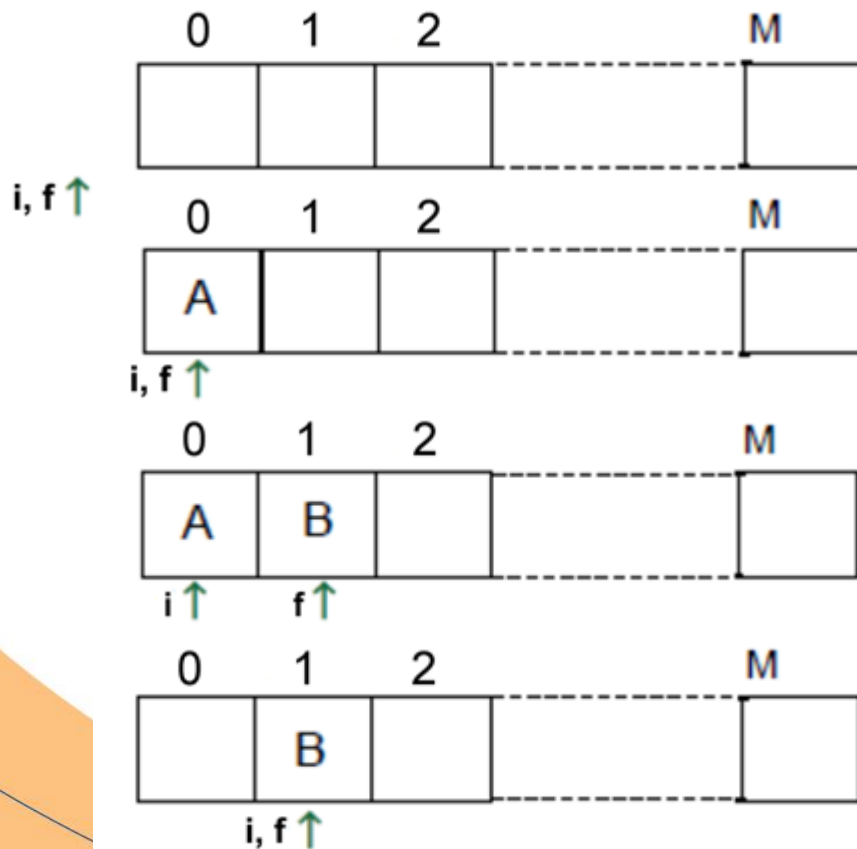
**Fila:** vetor de tamanho  $M$   
**i:** início da fila  
**f:** final da fila

