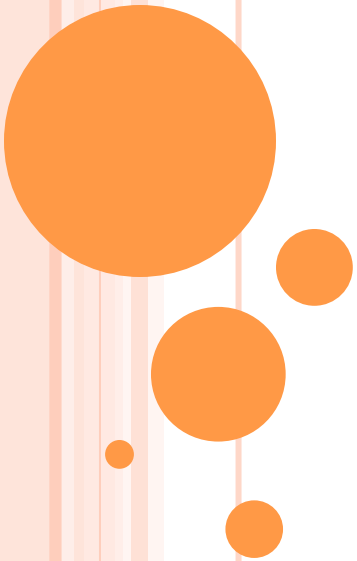


Programação Estruturada

Pilhas, Filas e Listas

Listas Lineares

- Uma lista agrupa um conjunto de elementos que se relacionam entre si
- Uma lista linear é um conjunto de $n \geq 0$ elementos:

$$L[1], L[2], L[3], \ldots, L[n]$$

- Tais que:
  - Se n > 0, então `L[1]` é o primeiro elemento
  - Para $1 < k \leq n$, o elemento `L[k]` é precedido por `L[k-1]`
- As operações principais de uma lista são: *inclusão*, *remoção* e *busca* de elementos

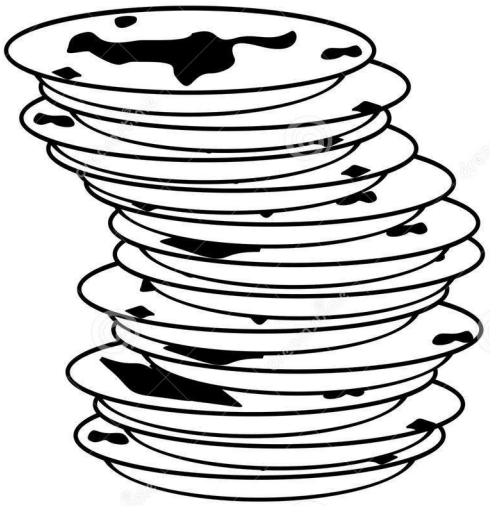
Listas Lineares

- Casos particulares de listas são de especial interesse
  - São listas de acesso restrito, onde as inserções e remoções de elementos são realizadas apenas nos extremos das estruturas
  - Pilhas
    - A inclusão e remoção de elementos é realizada em um único extremo da lista: TOPO
  - Filas
    - A inclusão de elementos é realizada em um extremo, FINAL, e as remoções são realizadas em outro extremo, INÍCIO

Pilhas e Filas

Insere prato

Remove prato



Sai da fila

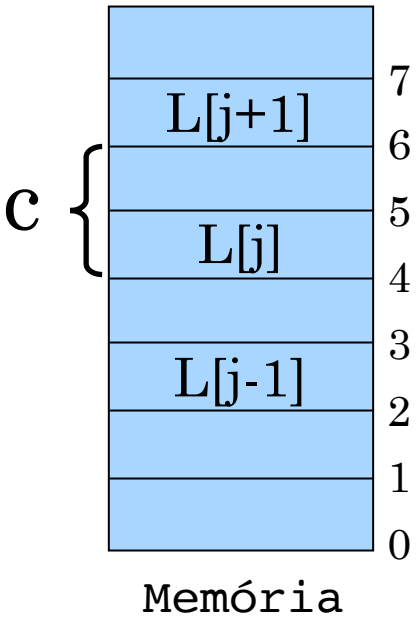
Entra na fila

Listas Lineares - Vetores

- Podemos implementar as listas lineares utilizando vetores
  - É a maneira mais simples
  - Os elementos são armazenados em posições consecutivas na memória
  - É feita uma reserva prévia de memória
  - Problema com dimensionamento
  - Inclusão e remoção de elementos não ocorrem de fato
  - Mas, é possível ter acesso direto a qualquer elemento da lista

Lista Lineares - Vetores

- O endereço real do (j+1)-ésimo elemento da lista se encontra *c* unidades adiante daquele correspondente ao j-ésimo.
- A constante *c* é o número de palavras de memória que cada elemento ocupa.
- Se L[j] está no endereço *i*, L[j+1] está no endereço *i*+*c*.

