

# Pilhas e Filas

Prof.<sup>a</sup> Hélida Salles

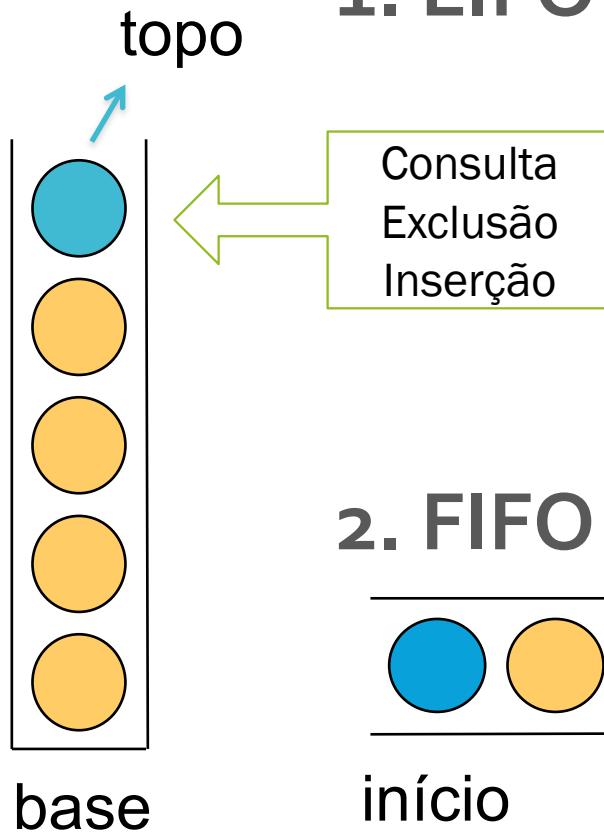
# Listas Lineares Especiais

❑ Algumas listas possuem disciplina restrita de acesso aos nodos

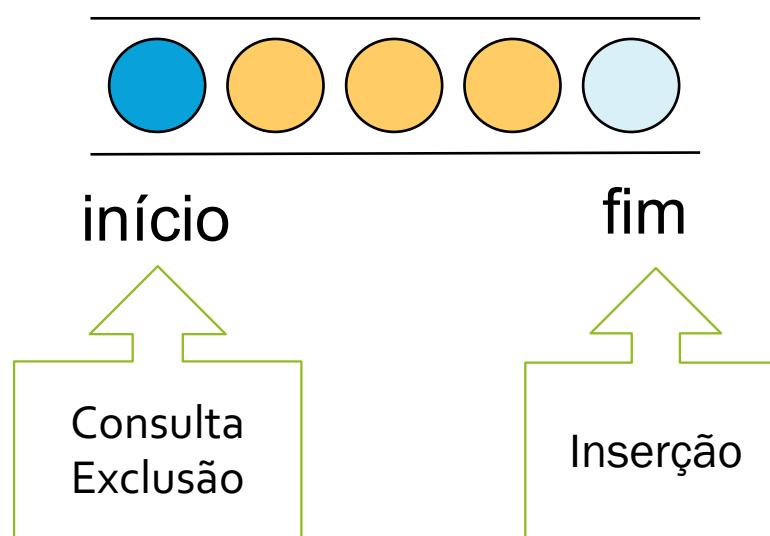
- Principais restrições:
  - Last In First Out (LIFO)
    - O último elemento inserido é o primeiro a ser retirado
  - First In First Out (FIFO)
    - O primeiro elemento inserido é o primeiro a ser retirado

# Listas Lineares Especiais

## 1. LIFO - Pilhas



## 2. FIFO - Filas

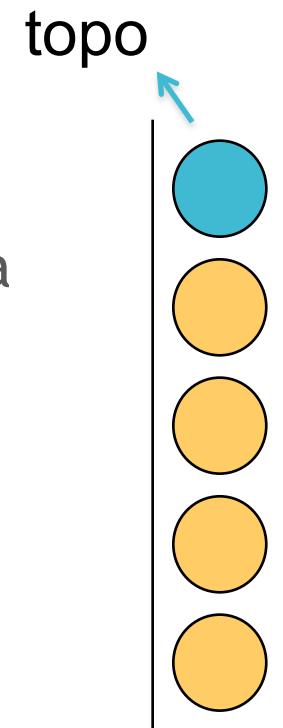


# Listas Lineares Especiais

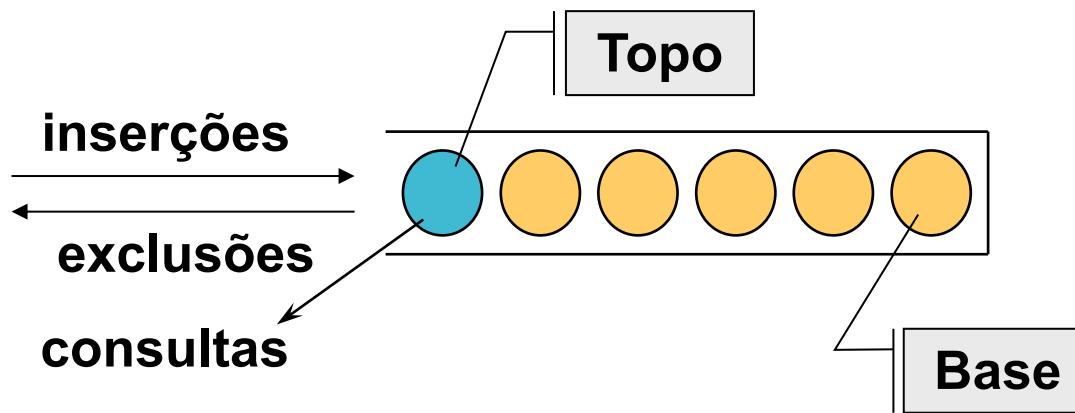
- ❑ Pilhas e filas modelam diversas aplicações práticas
  - ❑ Fila de banco
  - ❑ Fila de impressão
  - ❑ CRTL+ z
  - ❑ Histórico em um navegador, etc.

# PILHAS

- Principal restrição:
  - Operações somente ocorrem no topo da pilha
    - Inserção
    - Remoção
    - Consulta
- Comportamento
  - Last In, First Out (LIFO)
    - O último elemento inserido é o primeiro a ser retirado



# Operações sobre Pilhas



## Operações válidas:

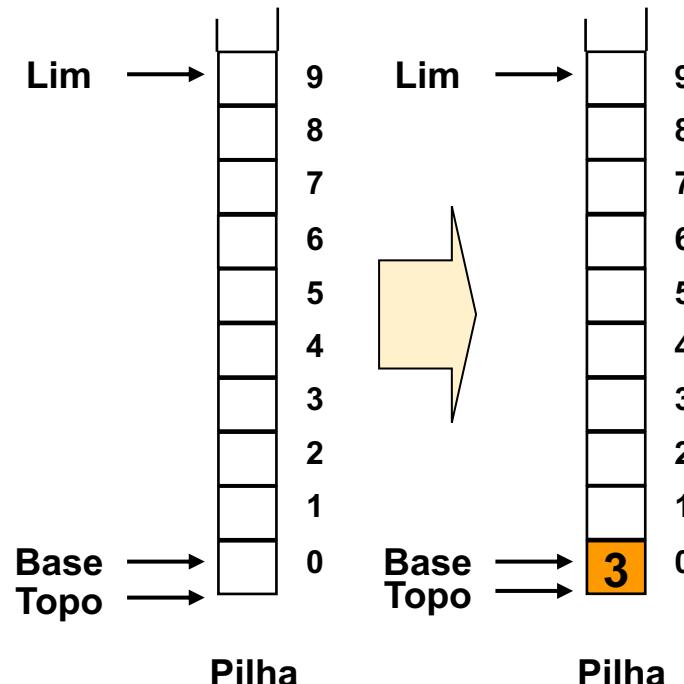
- criar uma pilha vazia (**new**)
- inserir um nó no topo da pilha (**push**)
- excluir o nó do topo de pilha (**pop**)
- consultar/modificar nó do topo da pilha (**top**)
- destruir a pilha (**destroy**)

# Implementação de Pilhas

## Contiguidade Física

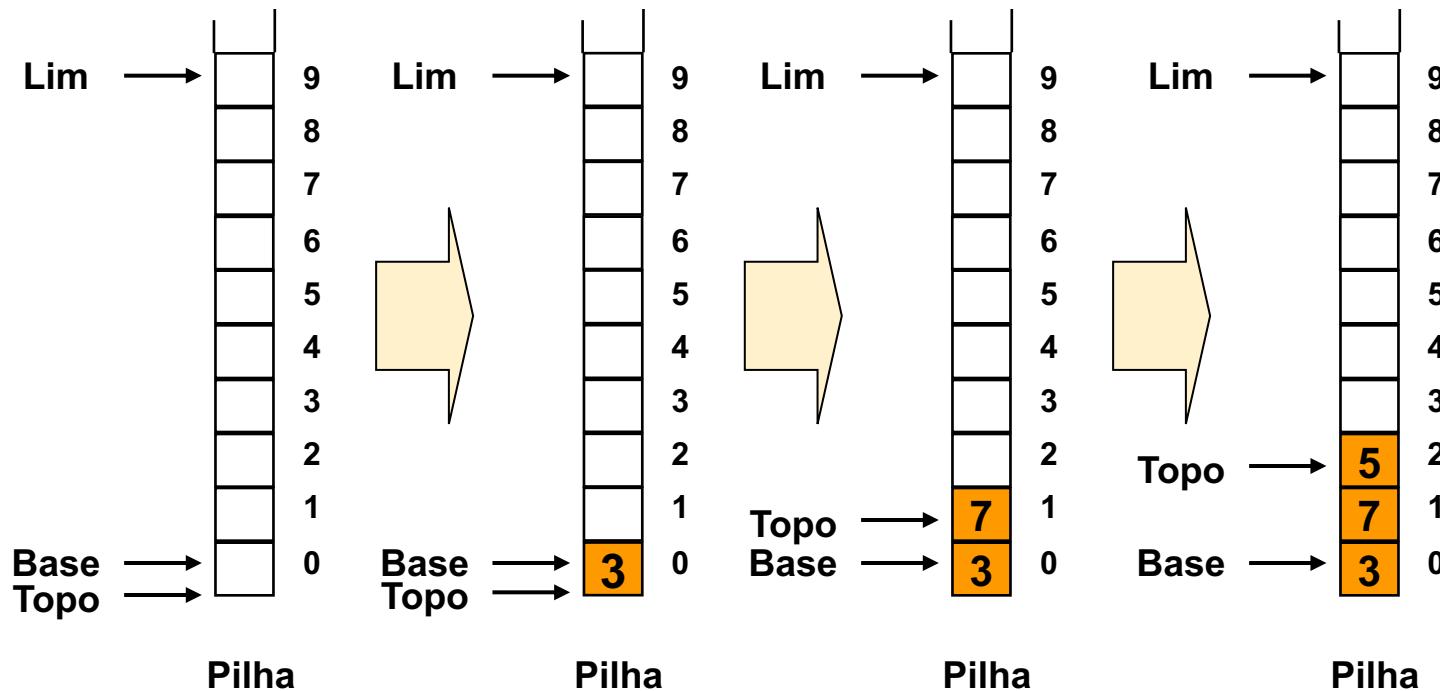
# Exemplo de manipulação de uma pilha

1. Inicializar pilha de valores inteiros, a partir do índice 0, máximo 10 nós
2. Inserir nó com valor 3



# Exemplo de manipulação de uma pilha

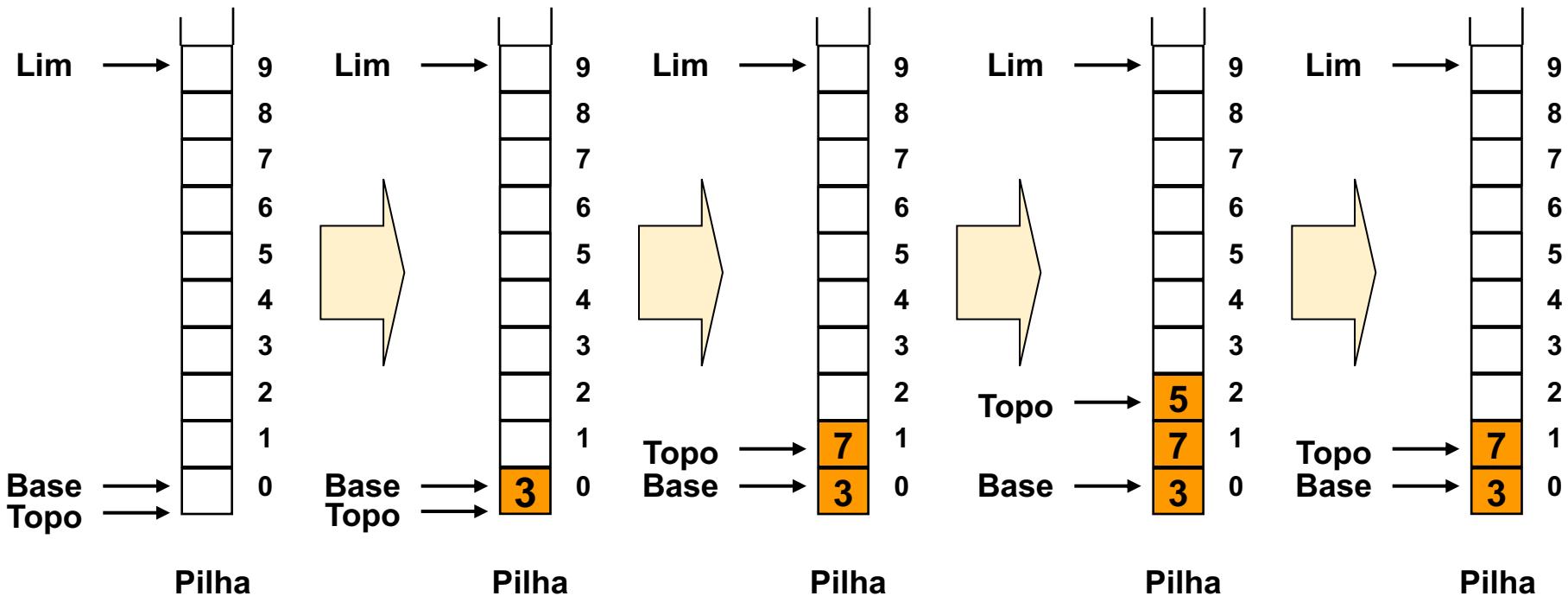
3. Inserir nó com valor 7
4. Inserir nó com valor 5