# Estrutura de dados

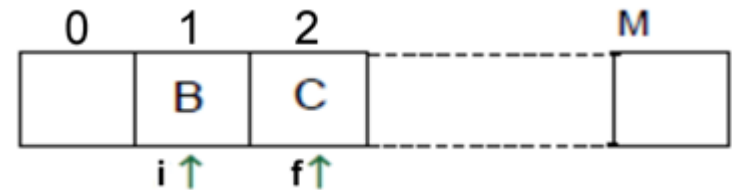


## FILAS – SIMULAÇÃO

### Situação 5: Insere(C)



Fila: vetor de tamanho M  
i: início da fila  
f: final da fila

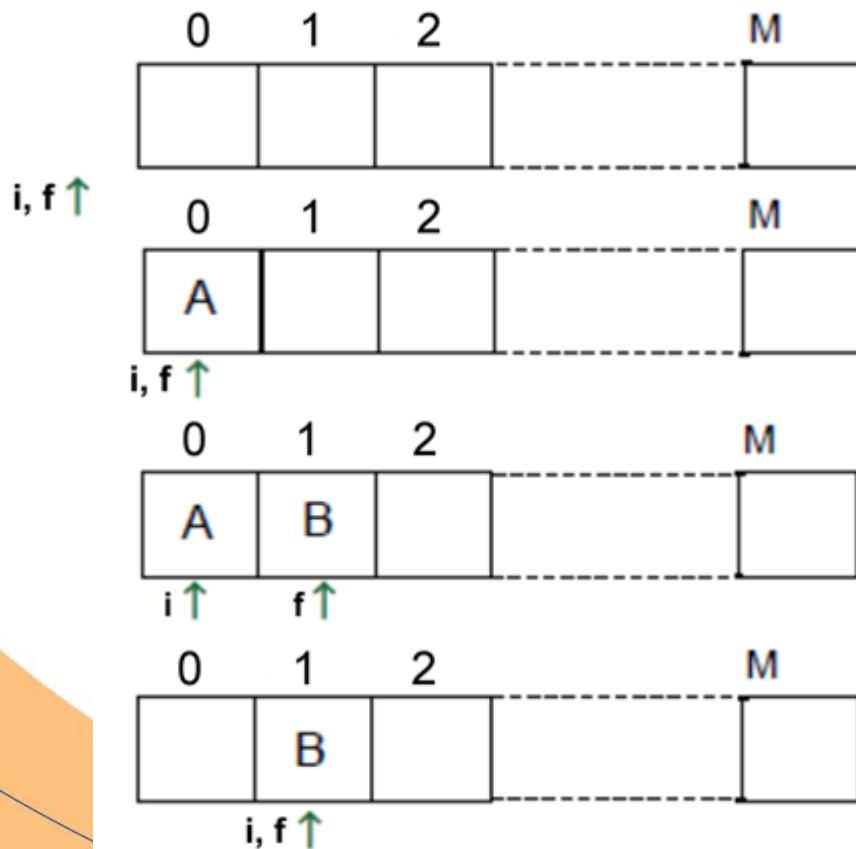


# Estrutura de dados

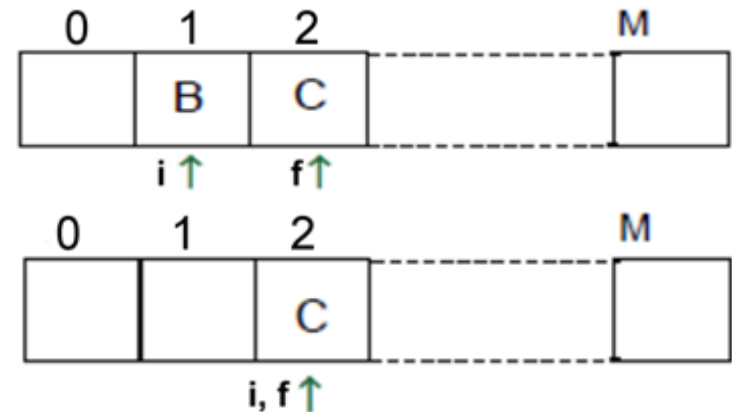


## FILAS – SIMULAÇÃO

### Situação 6: Remove()



Fila: vetor de tamanho  $M$   
 $i$ : início da fila  
 $f$ : final da fila

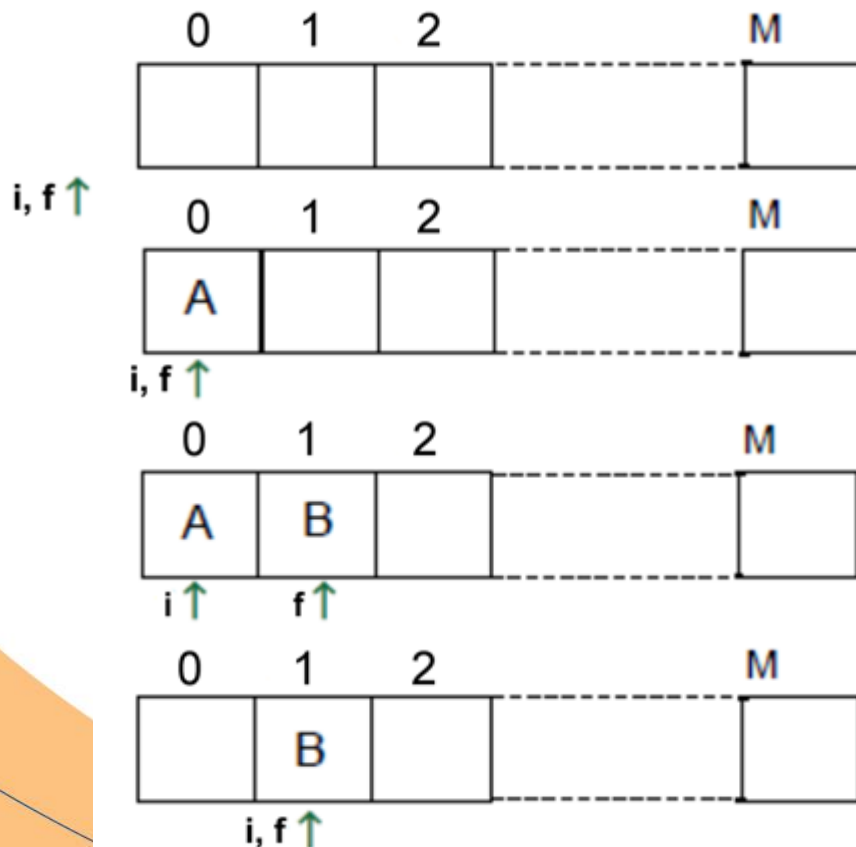


# Estrutura de dados

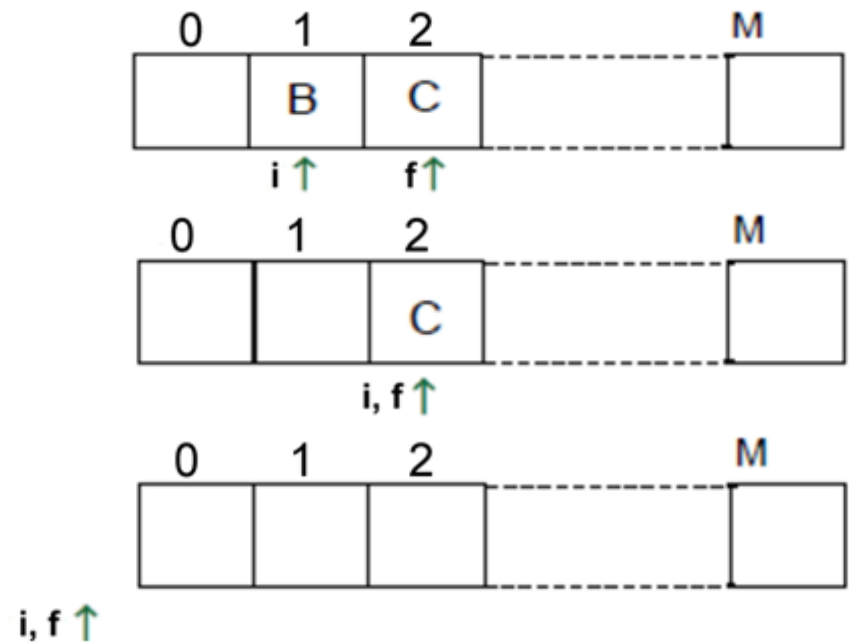


## FILAS – SIMULAÇÃO

### Situação 7: Remove()

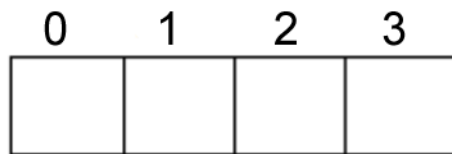


Fila: vetor de tamanho M  
i: início da fila  
f: final da fila

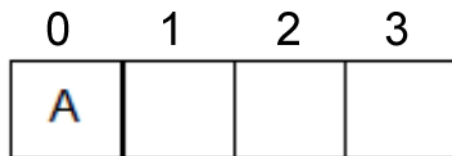


# Estrutura de dados

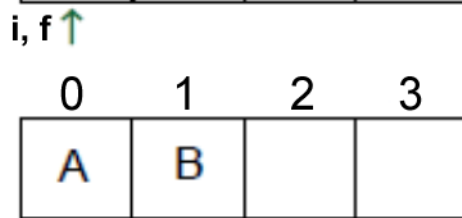