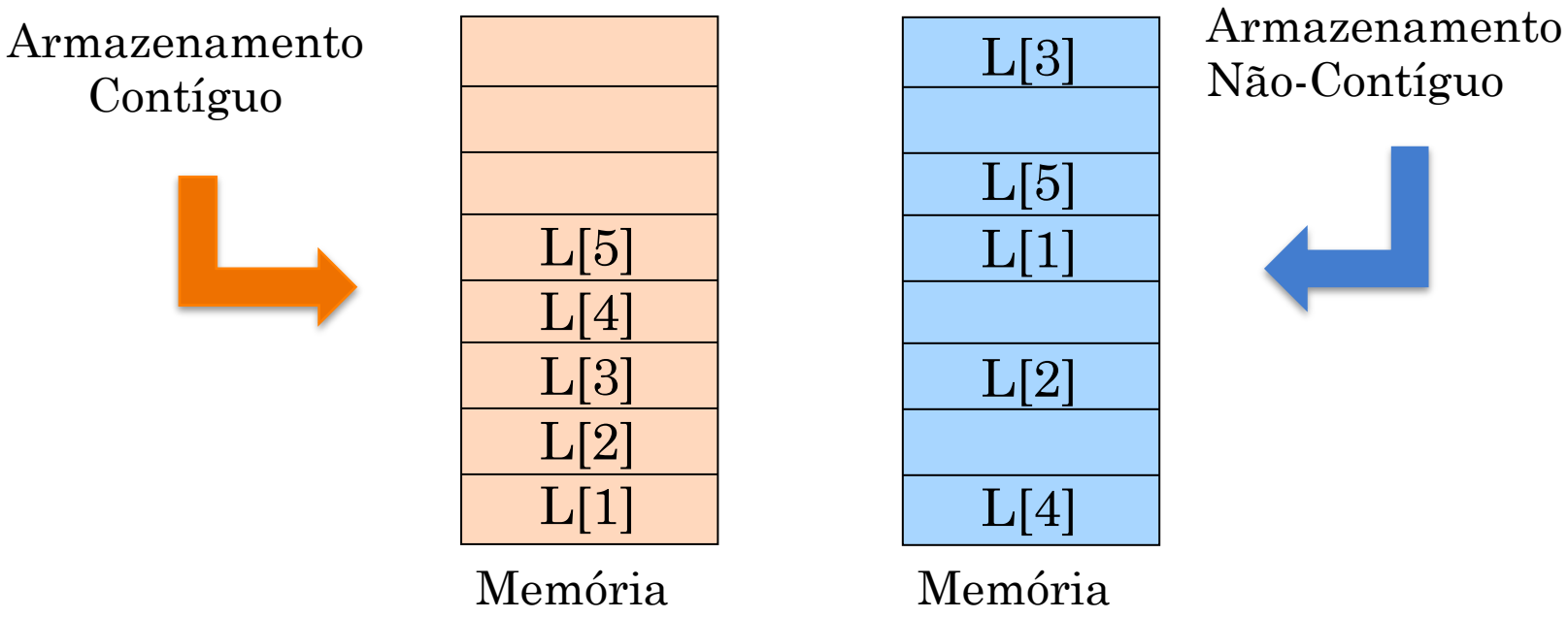


Listas Lineares - Ponteiros

- Um outro tipo de implementação de listas lineares é utilizando *alocação encadeada dinâmica*
  - As posições de memória para cada elemento da lista são alocadas a medida que se tornam necessárias
  - Ou seja, a estrutura pode aumentar e diminuir em tempo de execução
    - As inclusões e remoções de elementos ocorrem de fato, com alocação e liberação de memória
  - Não existe acesso direto a um elemento da lista