# Exemplo de manipulação de uma pilha

5. Remover nó
6. Consultar pilha

Retorna “7”

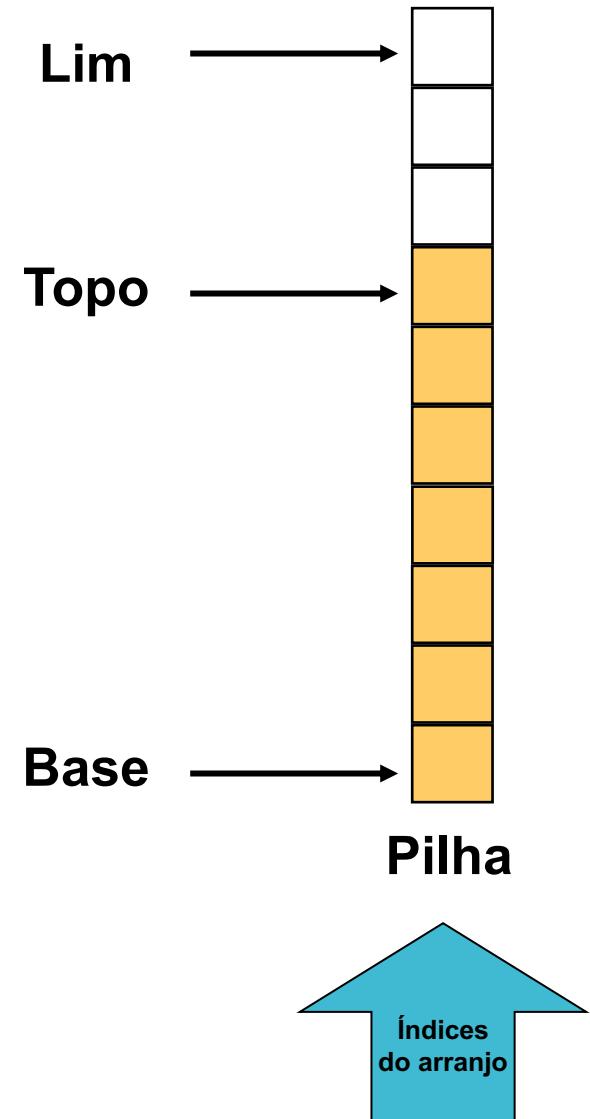


# Pilha - contiguidade física

```
typedef struct sProduto {  
    int cod;  
    char nome[40];  
    float preco;  
} Produto;  
  
int base, limite, topo;
```

## Operações válidas:

1. Criar uma pilha vazia
2. Inserir um nó no topo da pilha
3. Excluir o nó do topo da pilha
4. Consultar / modificar nó do topo da pilha
5. Destruir a pilha



# Pilha

contiguidade física

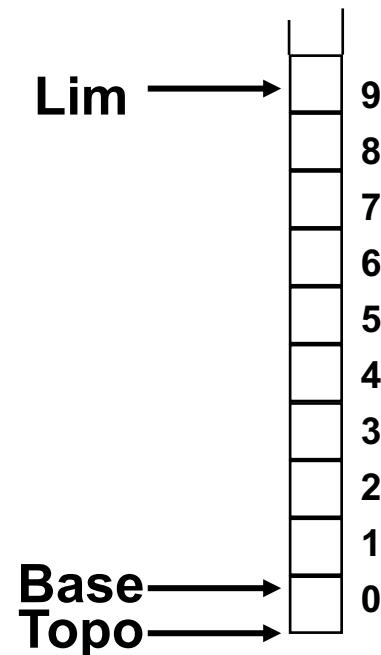
## Operações :

### 1. Criar uma pilha vazia

1. Definir valor do índice de Base da pilha
2. Definir valor máximo de nós que a pilha pode ter (Lim-Base+1)
3. Indicar que a pilha está vazia através do índice de Topo

Exemplo:

**base = ????**  
**limite = ????**  
**topo = ????**



**Pilha**

# Criar pilha vazia

contiguidade física

```
void inicializa (Produto P[], int *base, int *limite,  
int *topo)  
{  
    int i;  
    for (i=0; i<MAX; i++)  
    {  
        strcpy(P[i].nome,"");  
        P[i].cod = 0;  
        P[i].preco = 0;  
    }  
    *base = ???  
    *limite = ???  
    *topo = ???  
}
```

# Criar pilha vazia

contiguidade física

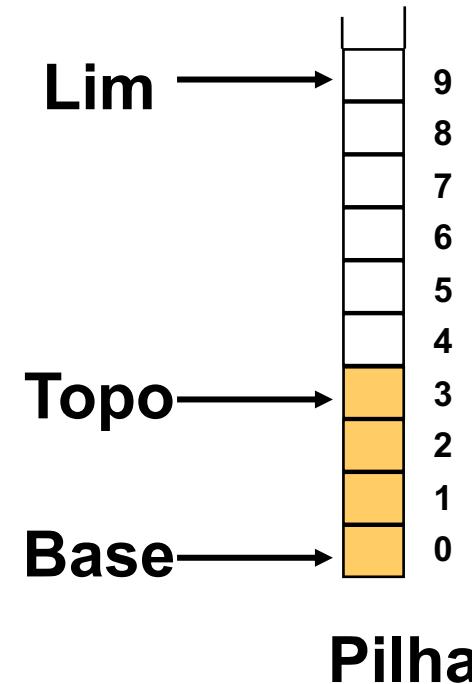
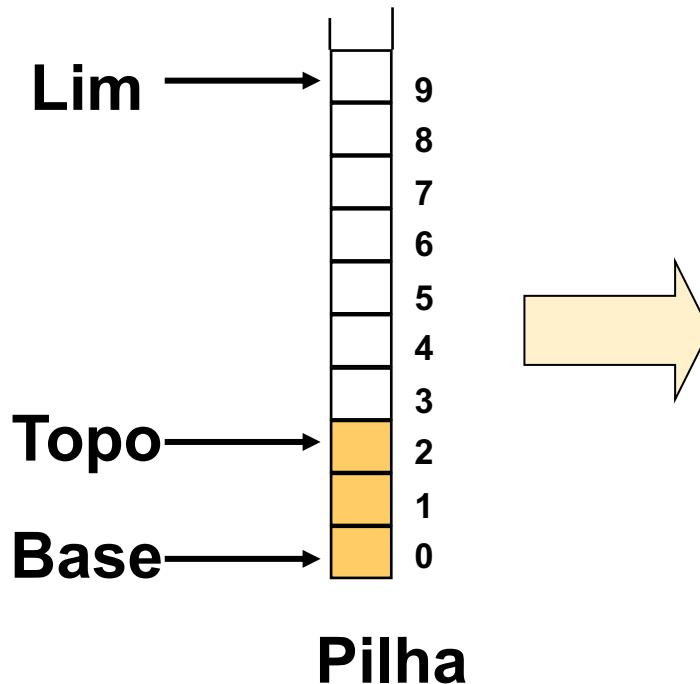
```
void inicializa (Produto P[], int *base, int *limite,  
int *topo)  
{  
    int i;  
    for (i=0; i<MAX; i++)  
    {  
        strcpy(P[i].nome,"");  
        P[i].cod = 0;  
        P[i].preco = 0;  
    }  
    *base = 0;  
    *limite = MAX;    // pode ser um tamanho menor  
    *topo = *base-1;  
}
```

## Operações :

2. Inserir um nó no topo da pilha

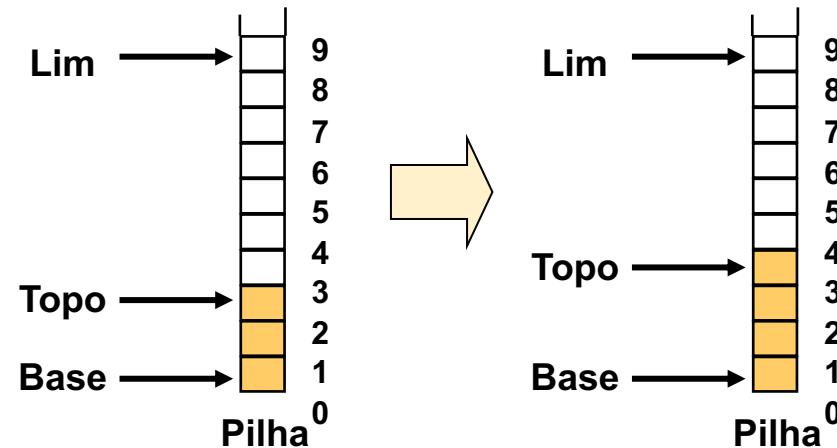
# Pilha

contiguidade física



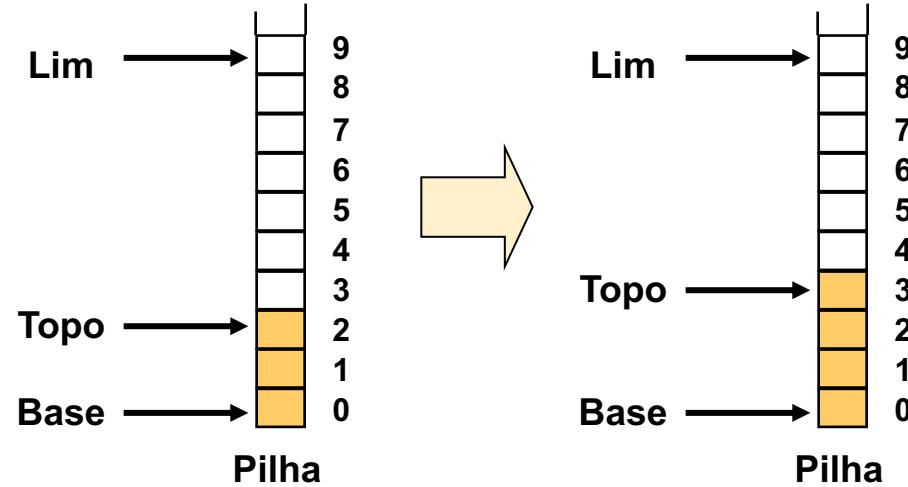
# Inserir nó em uma pilha

```
void Empilha (Produto P[], int limite, int *topo, Produto elem)
{
    if ( *topo ??? limite )
    {
        *topo = ???
        P[???] = elem;
    }
}
```