## FILAS – SIMULAÇÃO



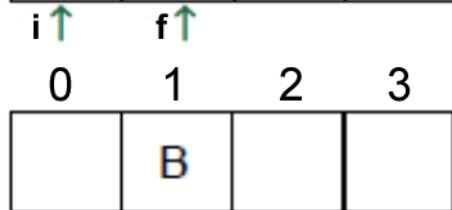
Fila vazia



Inserir A

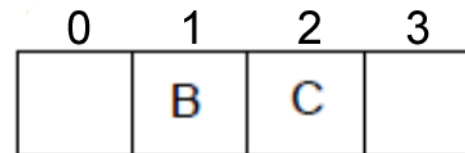


Inserir B

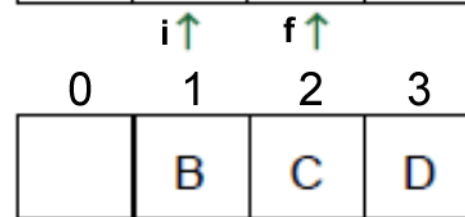


Remove

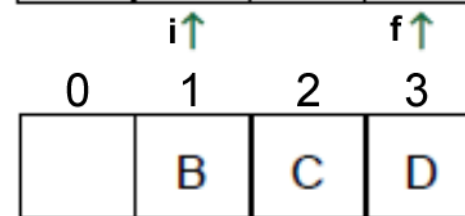
$i, f \uparrow$



Inserir C



Inserir D

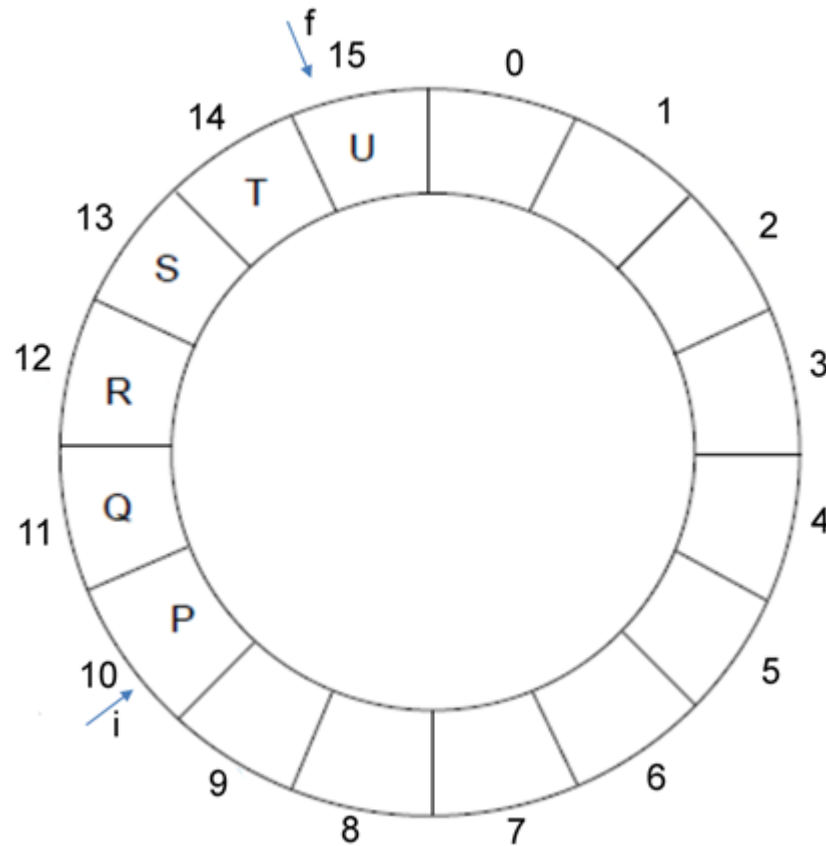


Inserir E???



# Estrutura de dados

## FILA CIRCULAR



# Estrutura de dados



## FILA CIRCULAR

$i \Rightarrow$  posição do elemento que está no início da fila;

$f \Rightarrow$  posição do último elemento inserido na fila;

$n \Rightarrow$  número de elementos na fila.

$M \Rightarrow$  tamanho máximo do vetor.

# Estrutura de dados

## FILA CIRCULAR – ENFILEIRAR

O que fazer para  
enfileirar???