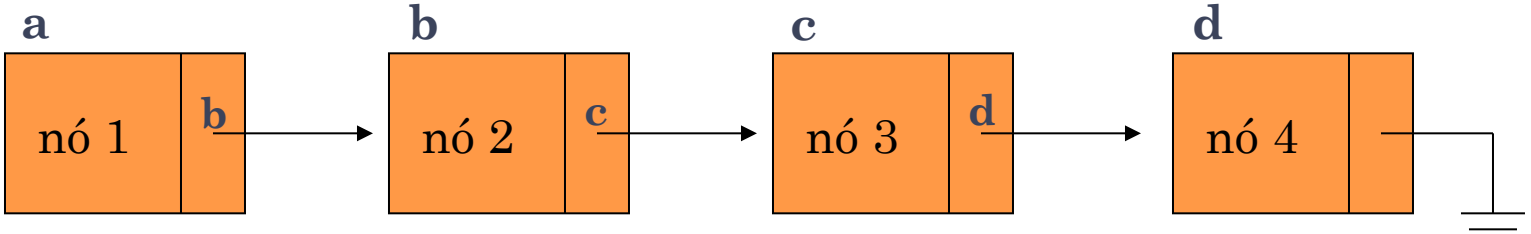
Lista Lineares

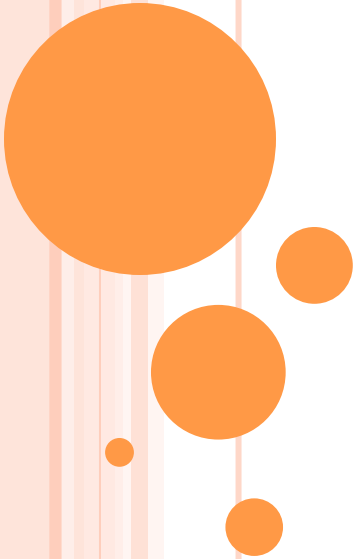
- Armazenamento Contígo x Não Contíguo



Listas Lineares - Ponteiros

- O elementos são chamados de nós
- Como os elementos não se encontram em posições contíguas de memória é necessário que cada elemento saiba o endereço do próximo
- Logo, cada nó é composto por duas partes:
  - A informação do elemento
  - O endereço do próximo nó → ponteiro





Exemplo de Implementação de Pilhas e Filas com Vetores

Pilhas e Filas

- Normalmente, o armazenamento sequencial de listas é empregado quando estas sofrem poucas inserções ou remoções ao longo do tempo.
- Em casos particulares como o de pilhas e filas tal armazenamento também pode ser usado com eficiência
- Nestes casos, inserções e remoções não acarretam a movimentação dos nós, já que estas operações ocorrem nos extremos da estrutura

Pilhas

- São estruturas do tipo LIFO (*last in – first out*)
- Utiliza um marcador especial que deve indicar a posição do topo, já que remoções e inserções são realizadas em um mesmo extremo.
- Pode se definir operações básicas para a manipulação de pilhas como: PUSH e POP.
  - PUSH(P, x): insere um elemento x no topo da pilha P (empilha).
  - POP(P): retira o elemento do topo da pilha P (desempilha).

PILHAS

- Implementação de uma pilha inteiros:

```
#define MAX 100;

struct PILHA {
  int dado[MAX];
  int topo;
};

typedef struct PILHA pilha;
```

```
int main(void) {

   pilha P;
   P.topo = -1;
   ...
}
```

Pilhas

- Representação gráfica:

  - P.dado: vetor que guarda os elementos da pilha
  - P.topo: variável que indica o índice do vetor onde se encontra o topo

- Quando se insere ou remove um elemento na pilha a variável topo deve ser atualizada.

PILHAS

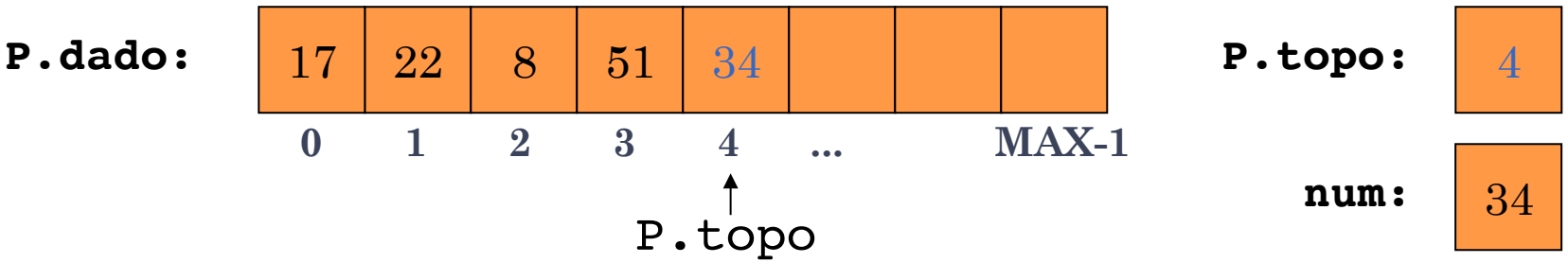
- Função para inserir um elemento no topo da pilha

```
void push(pilha *P, int num) {
    if (P->topo < MAX-1) {
        P->topo++;
        P->dado[P->topo] = num;
    }
}
```