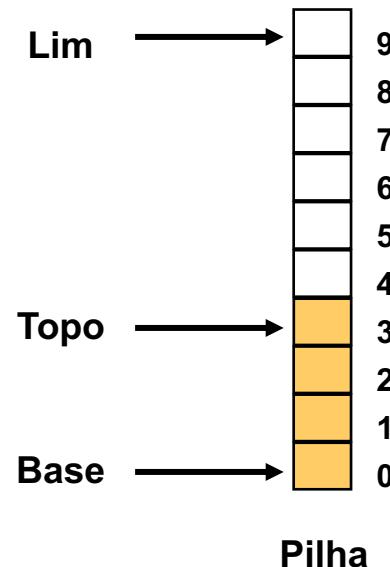
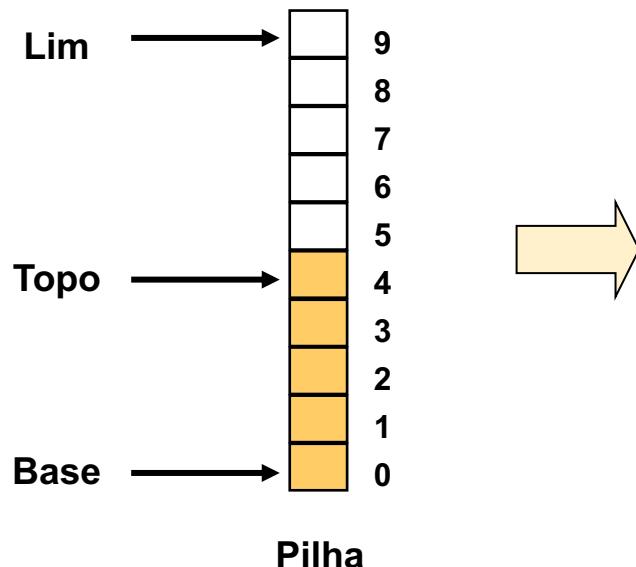
# Inserir nó em uma pilha

```
void Empilha (Produto P[], int limite, int *topo, Produto elem)
{
    if (*topo < limite)
    {
        *topo = *topo +1;
        P[*topo] = elem;
    }
}
```



# Pilha

contiguidade física



# Excluir nó da pilha

contiguidade física

Produto Desempilha( Produto P[], int base, int \*topo)

{

    Produto rem;

    rem.cod = 0;

    strcpy(rem.nome, "");

    rem.preco = 0;

    if (\*topo ??? base)

    {

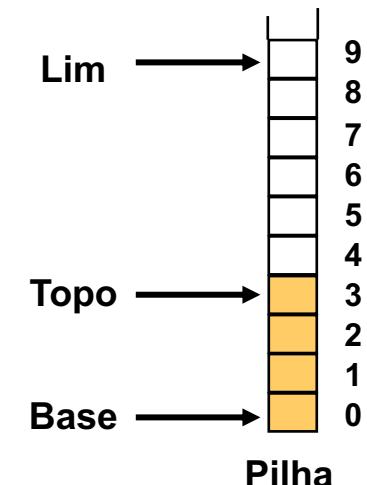
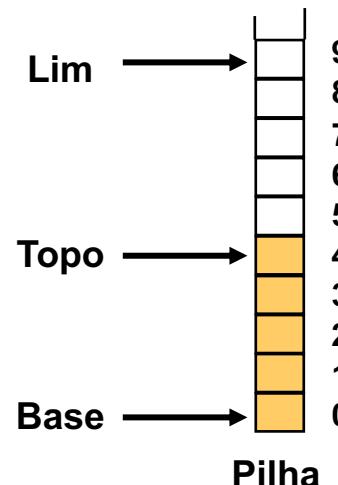
        rem = ???

        \*topo = ???

    }

    return ???;

}



# Excluir nó da pilha

contiguidade física

Produto Desempilha( Produto P[], int base, int \*topo)

{

    Produto rem;

    rem.cod = 0;

    rem.nome = "";

    rem.preco = 0;

    if (\*topo >= base)

    {

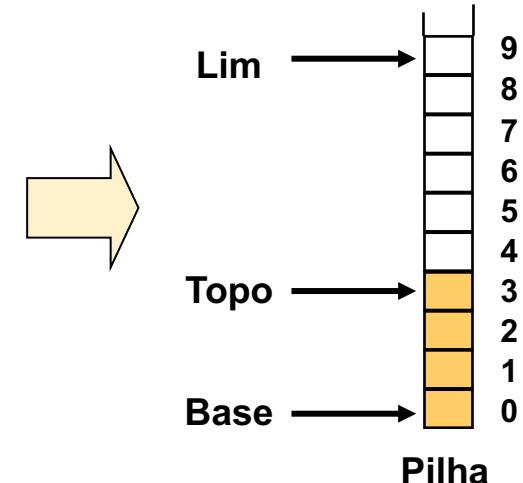
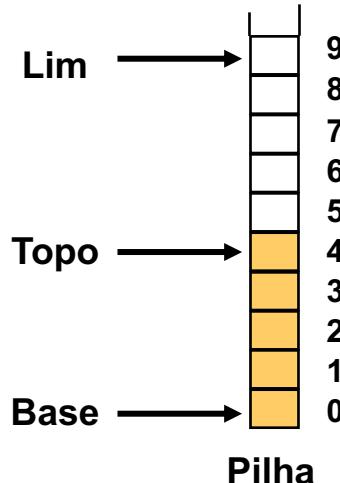
        rem = P[\*topo];

        \*topo = \*topo -1;

    }

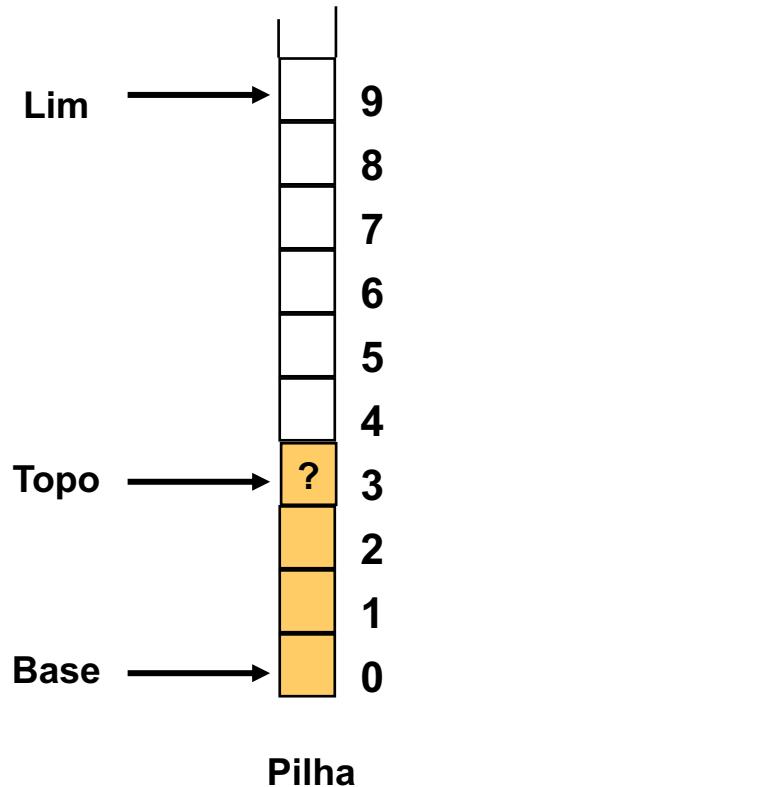
    return rem;

}



## Operações :

### 4. consultar nó do topo da pilha

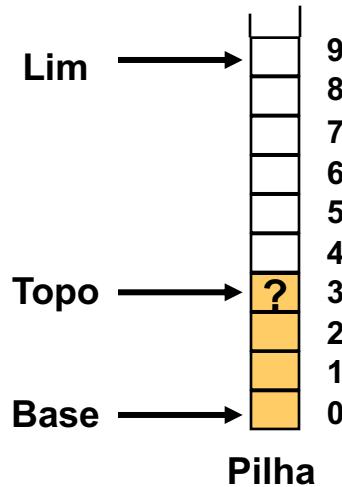


# Pilha

contiguidade física

# Consultar pilha

contiguidade física



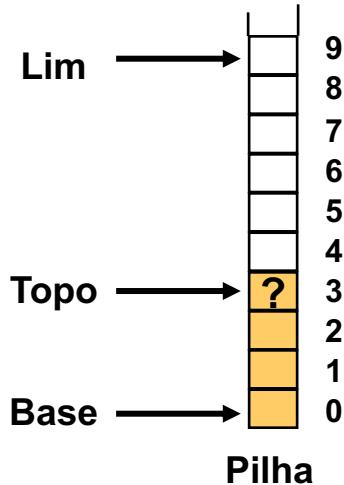
```
Produto consulta ( Produto P[], int base, int topo)
{
    Produto ret;
    ret.cod = 0;
    ret.nome = " ";
    ret.preco = 0;

    if ( topo ??? base)
        ret = ???;

    return ret;
}
```

# Consultar pilha

contiguidade física



```
Produto consulta ( Produto P[], int base, int topo)
{
```

```
    Produto ret;
```

```
    ret.cod = 0;
```

```
    ret.nome = " ";
```

```
    ret.preco = 0;
```

```
    if ( topo >= base)
```

```
        ret = P[topo];
```

```
    return ret;
```

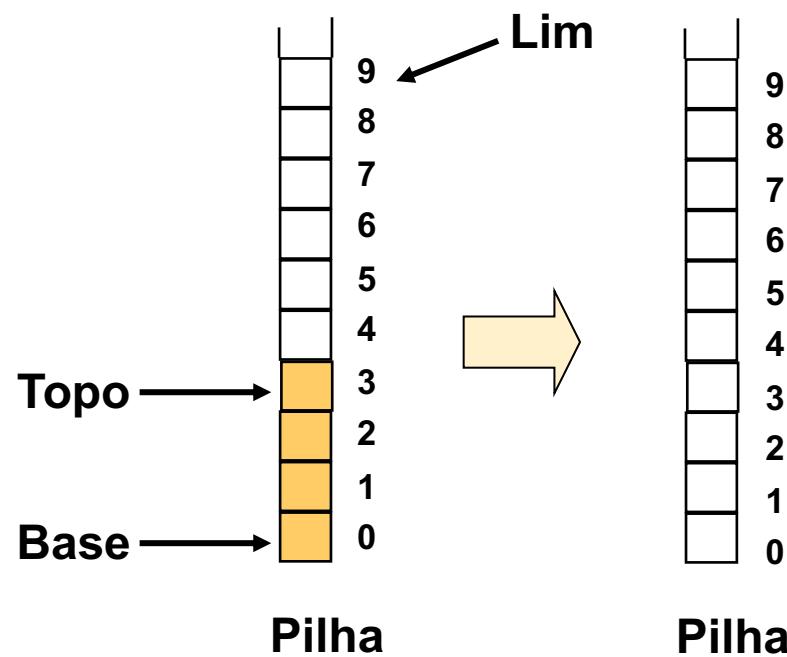
```
}
```

# Pilha

contiguidade física

Operações :

5. Destruir a pilha



# Destruir pilha

contiguidade física

```
void destroi ( Produto P[], int *base, int *limite,  
int *topo)  
{  
    *base = ???  
    *limite = ???  
    *topo = ???  
}
```