$n = 0$

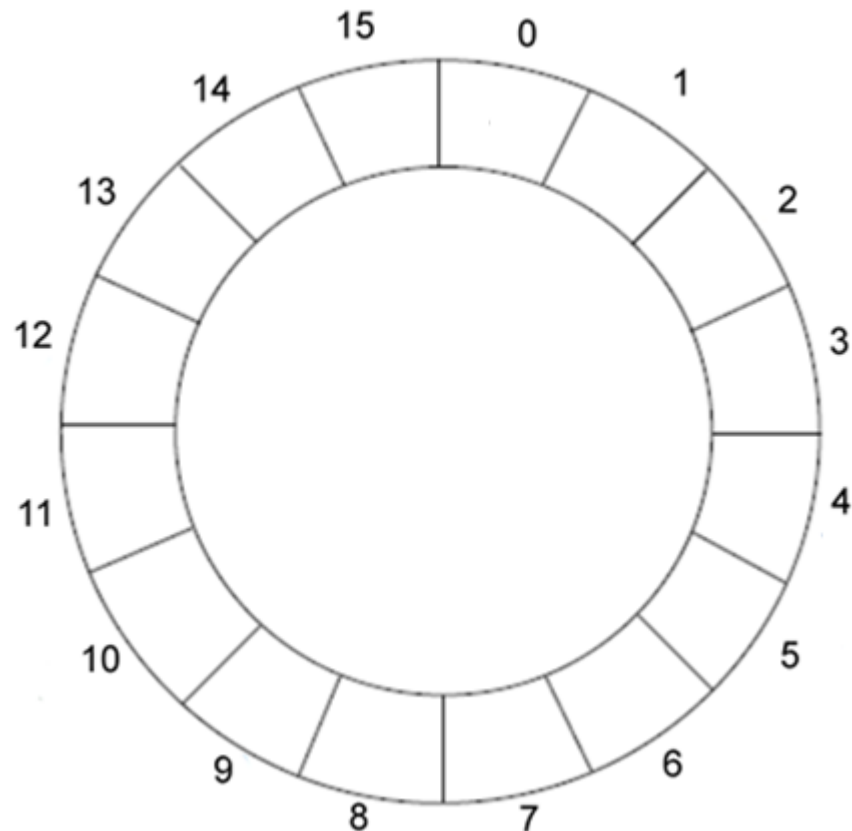
$i = -1$

$f = -1$

$M = 16$

**$f++;$**

**$n++;$**



# Estrutura de dados



## FILA CIRCULAR – ENFILEIRAR

$n = 1$

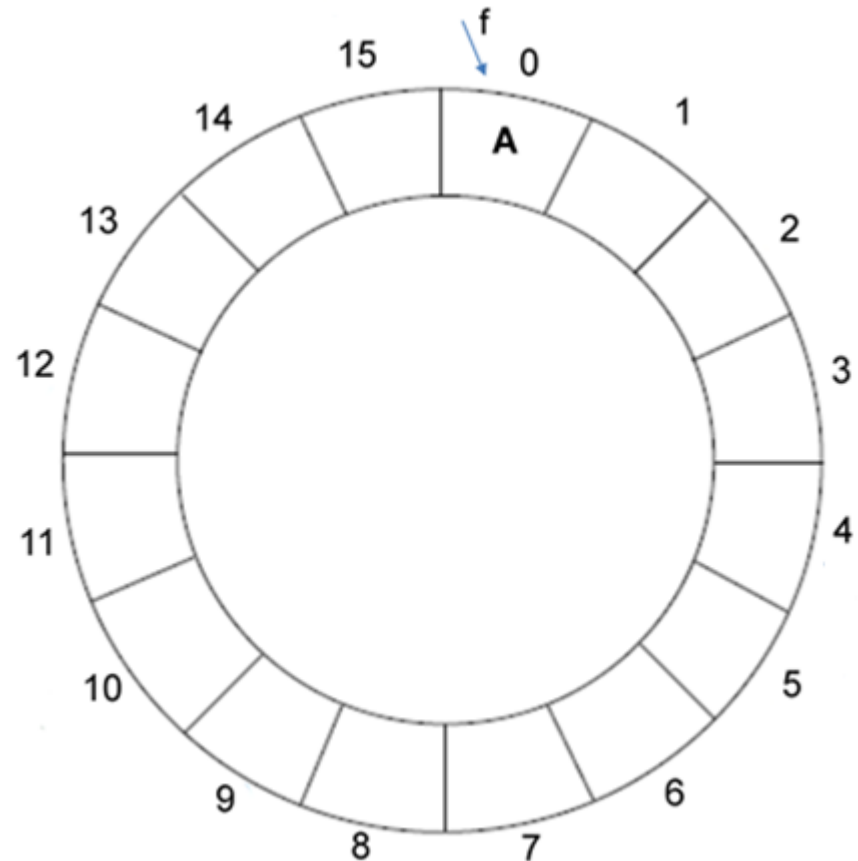
$i = -1$

$f = 0$

$M = 16$

**$\text{fila}[f] = \text{valor};$**

**$i++$**



# Estrutura de dados

## FILA CIRCULAR – ENFILEIRAR...

$n = 1$

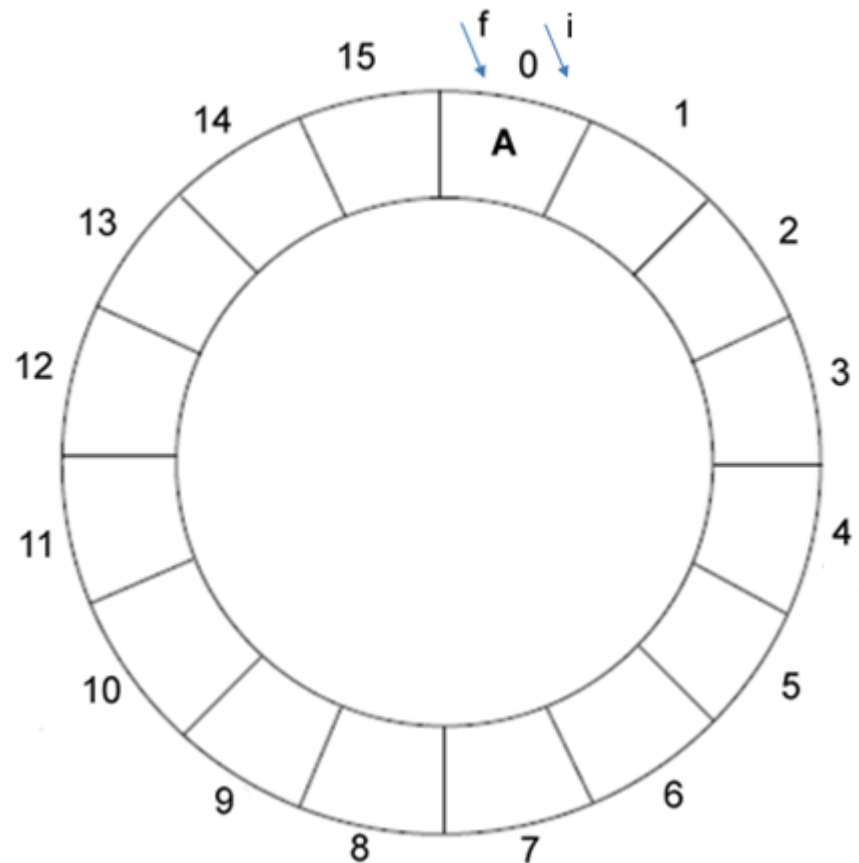
$i = 0$

$f = 0$

$M = 16$

**Inserindo  
novo valor**

**$f++;$**



# Estrutura de dados

## FILA CIRCULAR – ENFILEIRAR...

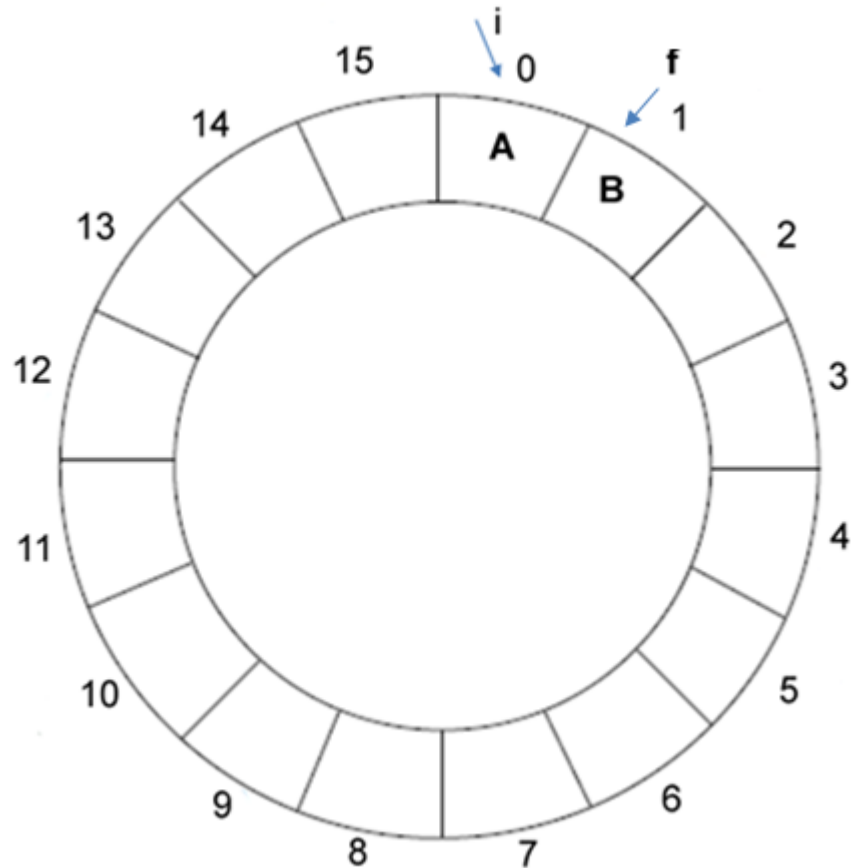
$n = 1$

$i = 0$

$f = 0$

$M = 16$

**$\text{fila}[f] = \text{valor};$**



# Estrutura de dados

## FILA CIRCULAR – ENFILEIRAR

$n = 5$

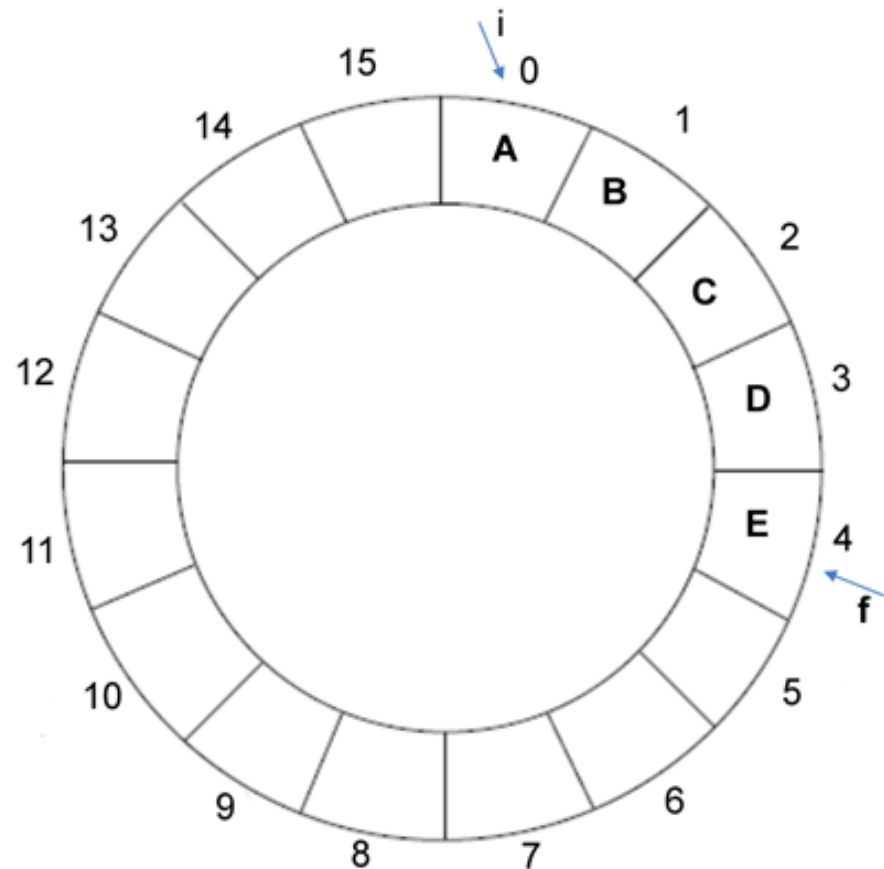
$i = 0$

$f = 4$

$M = 16$

Após algumas  
inserções...