Pilhas

- Função para inserir um elemento no topo da pilha

```
void push(pilha *P, int num) {
    if (P->topo < MAX-1) {
        P->topo++;
        P->dado[P->topo] = num;
    }
}
```



Pilhas

- Função para remover um elemento no topo da pilha

```
int pop(pilha *P) {
    int num;
    if (P->topo >= 0) {
        num = P->dado[P->topo];
        P->topo--;
    }
    return num;
}
```

Filas

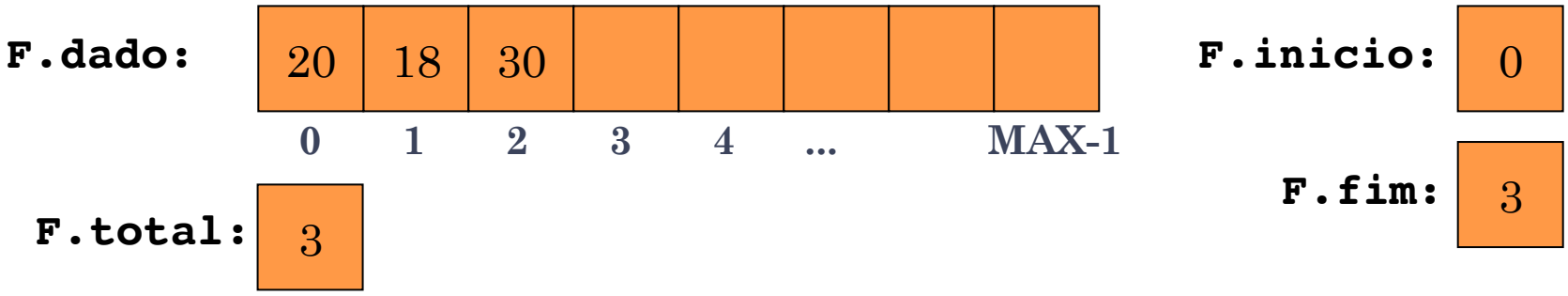
- São estruturas do tipo FIFO (*first in – first out*)
- São necessários dois marcadores: um que indique o inicio da fila e outro que indique o final
- Pode se definir operações básicas para a manipulação de pilhas como: INSERT e REMOVE
  - INSERT(F, x): insere um elemento x no final da fila F
  - REMOVE(F): retira o elemento do início da fila F

Filas

- Implementação de uma fila de inteiros:

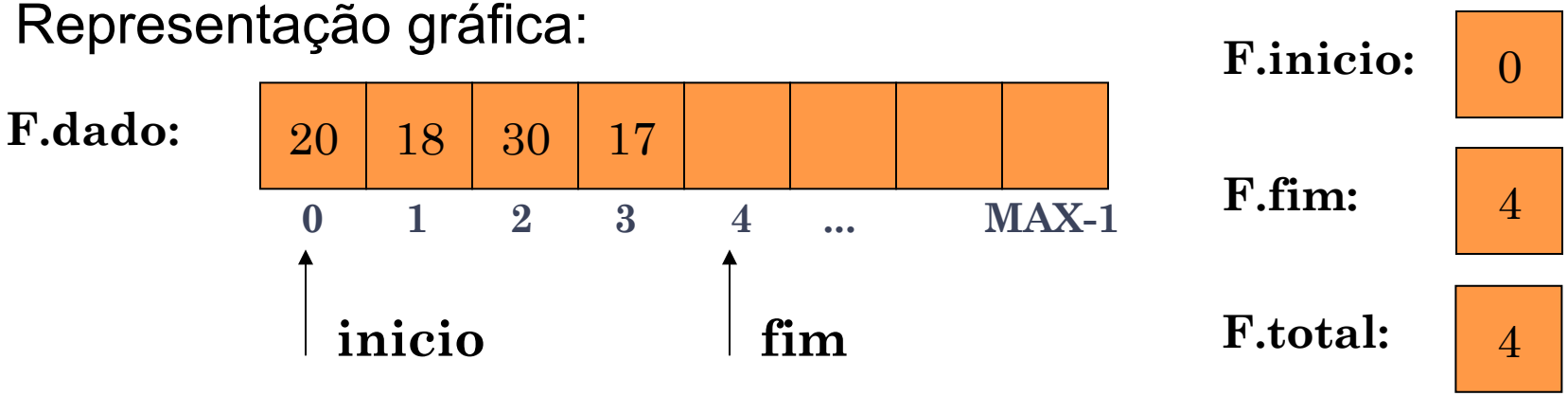
```
#define MAX 100;

struct FILA {
  int dado[MAX];
  int inicio,fim,total;
};

typedef struct FILA fila;
```

```
int main(void) {

   fila F;
   F.inicio = 0;
   F.fim = 0;
   F.total = 0;
   ...
}
```