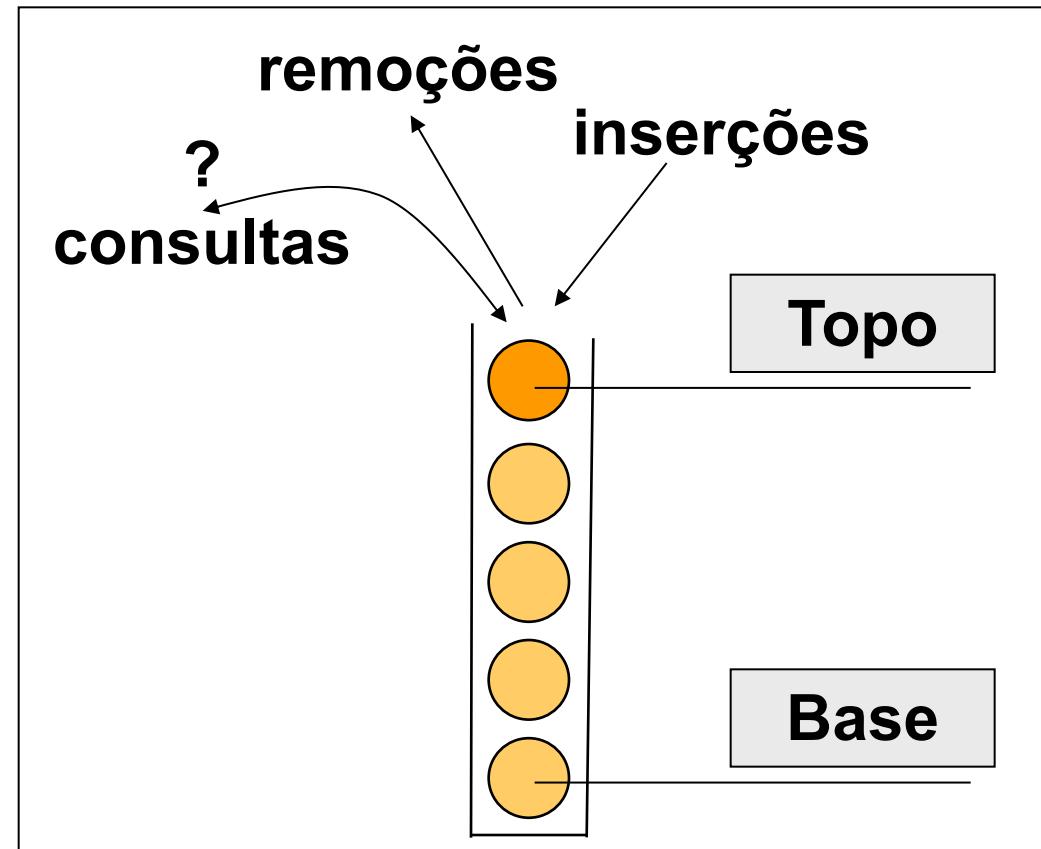
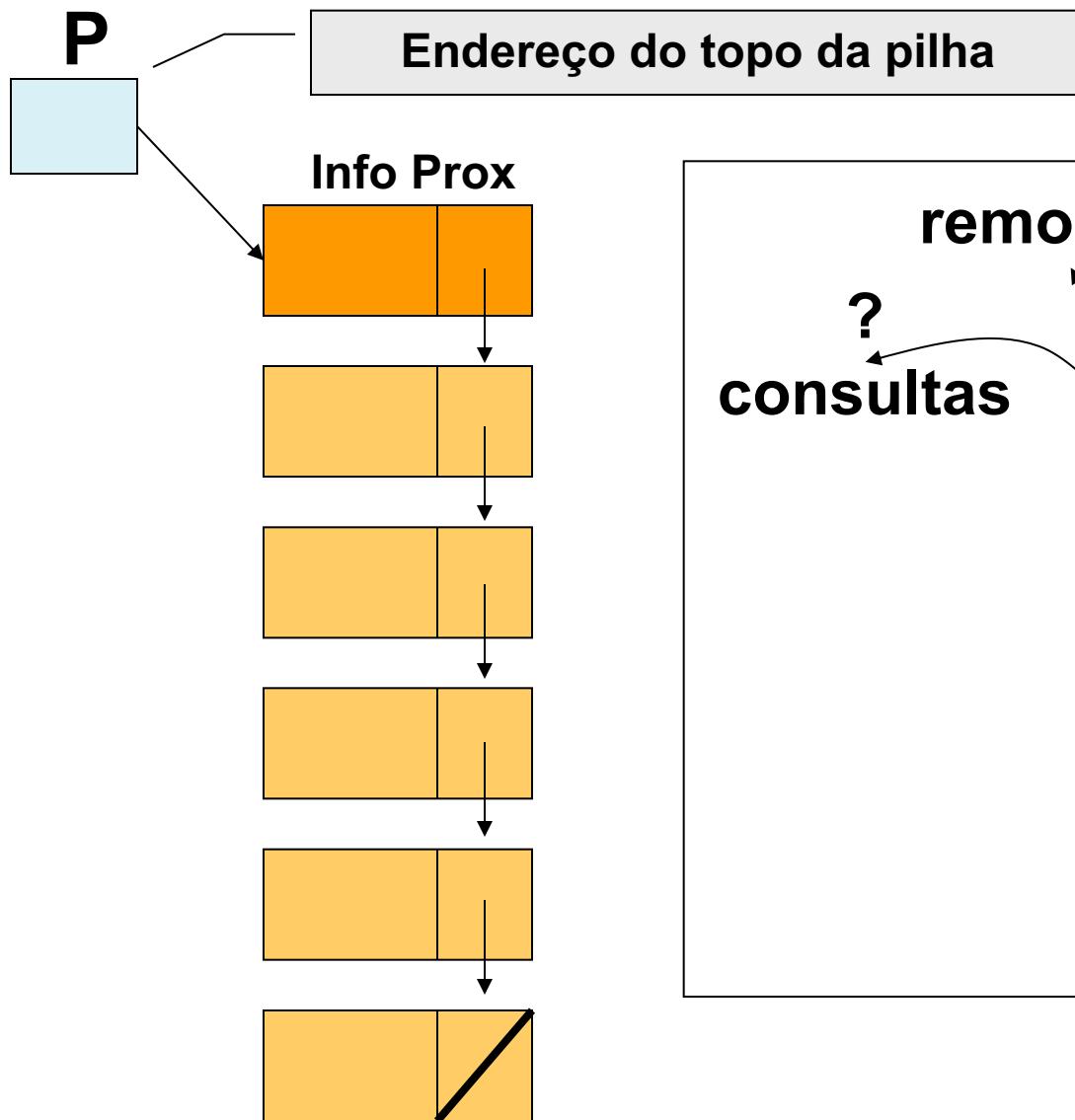
## **Destruir pilha**

contiguidade física

```
void destroi ( Produto P[], int *base, int *limite,  
int *topo)  
{  
    *base = -1;  
    *limite = -1;  
    *topo = -1;  
}
```

# Implementação de Pilhas Encadeamento

# Pilha encadeada



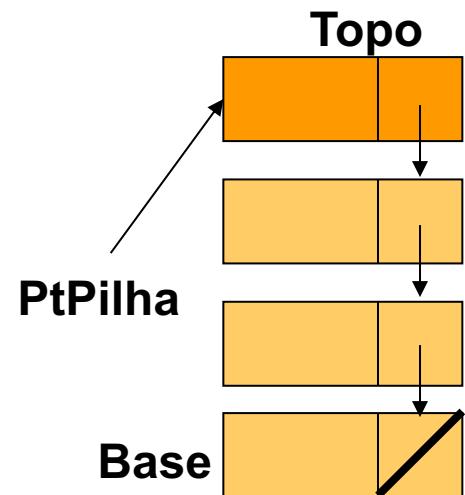
# Pilha encadeada

```
/* tipo pilha abstrai a estrutura de um nodoPilha */
typedef struct nodoPilha pilha;

/* tipo info abstrai o elemento de dado a ser
empilhado */
typedef int info;

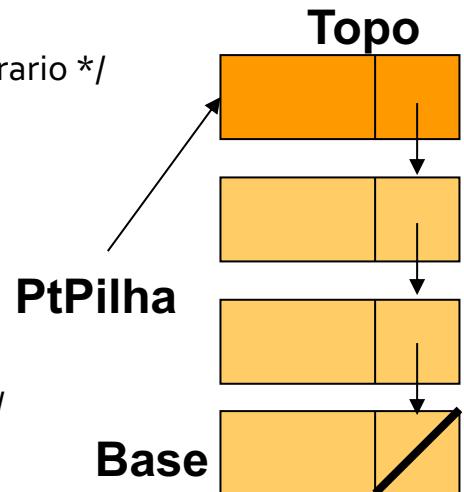
/* implementacao do nodoPilha */
struct nodoPilha{
    info dado;
    pilha *elo;
};

pilha* inicializaPilha ();
pilha* pushPilha (pilha *topo, info dado);
pilha* popPilha (pilha *topo, info *dado);
int popPilha (pilha **topo, info *dado);
info topPilha (pilha *topo);
pilha* destroiPilha (pilha *topo);
```



# Pilha encadeada

```
/* retorna uma pilha vazia */  
pilha* inicializaPilha ();  
  
/* empilha um dado e retorna a pilha resultante*/  
pilha* pushPilha (pilha *topo, info dado);  
  
/* retorna novo topo da pilha */  
pilha* popPilha (pilha *topo, info *dado);  
  
/* retorna 1 se conseguiu desempilhar, o caso contrario */  
int popPilha (pilha **topo, info *dado);  
  
/* desempilha e retorna o topo da pilha */  
info topPilha (pilha *topo);  
  
/* desaloca a memoria e retorna uma pilha vazia */  
pilha* destroiPilha (pilha *topo);
```



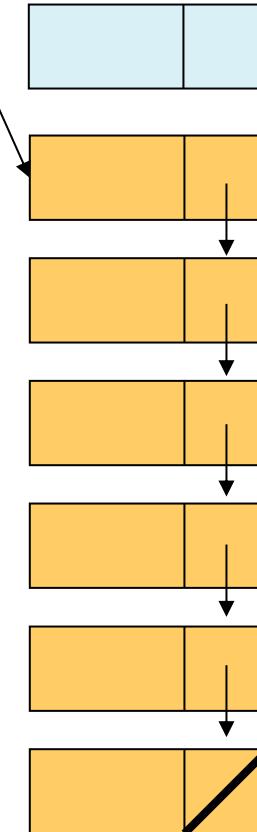
# Criar pilha vazia

```
pilha* inicializaPilha ()  
{  
    return NULL;  
}
```

# Inserção em pilha encadeada

```
pilha * pushPilha (pilha *topo, info dado);  
{  
    declarar e alocar um nodo auxiliar  
    se a alocação foi bem sucedida  
  
    senão  
        indicar erro de alocação  
        sair  
}
```

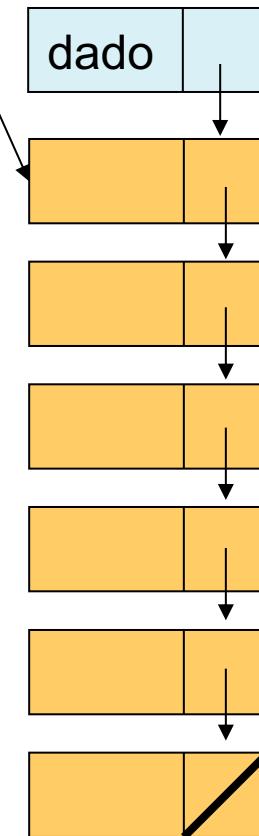
**topo**



# Inserção em pilha encadeada

```
pilha * pushPilha (pilha *topo, info dado);  
{  
    declarar e alocar um nodo auxiliar  
    se a alocação foi bem sucedida  
        armazenar dado  
        encadear o topo  
  
    senão  
        indicar erro de alocação  
        sair  
}
```

**topo**



# Inserção em pilha encadeada

```
pilha * pushPilha (pilha *topo, info dado);  
{
```

declarar e alocar um nodo auxiliar  
se a alocação foi bem sucedida

armazenar dado

encadear o topo

retornar endereço do nodo

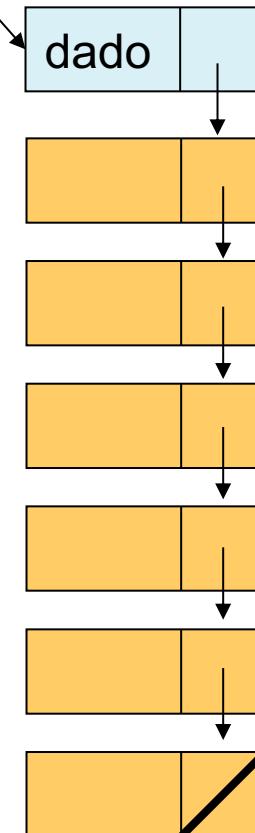
senão

indicar erro de alocação

sair

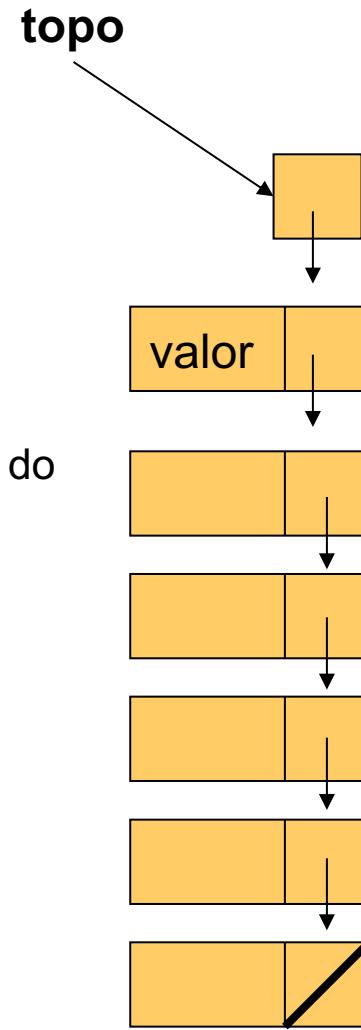
```
}
```

**topo**



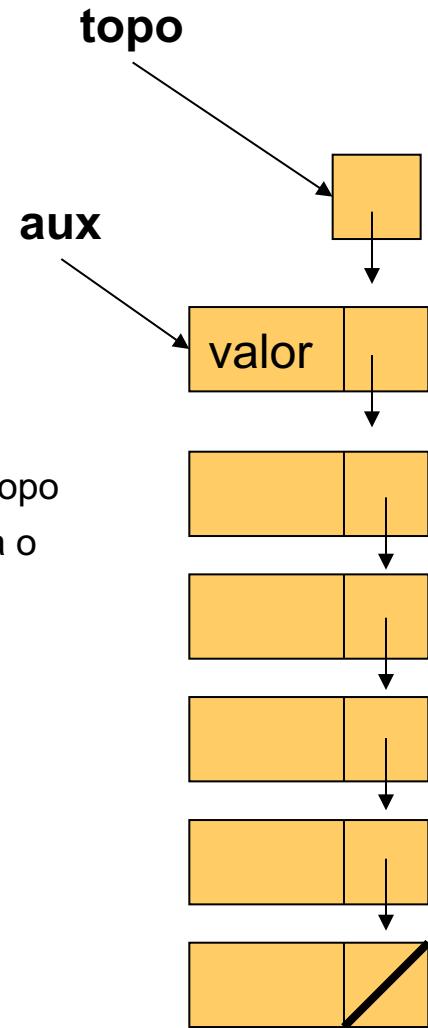
# Remoção em pilha encadeada

```
int popPilha (pilha **topo, info *dados)
{
    se pilha é vazia
        retorna 0
    senão
        armazena em dado o valor do
        topo
    retorna 1
}
```