$f ++;$



# Estrutura de dados



## FILA CIRCULAR – ENFILEIRAR

$n = 6$

$i = 10$

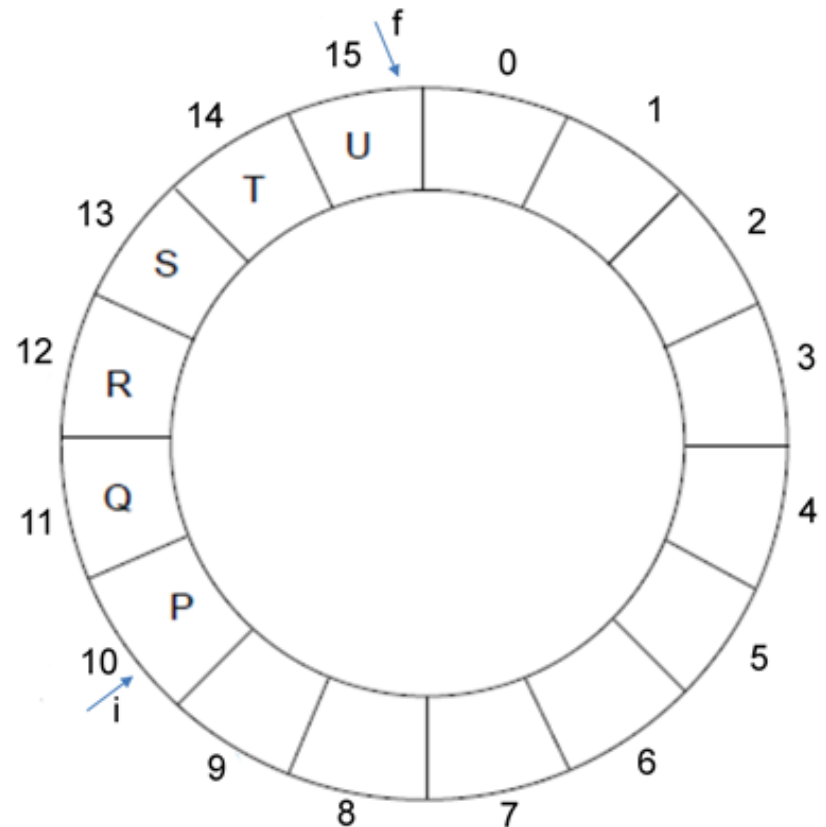
$f = 15$

$M = 16$

**Mais inserções  
e remoções...**

**Como enfileirar?**

**$f++$  ???**





# Estrutura de dados

## FILA CIRCULAR – ENFILEIRAR

$n = 6$

$i = 10$

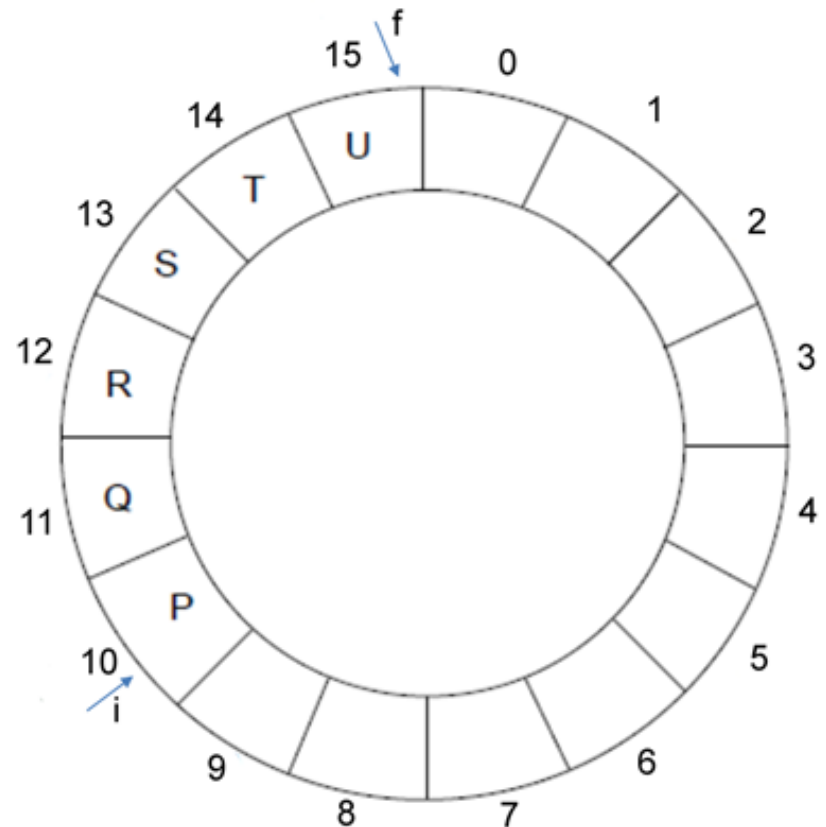
$f = 15$

$M = 16$

**Mais inserções  
e remoções...**

**Como enfileirar?**

~~**$f++$  ???**~~



# Estrutura de dados

## FILA CIRCULAR – ENFILEIRAR

$n = 6$

$i = 10$

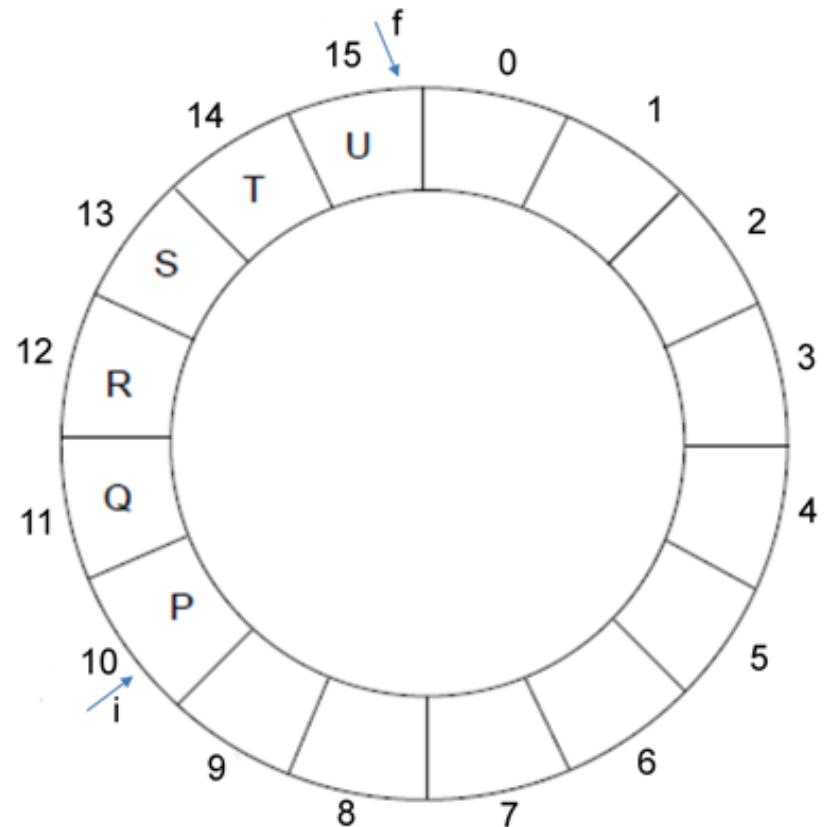
$f = 15$

$M = 16$

**$f == M-1$  ?**



**$f = 0;$**



# Estrutura de dados

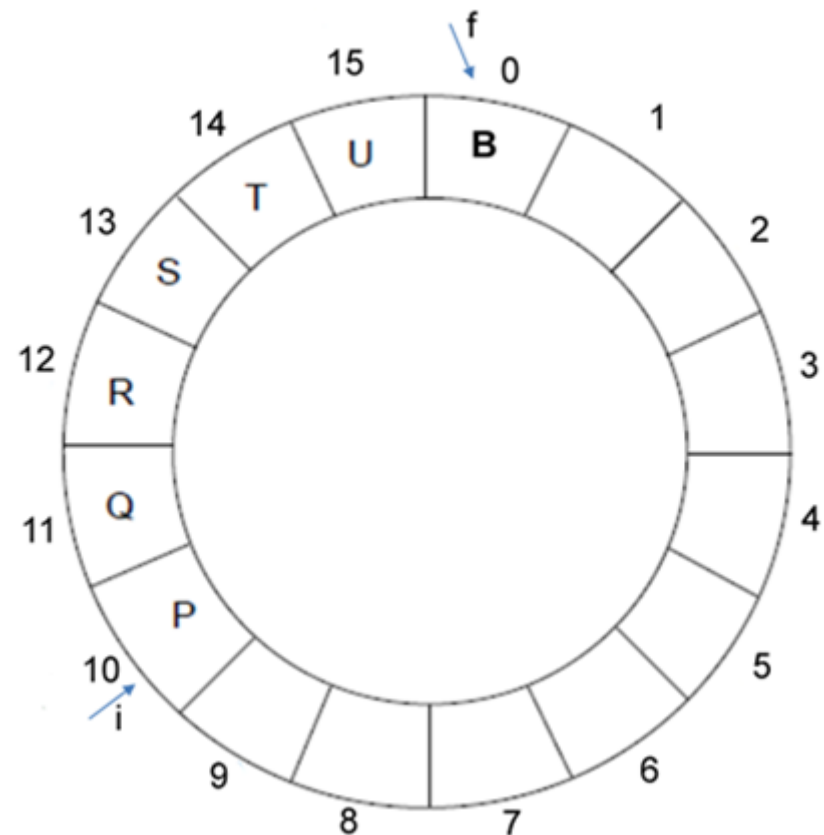
## FILA CIRCULAR – ENFILEIRAR

$n = 7$

$i = 10$

$f = 0$

$M = 16$



# Estrutura de dados



## FILA CIRCULAR

### Enfileirar

M -> tamanho do vetor

n -> quantidade de elementos armazenados

i -> início da fila

f -> final da fila

valor -> valor a ser inserido

# Estrutura de dados



## FILA CIRCULAR

## FILA SEQUENCIAL - IMPLEMENTAÇÃO

```
#include <stdio.h>
#define M 10
int fila_vazia(int n){
    if( n==0 )
        return 1;
    return 0;
}
```

# Estrutura de dados



## FILA CIRCULAR

## FILA SEQUENCIAL - IMPLEMENTAÇÃO

```
int fila_cheia(int n){  
    if( n==M )  
        return 1;  
    return 0;  
}
```

# Estrutura de dados



## FILA CIRCULAR – ENFILEIRAR

```
void enfileirar(int *i, int *f, int valor, int *n, int *fila ){
    if(!fila_cheia(*n)){
        if(*f==M-1)
            *f=0;
        else
            (*f)++;
        fila[*f]=valor;
        if((*i)==-1)
            (*i)++;
        (*n)++;
        printf("\nValor enfileirado!\n");
    }
    else
        printf("\nFila cheia!\n");
}
```

# Estrutura de dados

## FILA CIRCULAR – DESENFILAR

$i = 0$

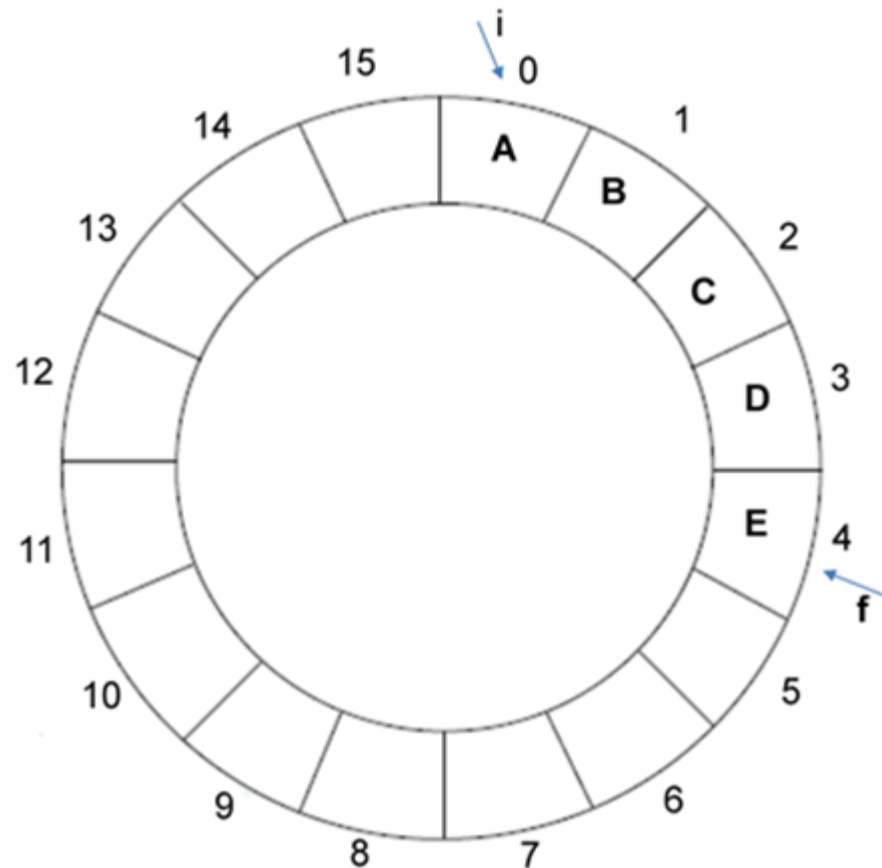
$M = 16$

$n = 5$

$f = 4$

Como  
desenfileirar?