Filas

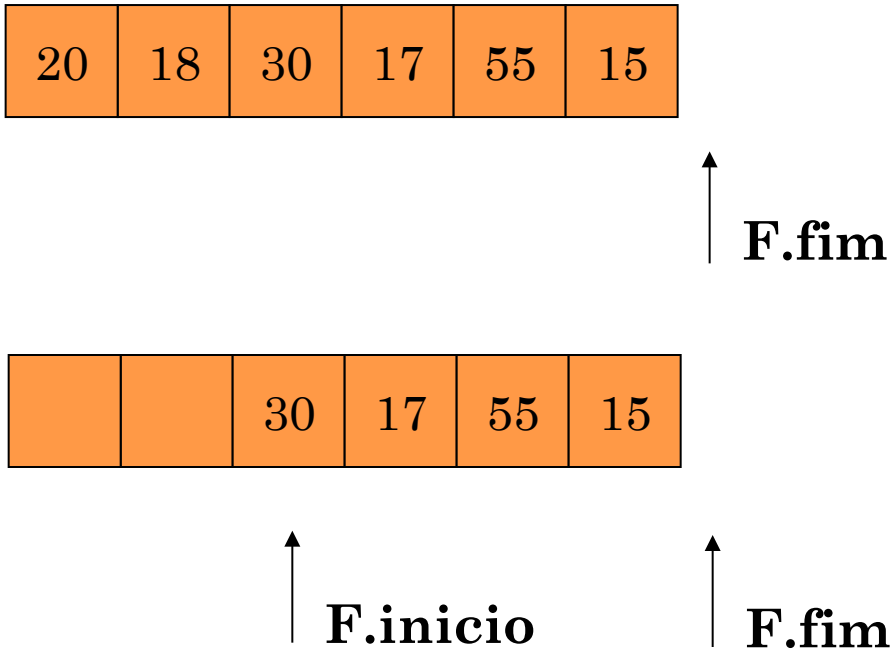
- Representação gráfica:



  - F.dado: vetor que guarda os elementos da fila.
  - F.inicio: indica o índice do vetor onde se encontra o primeiro elemento da fila
  - F.fim: indica o índice do primeiro espaço após o último elemento
  - F.total: indica a quantidade de elementos da fila