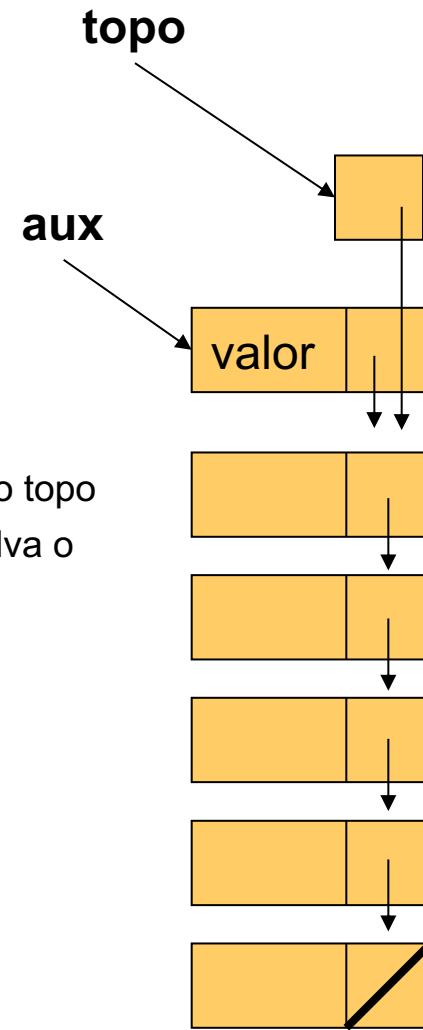
# Remoção em pilha encadeada

```
int popPilha (pilha **topo, info *dados)
{
    se pilha é vazia
        retorna 0
    senão
        armazena em dado o valor do topo
        declara ponteiro auxiliar e salva o
        topo
        retorna 1
}
```



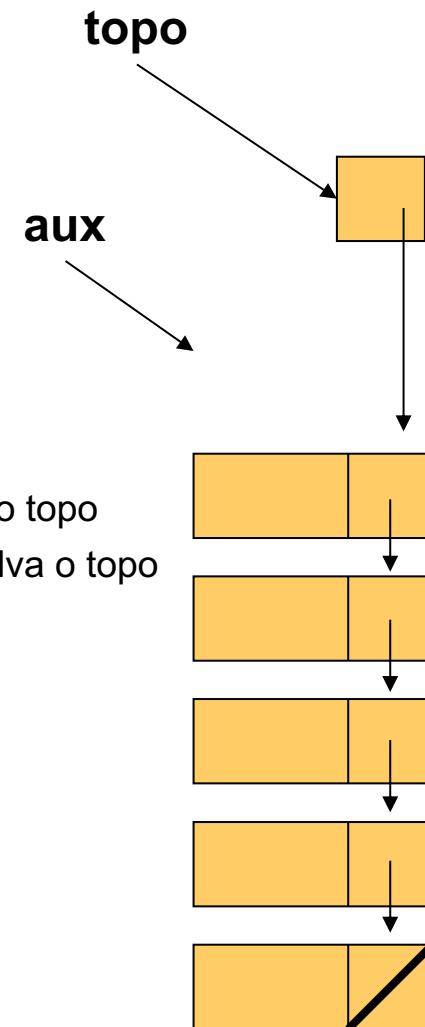
# Remoção em pilha encadeada

```
int popPilha (pilha **topo, info *dados)
{
    se pilha é vazia
        retorna 0
    senão
        topo
            armazena em dado o valor do topo
            declara ponteiro auxiliar e salva o
            atualiza o topo
            retorna 1
}
```



# Remoção em pilha encadeada

```
int popPilha (pilha **topo, info *dados)
{
    se pilha é vazia
        retorna 0
    senão
        armazena em dado o valor do topo
        declara ponteiro auxiliar e salva o topo
        atualiza o topo
        libera memória do auxiliar
        retorna 1
}
```



# Consulta à pilha encadeada

```
info topPilha (pilha *topo)
{
    se pilha é vazia
        retorna 0;
    senão
        retorna o valor do topo
}
```

# **Destruição de uma pilha encadeada**

```
pilha* destroiPilha (pilha *topo)
{
    ?
}
}
```

# Leitura Recomendada

- Capítulo 4 do livro
  - Nina Edelweiss, Renata Galante. Estruturas de Dados. Bookman, 2009.

# Exercícios

1. Descreva o resultado e estado da pilha resultante a cada operação da seguinte série: new, top, push(5), push(3), pop(), top, push(2), push(8), pop(), pop(), push(9), push(1), top, pop(), push(7), push(6), pop(), pop(), top, push(4), pop(), pop().
2. Implemente, utilizando encadeamento, os tipos e operações vistas em aula para o TAD pilha de valores inteiros. Escreva um programa que teste todas as operações do TAD.
3. Especifique e implemente em C uma função para testar se duas pilhas são iguais. Utilize apenas as funções básicas do TAD implementadas no exercício anterior.

# Filas

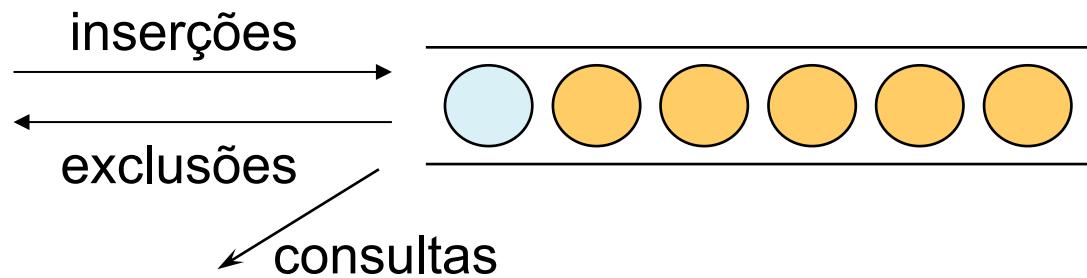
Listas Lineares Especiais

# Filas

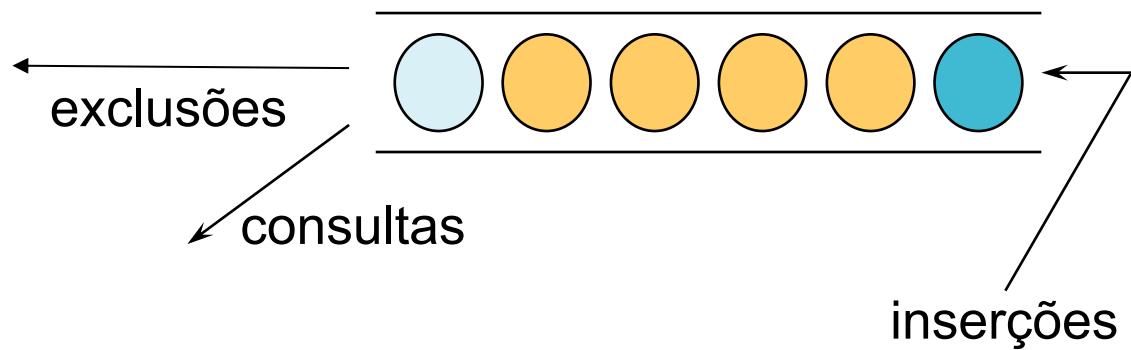
Listas Lineares  
Especiais

- Possuem disciplina restrita de acesso
  - Somente primeiro e último nó (início e fim)
- Restrições nas operações
  - Inserção
  - Remoção
  - Consulta
- Comportamento
  - First In, First Out (FIFO)
    - O primeiro elemento inserido é o primeiro a ser retirado

## Pilhas:

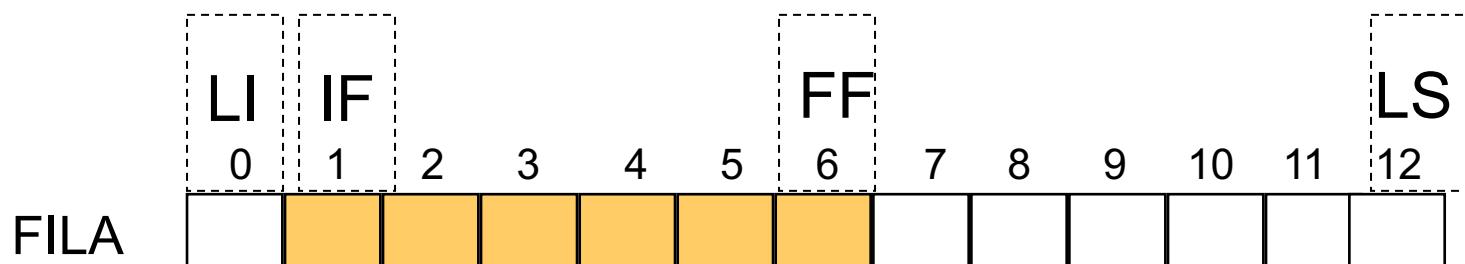
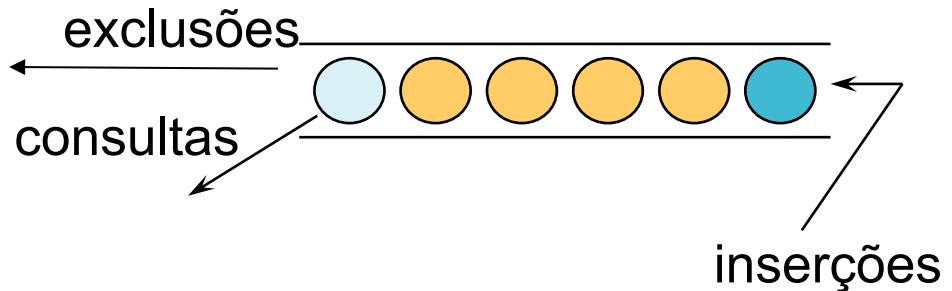