$i++;$   
 $n--;$





# Estrutura de dados

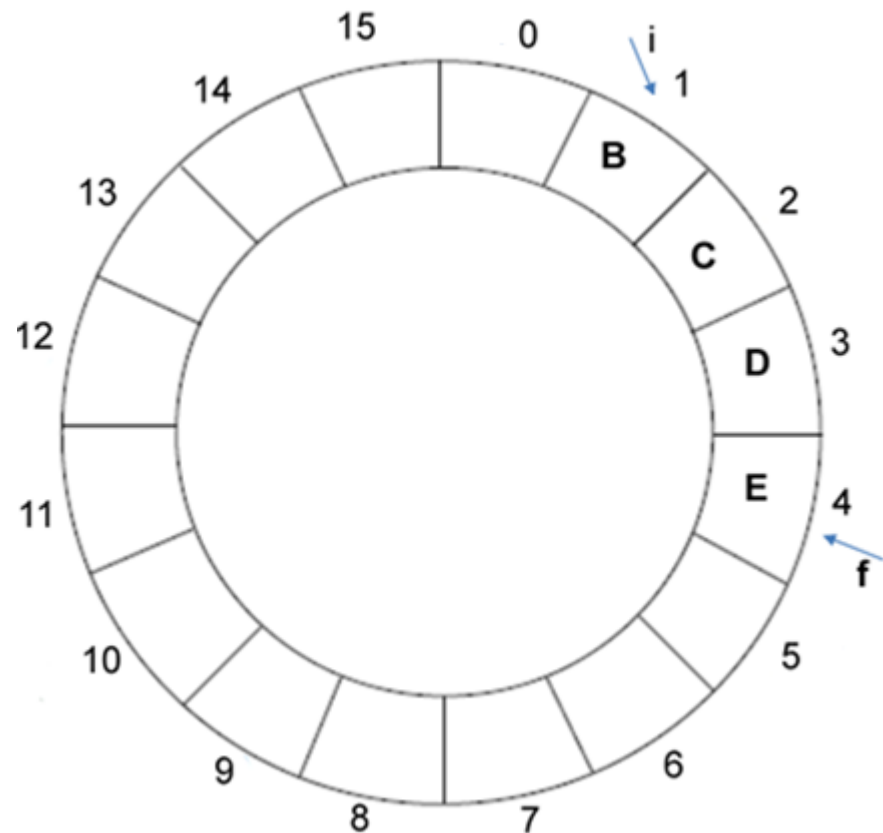
## FILA CIRCULAR – DESENFILAR

$$i = 1$$

$$M = 16$$

$$n = 4$$

$$f = 4$$



# Estrutura de dados



## FILA CIRCULAR – DESENFILAR

$i = 15$

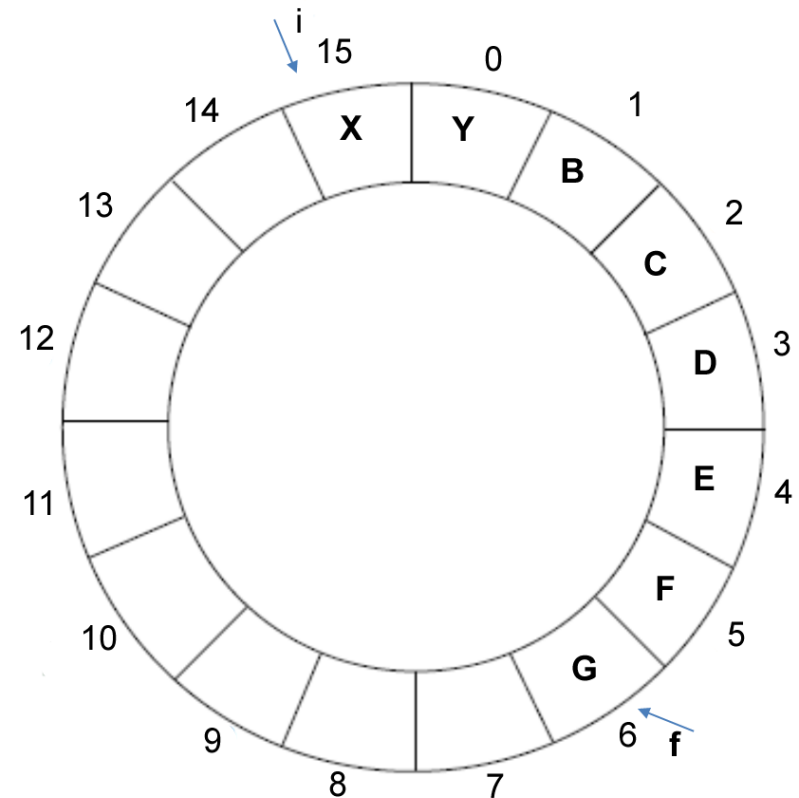
$M = 16$

$n = 8$

**Mais inserções  
e remoções...**

**Como desenfileirar?**

**$i++$ ; ???**



# Estrutura de dados

## FILA CIRCULAR – DESENFILAR

$i = 15$

$M = 16$

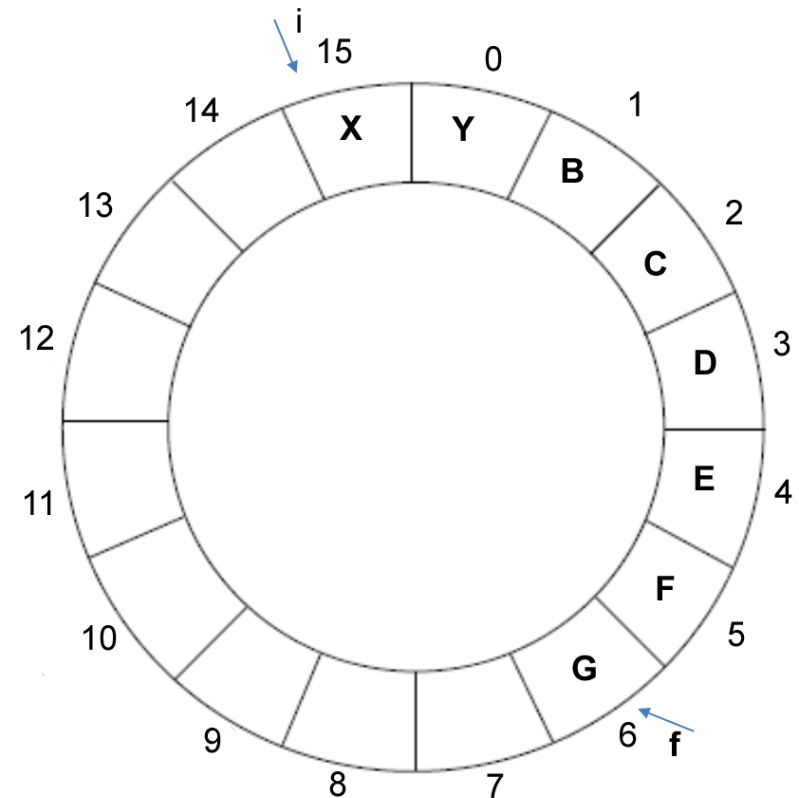
$n = 8$

$i == M-1 ?$



$i = 0;$

$n--;$



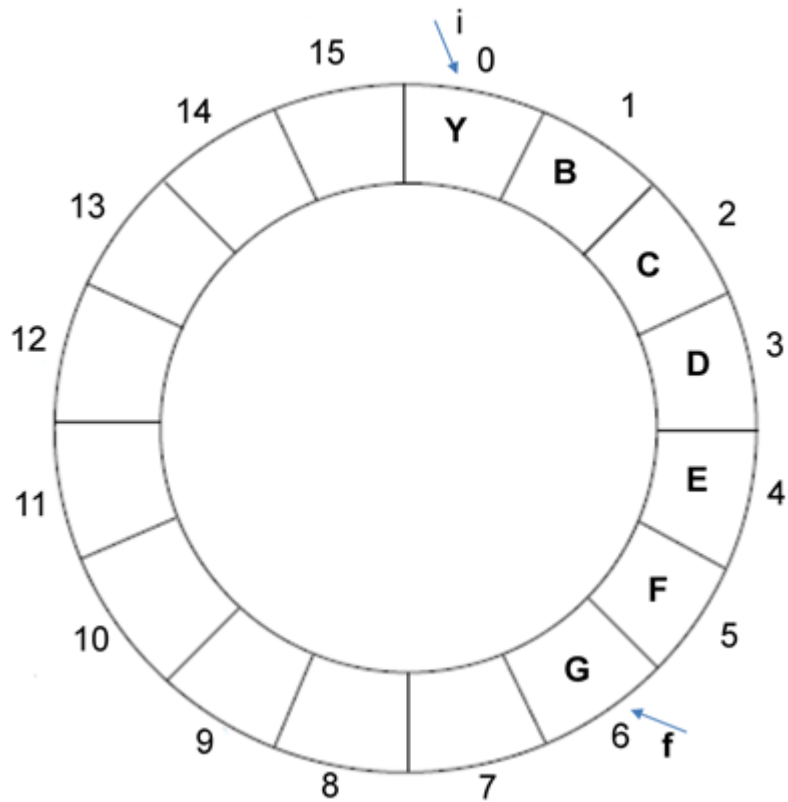
# Estrutura de dados

## FILA CIRCULAR – DESENFILAR

$i = 0$

$M = 16$

$n = 7$



# Estrutura de dados

## FILA CIRCULAR – DESENFILAR

$i = 0$

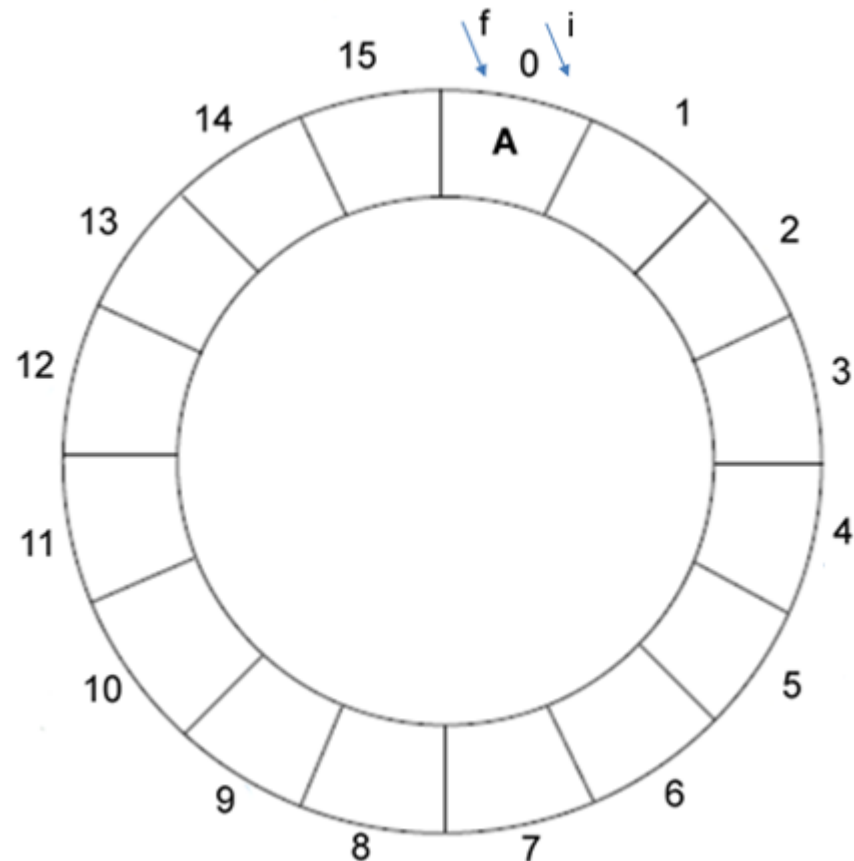
$f = 0$

$M = 16$

$n = 1$

**Como  
desenfileirar?**

**$i = -1;$   
 $f = -1;$   
 $n--;$**



# Estrutura de dados



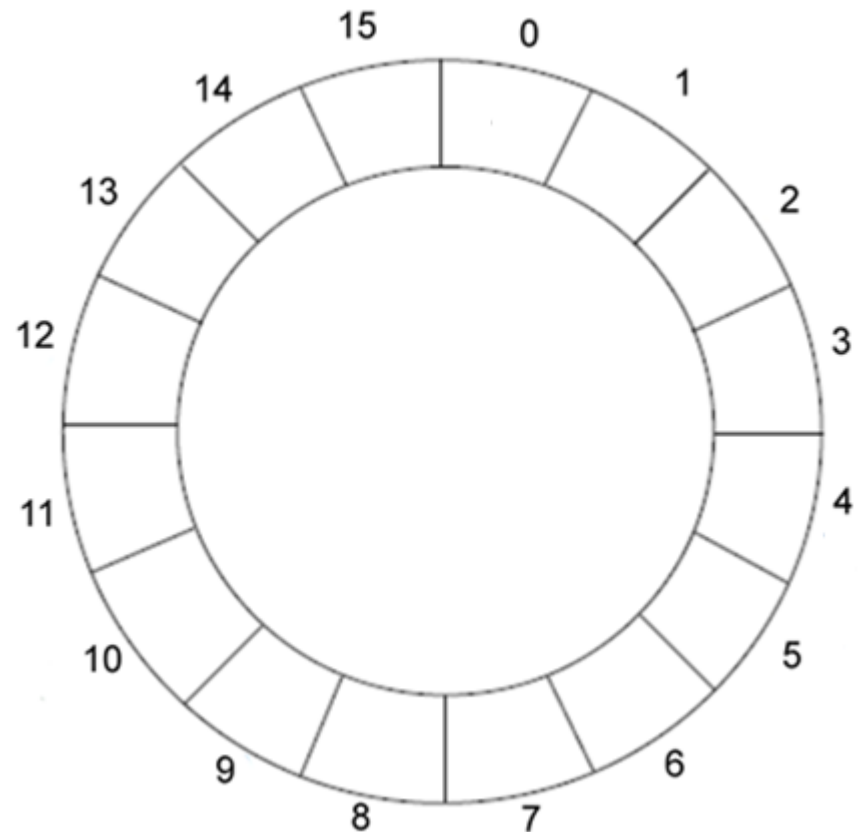
## FILA CIRCULAR – DESENFILAR

$i = -1$

$f = -1$

$M = 16$

$n = 0$



## FILA CIRCULAR – DESENFILAR

```
void desenfileirar(int *n, int *i, int *f){  
    if (fila_vazia(*n))  
        printf("Fila vazia.");  
    else{  
        if (*n==1){           //um só elemento na fila  
            *i = -1;  
            *f = -1;  
        }  
        else{ //Mais de um elemento armazenado  
            if (*i == M-1) //Última posição  
                *i=0;  
            else //Qualquer posição  
                (*i)++;  
        }  
        (*n)--;  
    }  
}
```

## FILA CIRCULAR – IMPLEMENTAÇÃO

```
int main(){
    int fila[M];
    int i, f, n;
    i = -1; f= -1, n=0;
    int num,op,aux;
    do{
        printf("\n1- Enfileirar");
        printf("\n2- Desenfileirar");
        printf("\n3- Sair");