Filas

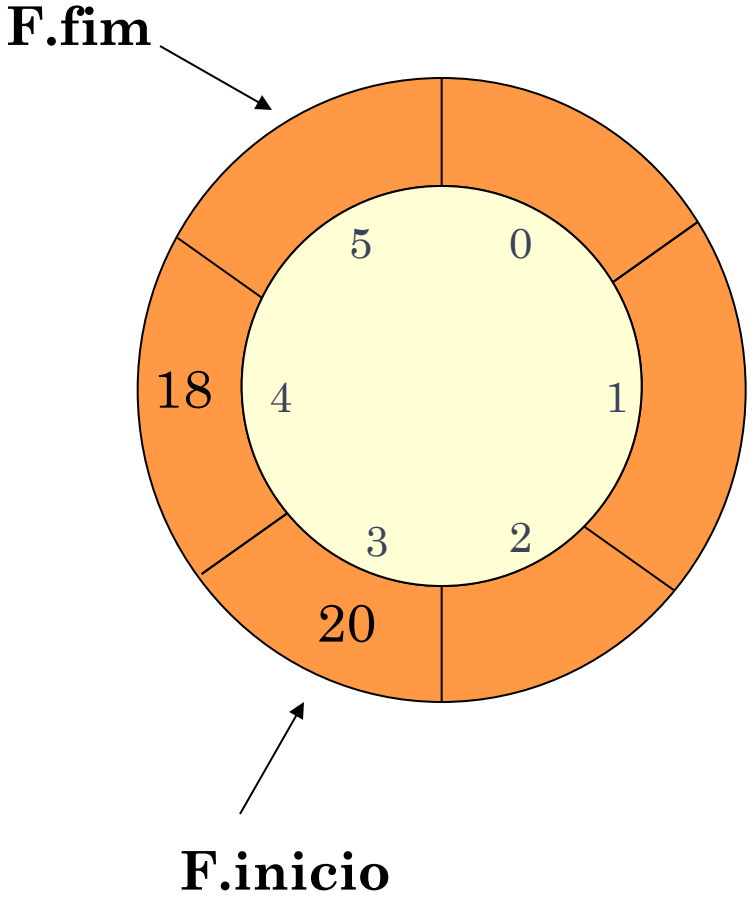
- O problema da implementação



A fila possui espaço mas não é possível inserir elementos, pois, **F.fim == MAX**.

Filas

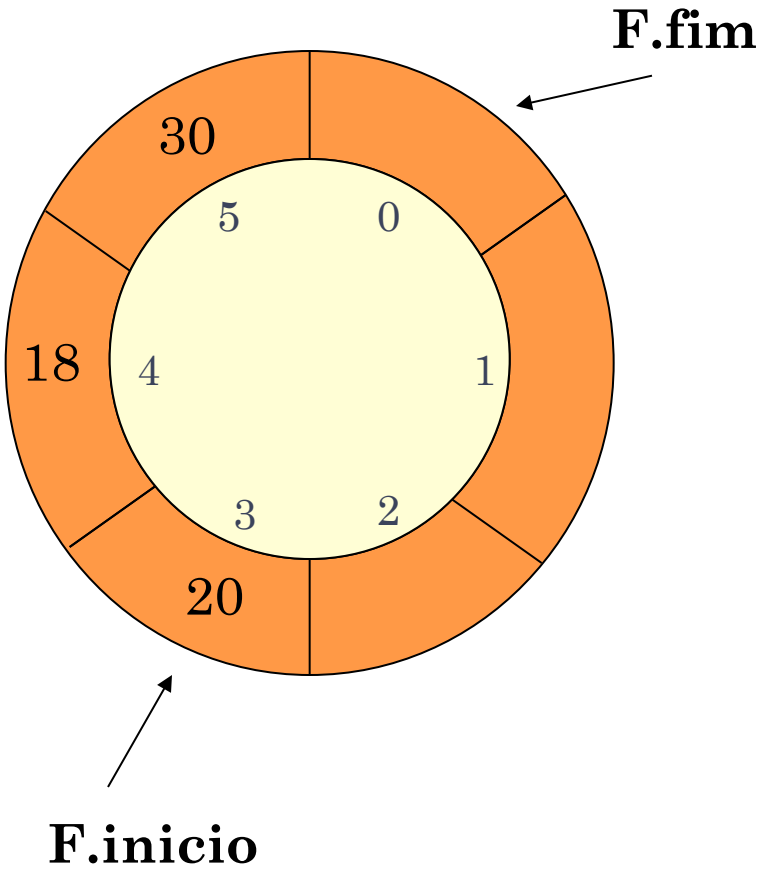
- A solução é usar uma implementação circular



Sempre que os apontadores de fim e de começo ultrapassarem o valor de *MAX*, eles "pulam" para a posição 0.

Filas

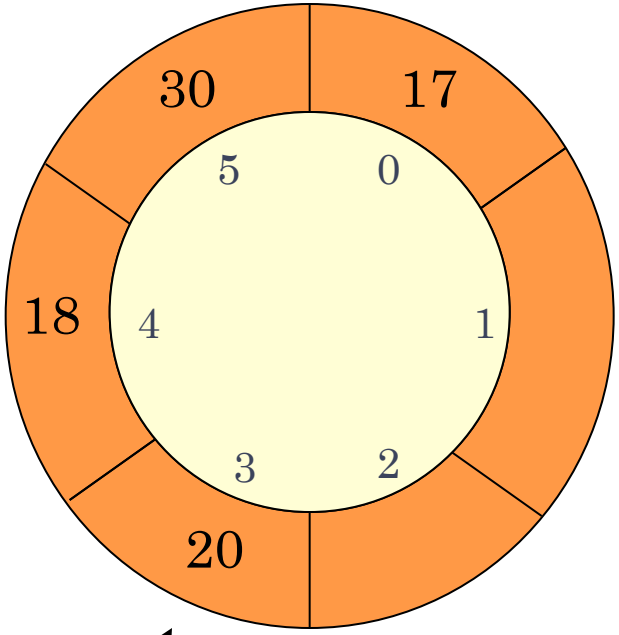
- A solução é usar uma implementação circular



Sempre que os apontadores de fim e de começo ultrapassarem o valor de *MAX*, eles "pulam" para a posição 0.