## Filas:



# Filas

- Contiguidade física
- Implementada sobre arranjo



LI : limite inferior do arranjo

IF : início da fila

FF : final da fila

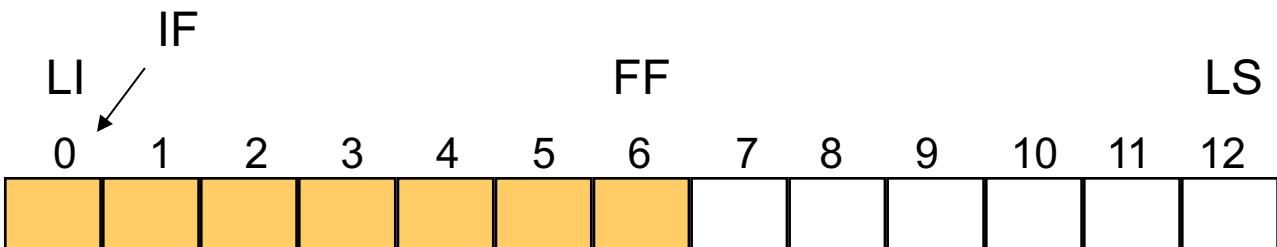
LS : limite superior do arranjo

Fila vazia

IF = -1

# Evolução da Fila

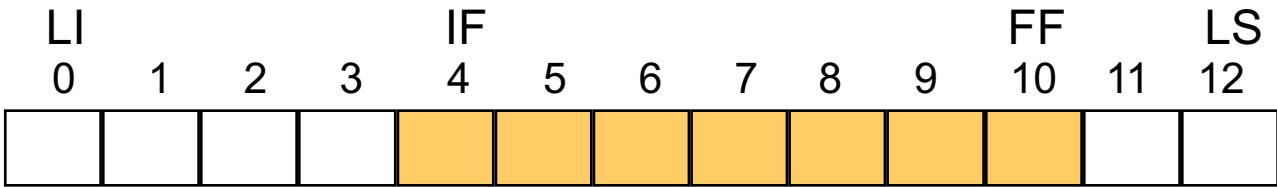
FILA 1



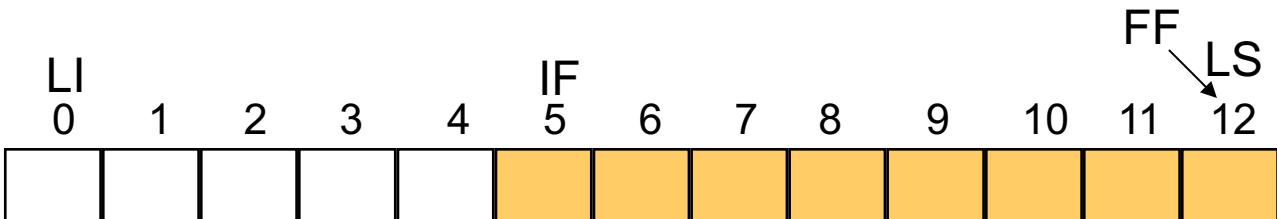
FILA 2



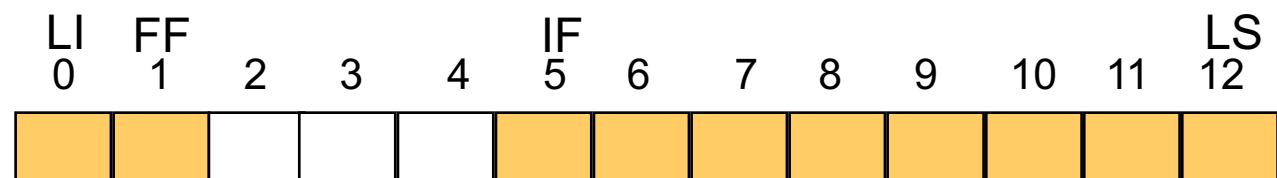
FILA 3



FILA 4



FILA 5



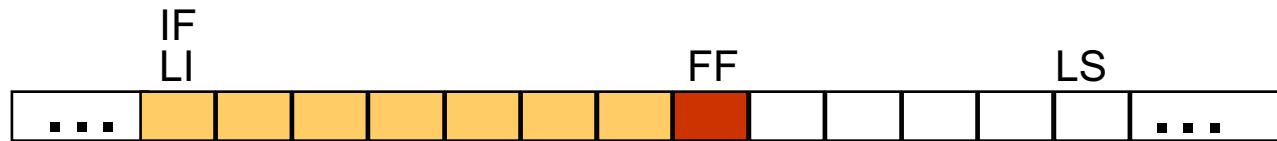
Lista circular

# Operações sobre Filas

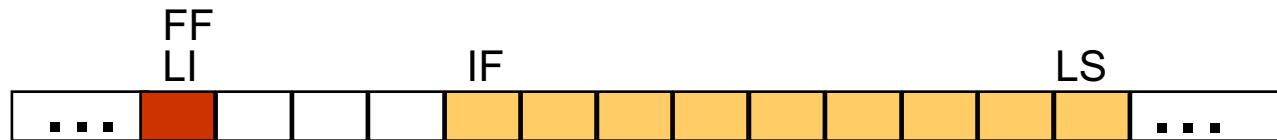
- Operações válidas:
  - criar uma fila vazia
  - inserir um nó no final da fila
  - excluir o nó do início da fila
  - consultar nó do início da fila
  - destruir a fila

# Inserção em fila - contiguidade física

- Testar se tem espaço livre para inserir:

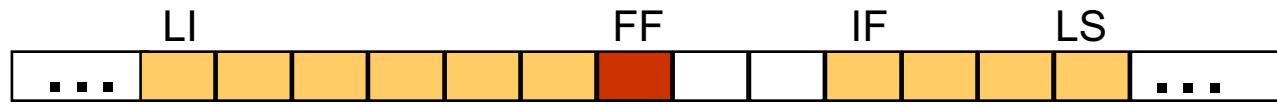


atrás



na frente

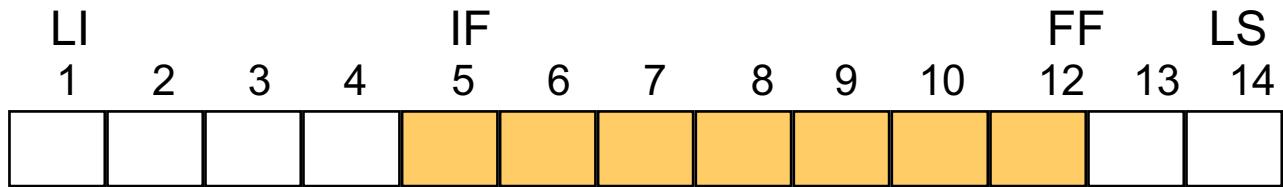
no meio



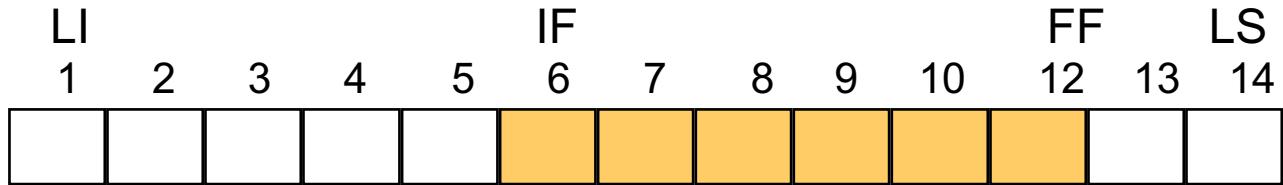
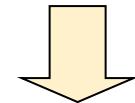
# Remoção de fila

contiguidade física

FILA

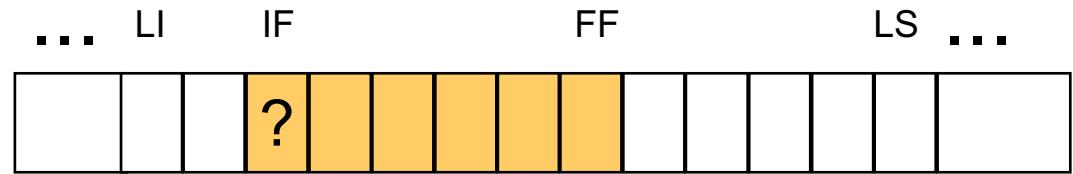


FILA

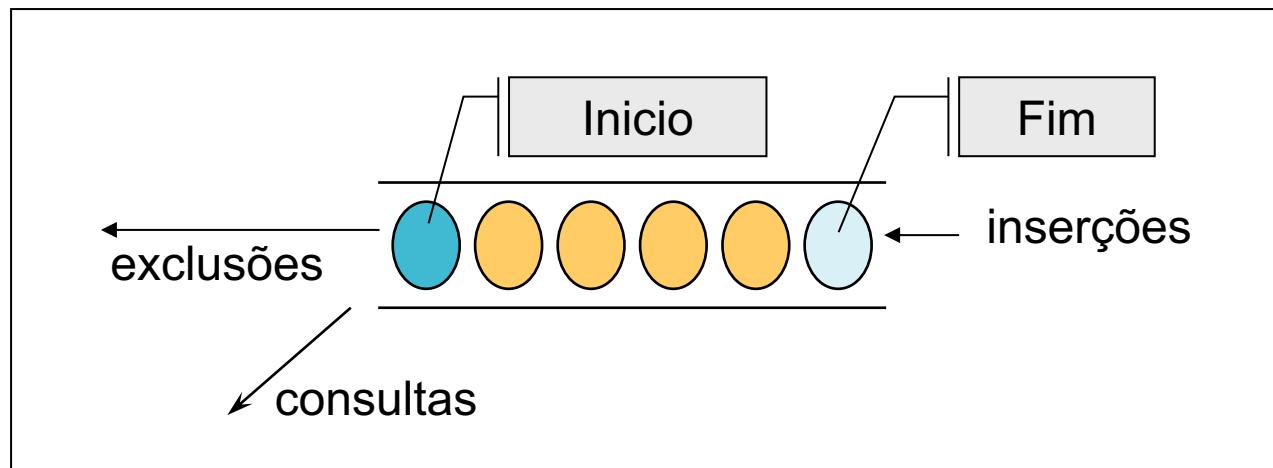


# Consulta a fila

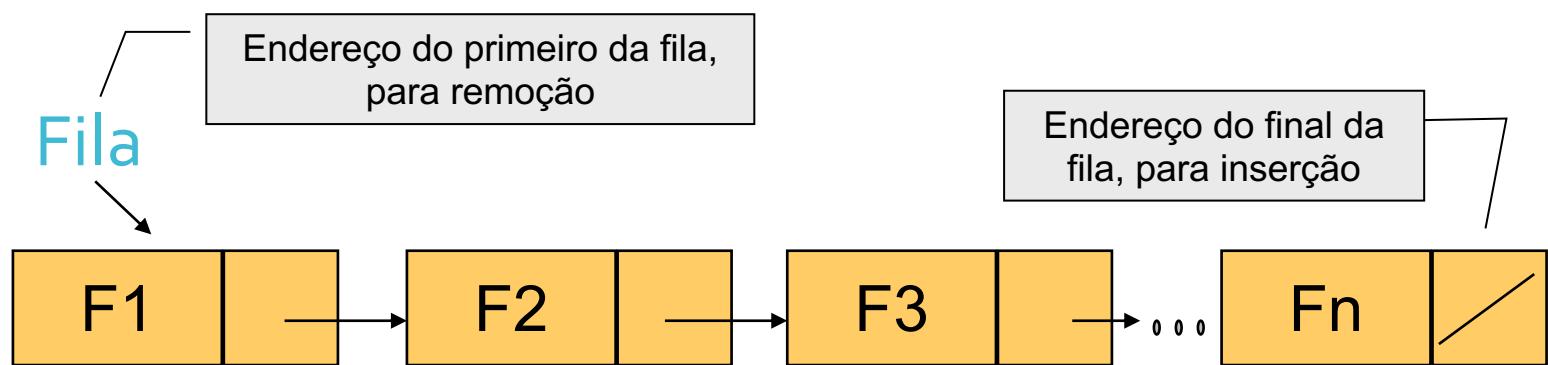
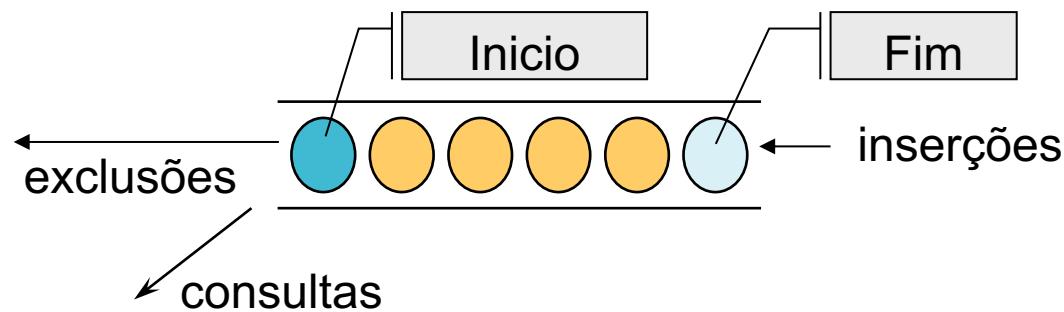
contiguidade física



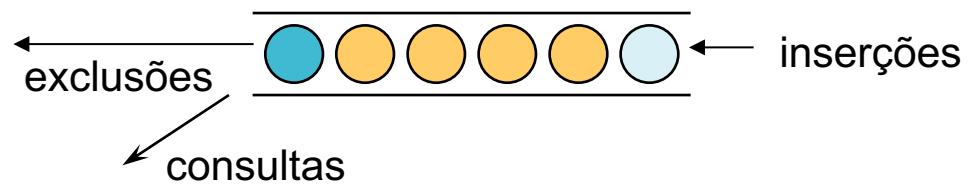
# Filas encadeadas



# Filas encadeadas

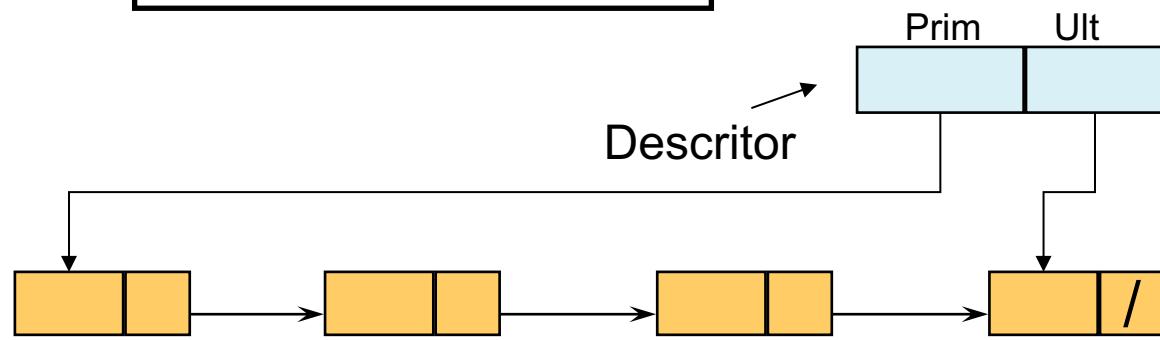


# Filas encadeadas com descriptor



Descriptor

*tipo descriptorFila*  
prim: primeiro da fila  
ult : último da fila



# Especificação da Estrutura de Dados

```
/* tipo descriptorFila abstrai a estrutura do descritor de fila*/
typedef struct descriptor descriptorFila;

/* tipo info abstrai o elemento de dado a ser enfileirado */
typedef int info;

/* tipo fila abstrai a estrutura de um nodoFila */
typedef struct nodoFila fila;

/* implementacao do nodoFila */
struct nodoFila{ info dado; fila *elo; };

/* implementacao do descritor de fila */
struct descriptor{ fila *prim; fila *ult; };
```