        printf("\nInforme sua opção:");
        scanf("%d",&op);
```



## FILA CIRCULAR – IMPLEMENTAÇÃO

```
if(op==1){  
    printf("Informe um valor:");  
    scanf("%d",&num);  
    enfileira(num, fila, &i, &f, &n);  
}  
else{  
    if(op==2)  
        desenfileira(&n, &i, &f);
```

## FILA CIRCULAR – DESENFILAR

```
    else{
        if (op==3){
            printf("\nFinalizando...\n");
            break;
        }
        else
            printf("\nOpção inválida!\n");
    }
}while(op!=3);
}
```

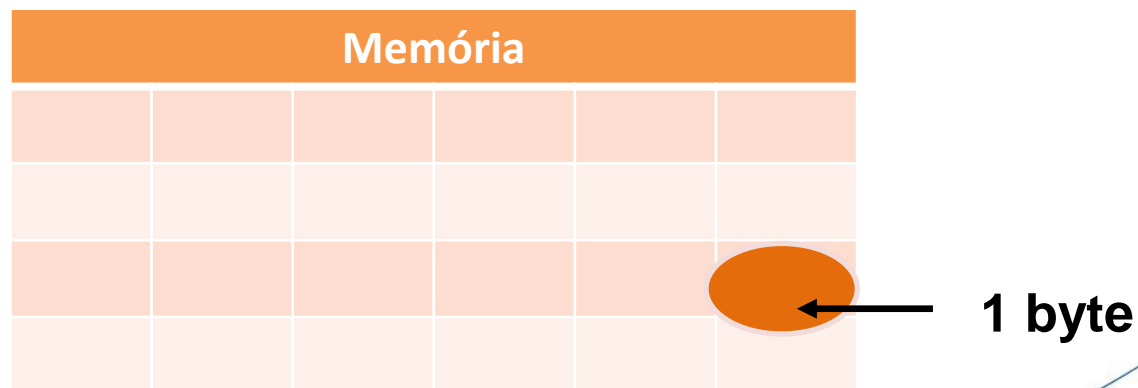
# PONTEIROS

## REVISÃO

1. O que são ponteiros?
2. Em que situações os ponteiros costumam ser usados?

## ARMAZENANDO INFORMAÇÃO

- Armazenamento de informação -> ocupa espaço;
- Declaração de variável -> alocação de espaço de memória;
- Variável associada a um espaço de memória do computador. Tal espaço de memória é representado por um valor, seu endereço.



# Estrutura de dados

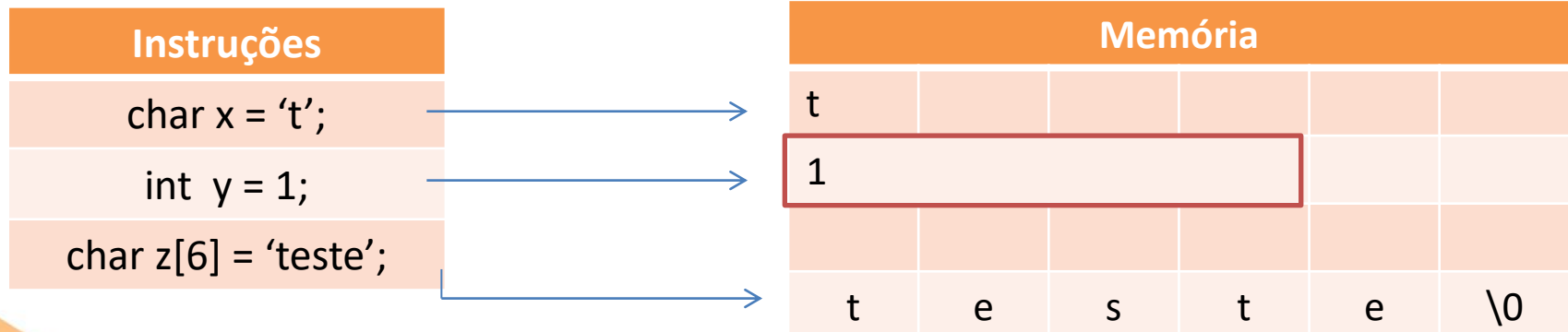


## ARMAZENANDO INFORMAÇÃO

Cada tipo de dado declarado em um programa ocupa um tamanho diferente na memória do computador.

Tipo de Dado	Tamanho
char	1 byte
int	4 bytes
float	4 bytes
Double	8 bytes

**Uso do comando sizeof() para obter o tamanho que cada tipo de dado ocupa.**



# PONTEIRO

## O QUE É UM PONTEIRO?

- Variável que armazena um endereço de memória.
- É um tipo de variável especial que armazena o endereço de uma segunda variável alocada na memória;
- Tem por função apontar para um endereço de memória determinado, isto é, o endereço que o ponteiro armazena.
- Os ponteiros são muito úteis e bastante utilizados nas situações onde é necessário conhecer o endereço onde está armazenada fisicamente uma variável e não propriamente o seu conteúdo.



## DECLARANDO UM PONTEIRO?

- Sintaxe da declaração de ponteiros:  
tipo \*nome\_ponteiro;
- Declaração:  
int x, \* pt\_x;
- Variável x do tipo int e ponteiro \*pt\_x que armazenará o endereço de uma variável do tipo int.

## INICIALIZAÇÃO DE PONTEIROS

### Inicialização de um ponteiro

- Ponteiros devem ser inicializados antes de serem usados, o que pode ser feito na declaração ou através de uma atribuição.
- Um ponteiro pode ser inicializado com um endereço ou com o valor NULL. O valor NULL é uma constante definida na biblioteca `<stdio.h>` e significa que o ponteiro não aponta para lugar nenhum.

## OPERADORES DE MANIPULAÇÃO

- Manipulação de ponteiros ocorre de duas maneiras:
  - Por meio do endereço de uma variável;
  - Por meio do conteúdo armazenado no endereço apontado pelo ponteiro.
- Operadores:
  - Operador de endereço: &
  - Operador de conteúdo: \*

## OPERADORES DE MANIPULAÇÃO

### Operador de endereço e conteúdo (& e \*)

#### Programa

```
#include <stdio.h>
int main(){
    char x = 't', *p;
    p = &x;
    printf("Conteúdo do ponteiro: %p", p);
    printf("Conteúdo de x é: %c ", x);
    printf("Conteúdo apontado: %c", *p);
}
```

Busca o endereço  
armazenado

Busca o conteúdo de x

Busca o conteúdo apontado pelo ponteiro

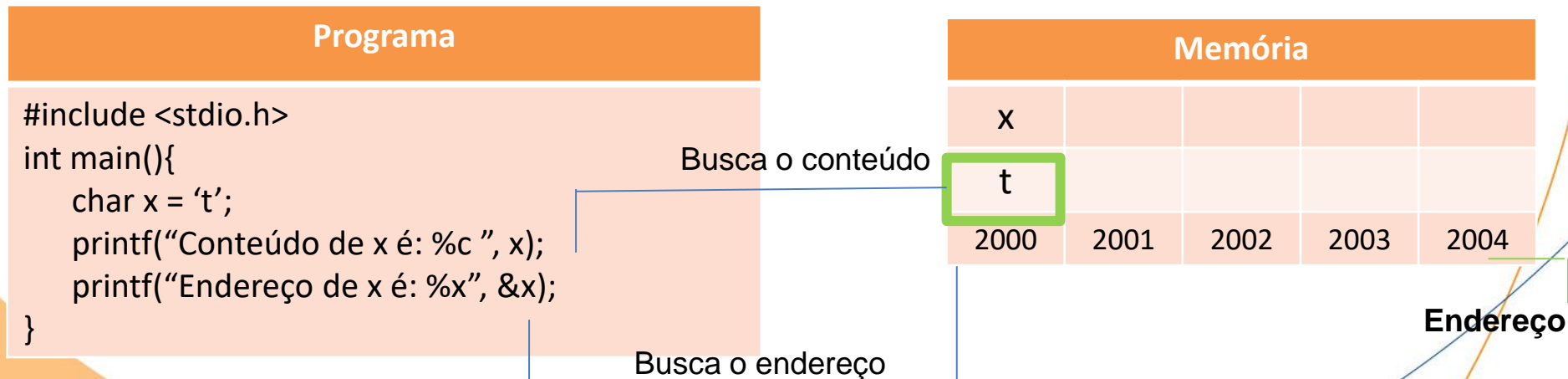
#### Memória

Memória			
x			p
t			2000
2000	2001	2002	2003

## OPERADORES DE MANIPULAÇÃO

### Operador de endereço (&)

Indica o endereço de uma variável, que pode ser impresso ou lido com printf e scanf a partir dos operadores de conversão %x ou %X.



## OPERADORES DE MANIPULAÇÃO

### **Operador de conteúdo (\*)**

Utilizado na declaração de um ponteiro, mas também toda vez que se deseja saber qual o valor contido no endereço armazenado pelo ponteiro.

Endereço de um ponteiro: endereço físico alocado para essa variável no momento da sua declaração;

Endereço armazenado pelo ponteiro: conteúdo do ponteiro.

## OPERADORES DE MANIPULAÇÃO

### Operador de conteúdo (\*)

Através do operador de conteúdo, também é possível atribuímos um valor à variável apontada pelo ponteiro.

Exemplo:

```
int a = 5, *p;  
p = &a;  
printf("a = %d", a);  
*p = 10;  
printf("a = %d", a);
```

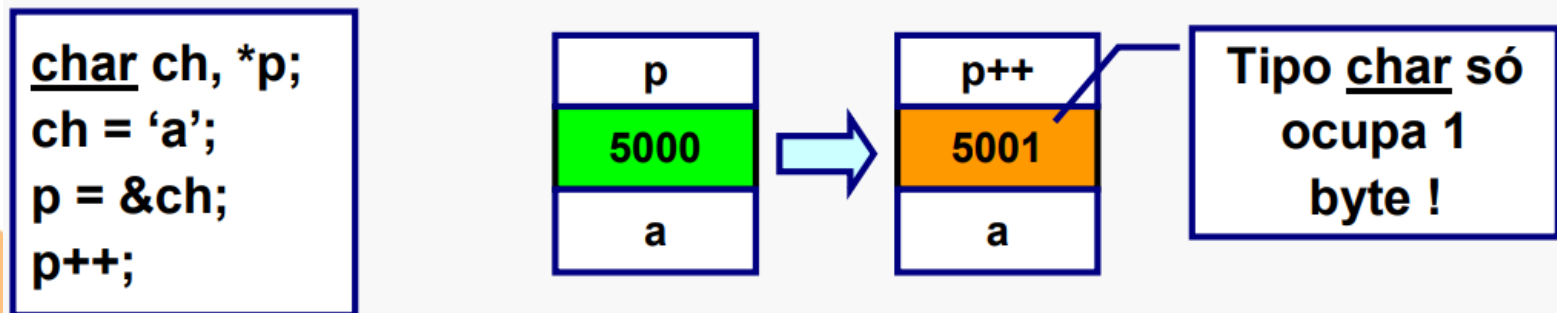
# Estrutura de dados



## ARITMÉTICA DE PONTEIROS

É possível realizar operações de soma e subtração nos ponteiros.

Ao somar o valor 1 a um ponteiro, o endereço contido no ponteiro será modificado para o próximo endereço de memória correspondente ao tipo de dado especificado.





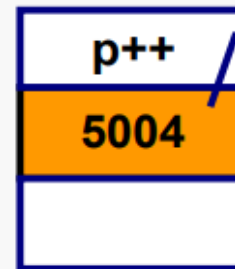
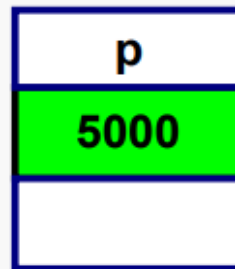
# Estrutura de dados



## ARITMÉTICA DE PONTEIROS

```
int x, *p;  
x = 10;  
p = &x;  
p++;
```

Memória



Tipo int  
ocupa 4  
bytes !

Espaço de 4 bytes para  
armazenar a variável x tipo int.

A operação p++ percorre  
sizeof(tipo p) bytes !

p				p++			
x	x	x	x	?	?	?	?
5000	5001	5002	5003	5004	5005	5006	5007

## COMPARAÇÃO DE PONTEIROS

É possível comparar ponteiros em uma expressão relacional. No entanto, só é possível comparar ponteiros de mesmo tipo.

`if (px == py) // se px aponta para o mesmo bloco que py ...`

`if (px > py) // se px aponta para um bloco posterior a py ...`

`if (px != py) // se px aponta para um bloco diferente de py ...`

`if (px == NULL) // se px é nulo...`

## PONTEIROS E VETORES

Na inicialização de um ponteiro com variáveis do tipo vetor, matriz ou string não é preciso usar o operador `&`, pois eles já são considerados ponteiros constantes.

Na atribuição será repassado o endereço alocado referente à primeira posição da variável.

Exemplo:

```
char nome[34];  
char *ponteiro;  
ponteiro = nome;
```

# Estrutura de dados



## PONTEIROS E VETORES

```
#include <stdio.h>
int main (void){
    float v[] = {1.0 , 2.0, 3.0, 4.0, 5.0, 6.0, 7.0};
    int i;
    float *p;
    p = v;
    for (i = 0; i < 7; i++)
        printf("%.1f ", p);
    printf("\n");
}
```

# Estrutura de dados



## PONTEIROS E FUNÇÕES

A passagem de parâmetros entre as sub-rotinas se dá de duas formas:

- Passagem por valor;
- Passagem por referência.

## PONTEIROS E FUNÇÕES

Passagem por referência:



permite a alteração do valor de uma variável



necessária a passagem do endereço do  
argumento para a função.



Uso de ponteiros.