# Operações

```
/* aloca e retorna um descritor com primeiro e ultimo nodos nulos */
descritorFila* inicializaFila ();

/* retorna o primeiro elemento da fila */
info consultaFila (descritorFila *df);

/* retorna 1 se conseguiu enfileirar um dado, 0 caso contrario */
int insereFila(descritorFila *df, info dado);

/* tenta remover um elemento da fila df e colocar em dado */
/* retorna 1 se conseguiu, 0 caso contrario */
int removeFila (descritorFila *df, info *dado);

/* desaloca a memoria (inclusive do descritor) */
void destroiFila (descritorFila *df);
```

# Criação de fila encadeada com descritor

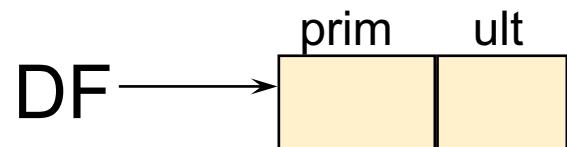
```
/* aloca e retorna um descritor com primeiro e
ultimo nodos nulos */

descritorFila* inicializaFila () {

}
```

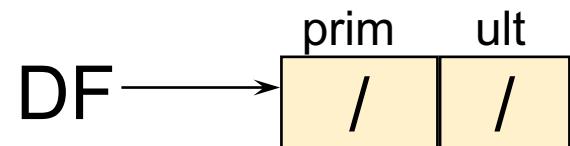
# Criação de fila encadeada com descritor

```
/* aloca e retorna um descritor com primeiro e ultimo  
nodos nulos */  
  
descritorFila* inicializaFila () {  
    declara e aloca o descritor  
    se alocação bem sucedida  
  
    senão  
        erro de alocacao  
    }  
  
}
```



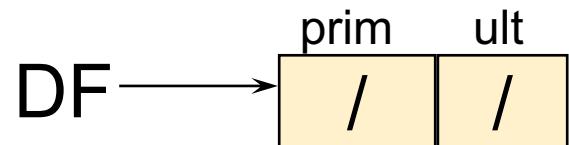
# Criação de fila encadeada com descritor

```
/* aloca e retorna um descritor com primeiro e ultimo  
nodos nulos */  
  
descritorFila* inicializaFila () {  
    declara e aloca o descritor  
    se alocação bem sucedida  
        Inicializa prim e ult com valores nulos  
  
    senão  
        erro de alocacao  
}
```



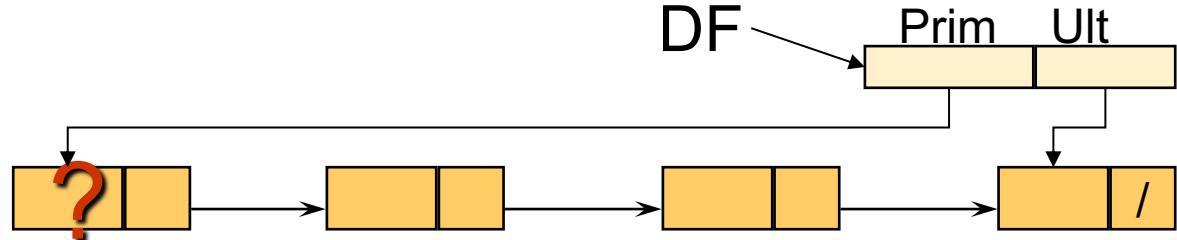
# Criação de fila encadeada com descriptor

```
/* aloca e retorna um descriptor com primeiro e ultimo  
nodos nulos */  
  
descriptorFila* inicializaFila () {  
    declara e aloca o descritor  
    se alocação bem sucedida  
        Inicializa prim e ult com valores nulos  
        retorna o endereço do descritor alocado  
    senão  
        erro de alocacao  
}
```



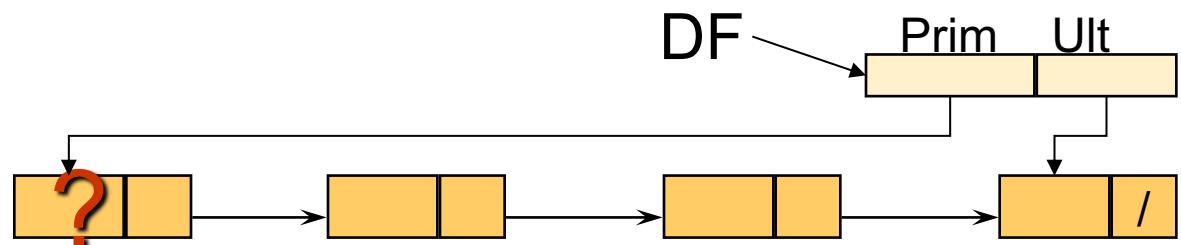
# Consultar a fila encadeada com descriptor

```
/* retorna o primeiro elemento da fila */
info consultaFila (descriptorFila *df) {
    se fila não é vazia
        senão
    }
}
```



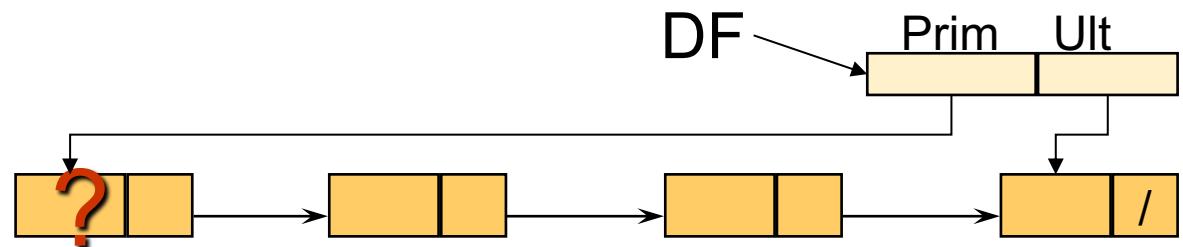
# Consultar a fila encadeada com descriptor

```
/* retorna o primeiro elemento da fila */  
info consultaFila (descriptorFila *df) {  
    se fila não é vazia  
        retornar o dado do primeiro elemento  
    senão  
}  
}
```

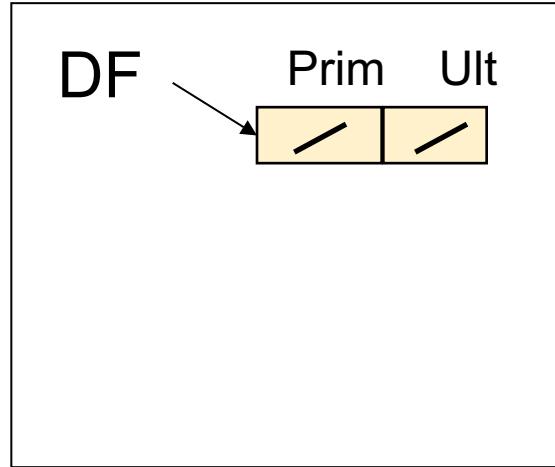
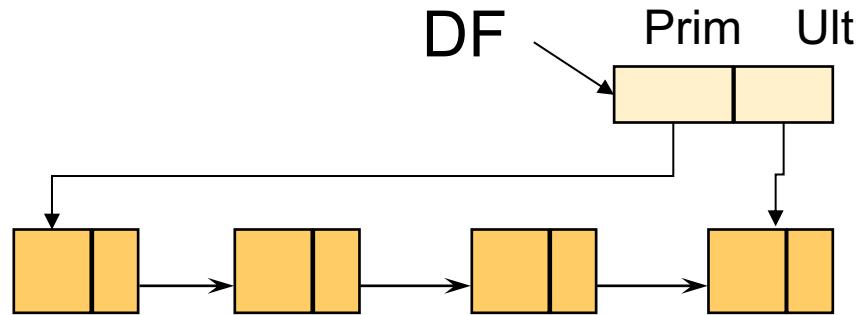


# Consultar a fila encadeada com descriptor

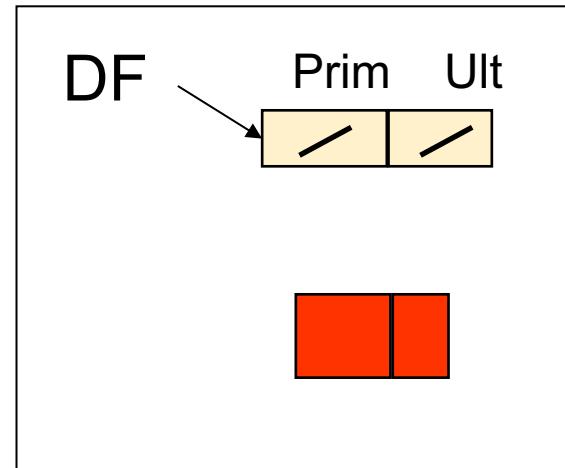
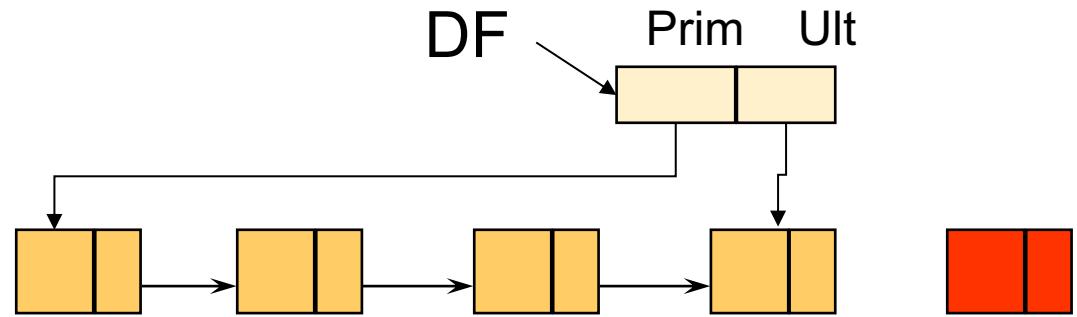
```
/* retorna o primeiro elemento da fila */
info consultaFila (descriptorFila *df) {
    se fila não é vazia
        retornar o dado do primeiro elemento
    senão
        imprimir “vazia”
        retornar info padrão (ex. 0)
}
```



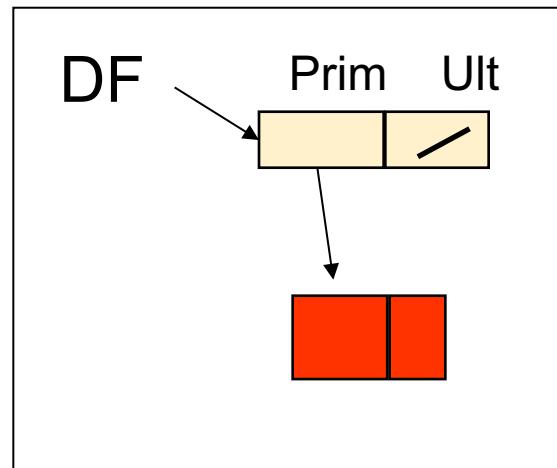
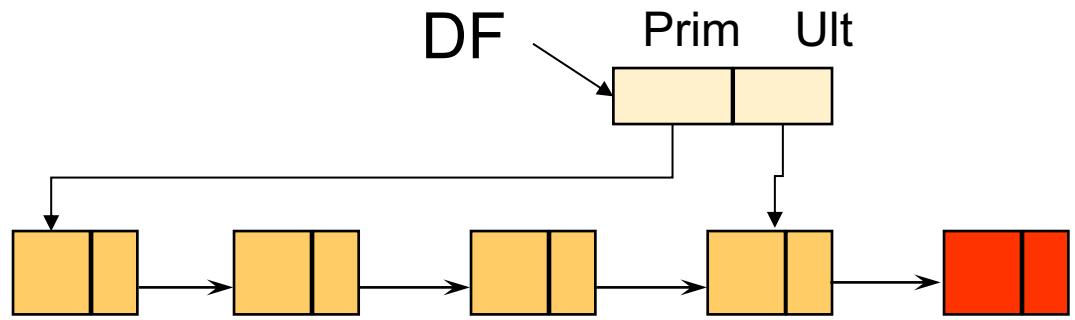
# Inserção em fila encadeada com descritor



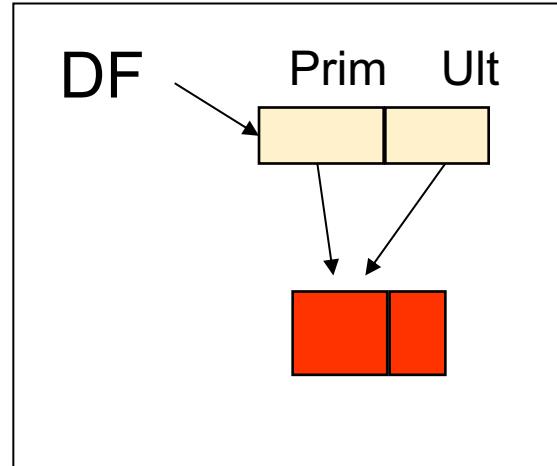
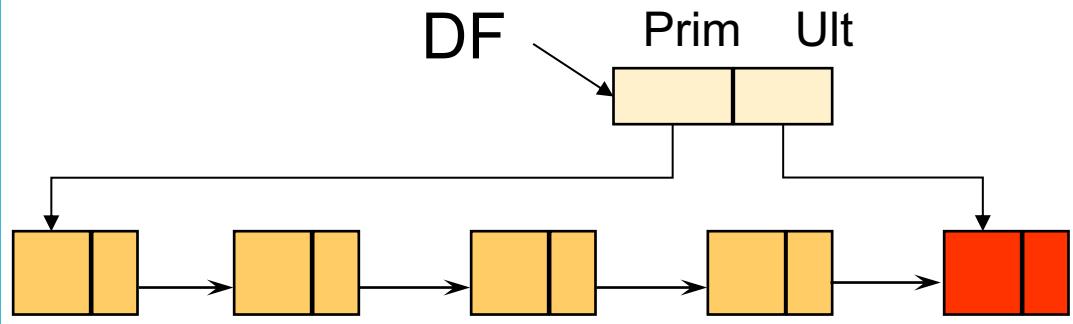
# Inserção em fila encadeada com descritor



# Inserção em fila encadeada com descritor

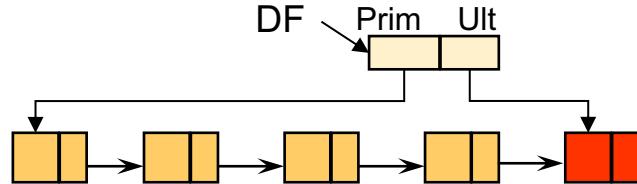


## Inserção em fila encadeada com descritor

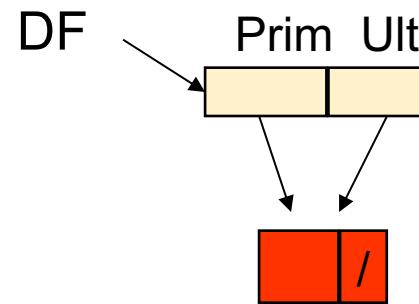
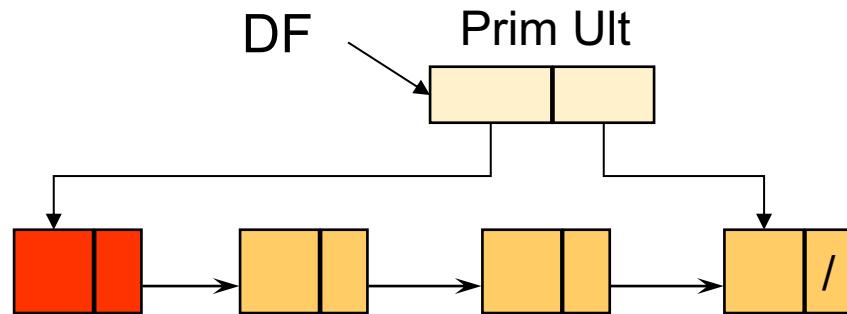


# Inserção em fila encadeada com descritor

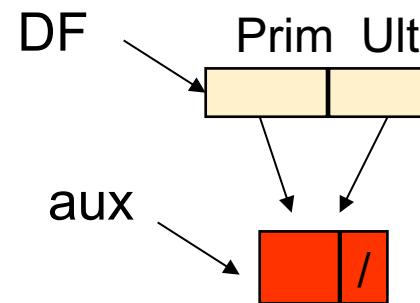
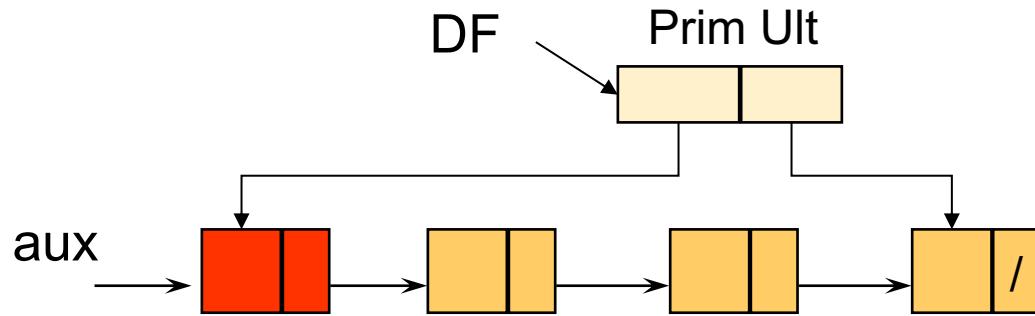
```
/* retorna 1 se conseguiu enfileirar um dado, 0 caso contrario */  
int insereFila(descritorFila *df, info dado){  
    declarar e alocar nodo  
    se alocação bem sucedida  
        copiar dado dentro do nodo alocado  
        anular referencia do nodo alocado  
        se a fila é vazia apontar prim e ult para o nodo alocado  
        senão  
            atualizar referencia do último nodo para o nodo alocado  
            atualizar o descritor apontando ult para no nodo alocado  
        retornar 1  
    senão  
        indicar erro de alocação  
        retornar 0  
}
```



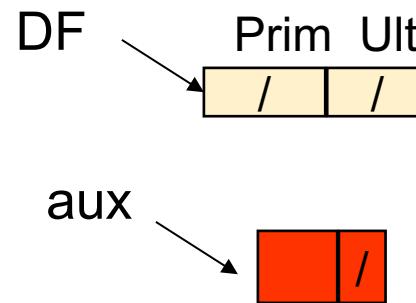
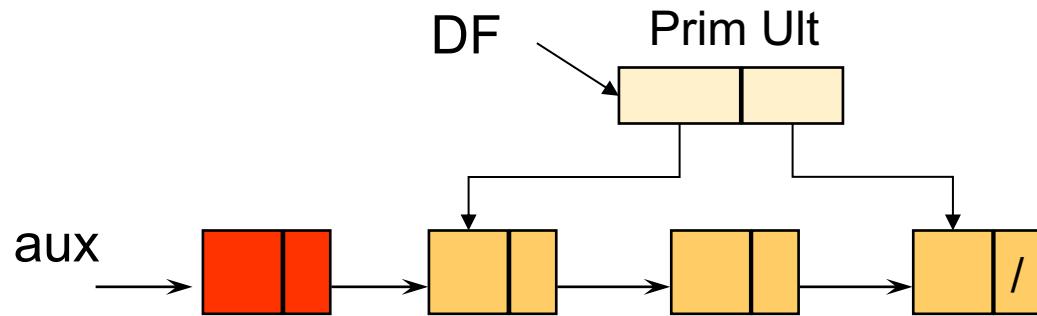
# Remoção de fila encadeada com descriptor



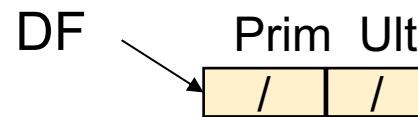
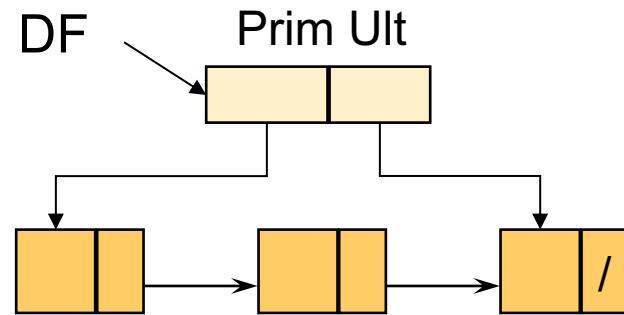
# Remoção de fila encadeada com descriptor



# Remoção de fila encadeada com descritor



# Remoção de fila encadeada com descriptor

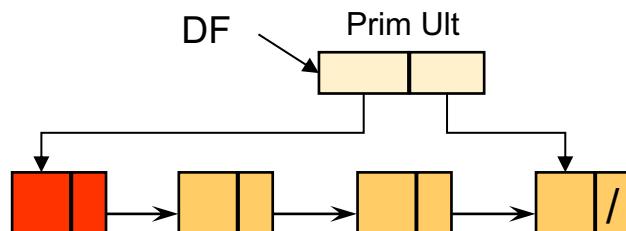


```

/* tenta remover um elemento da fila df e colcar em dado */
/* retorna 1 se conseguiu, 0 caso contrario */

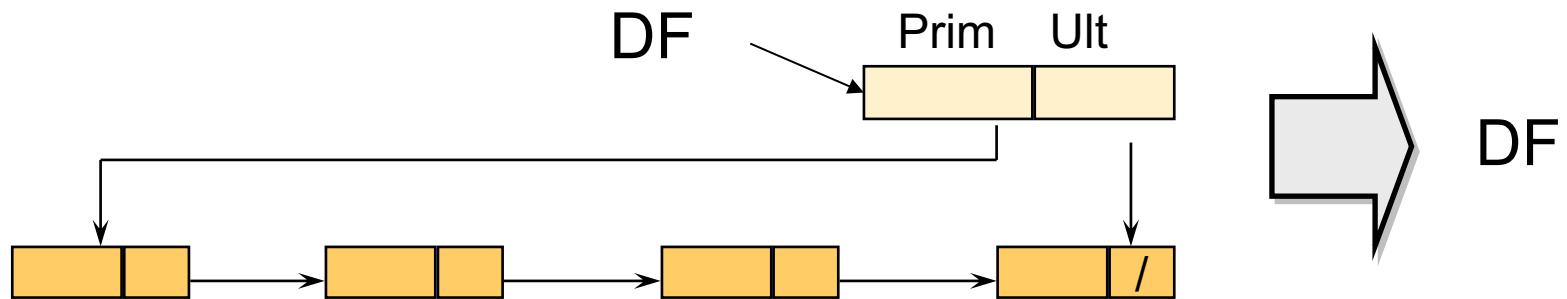
int removeFila (descriptorFila *df, info *dado)
{
    se fila é vazia retornar 0
    senão
        salvar no parâmetro dado o conteúdo do primeiro nodo
        declarar uma fila auxiliar e salvar o endereço do primeiro nodo
        atualizar o descriptor no campo prim
        liberar a memória do nodo removido
        retornar 1
}

```



## Remoção de fila encadeada com descritor

# Destruição de fila encadeada com descritor



*/\* desaloca a memoria (inclusive do descritor) \*/*

```
void destroiFila (descritorFila *df) {  
    declarar fila auxiliar e salvar o endereço do primeiro nodo  
    enquanto fila auxiliar não for vazia  
        atualizar o primeiro nodo do descritor  
        liberar nodo apontado pelo auxiliar  
        atualizar auxiliar com endereço do primeiro nodo  
        liberar memória ocupada pelo descritor  
}
```

## Leitura Recomendada

- Capítulo 4 do livro
  - Nina Edelweiss, Renata Galante. Estruturas de Dados. Bookman, 2009.

# EXERCÍCIOS

1. Dadas duas filas implementadas sobre dois arranjos, escreva um algoritmo que devolva o maior dos dois elementos na frente destas filas, removendo este elemento da fila correspondente. O algoritmos deverá receber como parâmetros, os índices de início e de final de cada uma das filas, e atualizar o índice de início da fila da qual foi removido o elemento.
2. Implemente um algoritmo que receba três filas,  $F$ ,  $F_{\text{Ímpares}}$  e  $F_{\text{Pares}}$ , e separe todos os valores guardados em  $F$  de tal forma que os valores pares são movidos para a fila  $F_{\text{Pares}}$  e os valores ímpares são movidos para a fila  $F_{\text{Ímpares}}$