Filas

- A solução é usar uma implementação circular



F.fim

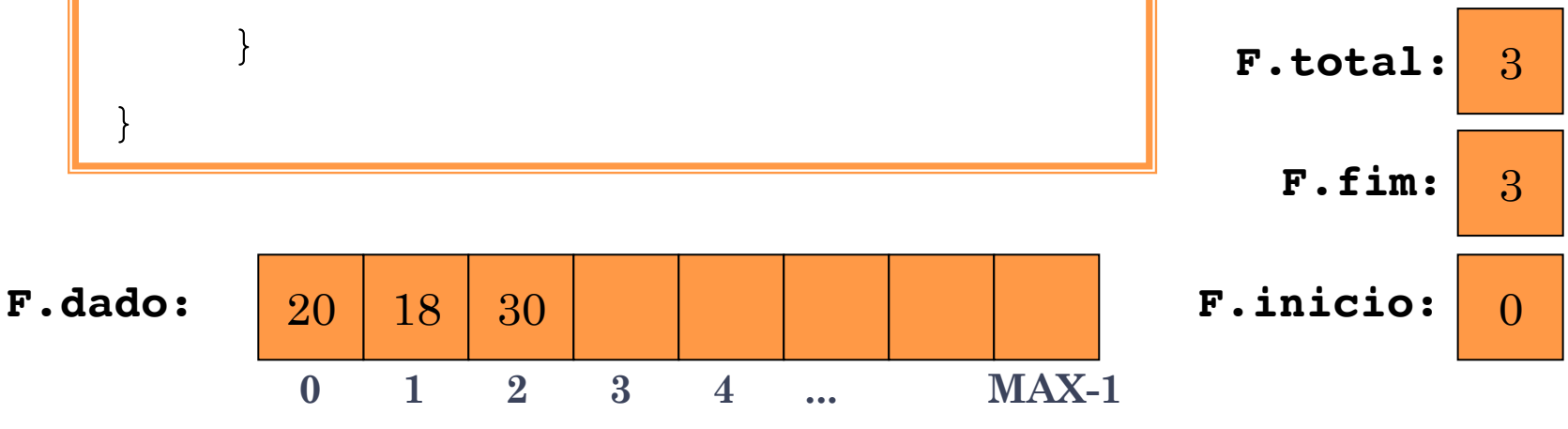
F.inicio

Sempre que os apontadores de fim e de começo ultrapassarem o valor de *MAX*, eles "pulam" para a posição 0.

Filas

- Função para inserir um elemento no final da fila

```
void insere(fila *F, int num) {
    if (F->total < MAX) {
        F->dado[F->fim] = num;
        F->fim++;
        F->fim = F->fim%MAX;
        F->total++;
    }
}
```



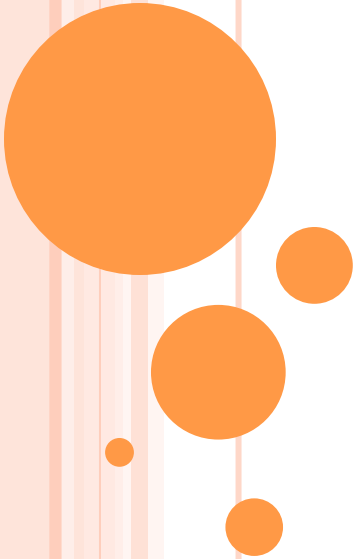
Filas

- Função para remover um elemento do inicio da fila

```
int remove(fila *F) {
    int num;
    if (F->total > 0) {
        num = F->dado[F->inicio];
        F->inicio++;
        F->inicio = F->inicio%MAX;
        F->total--;
    }
    return num;
}
```

Exercício

- Faça um programa que crie uma fila com os N números de processos que estão esperando para executar em um sistema
- Considere que os X (X < N) primeiros processos terminaram a sua execução
- Escreva o nome do primeiro processo da Fila
- Implemente a estrutura de Fila e as funções para inserir e remover elementos

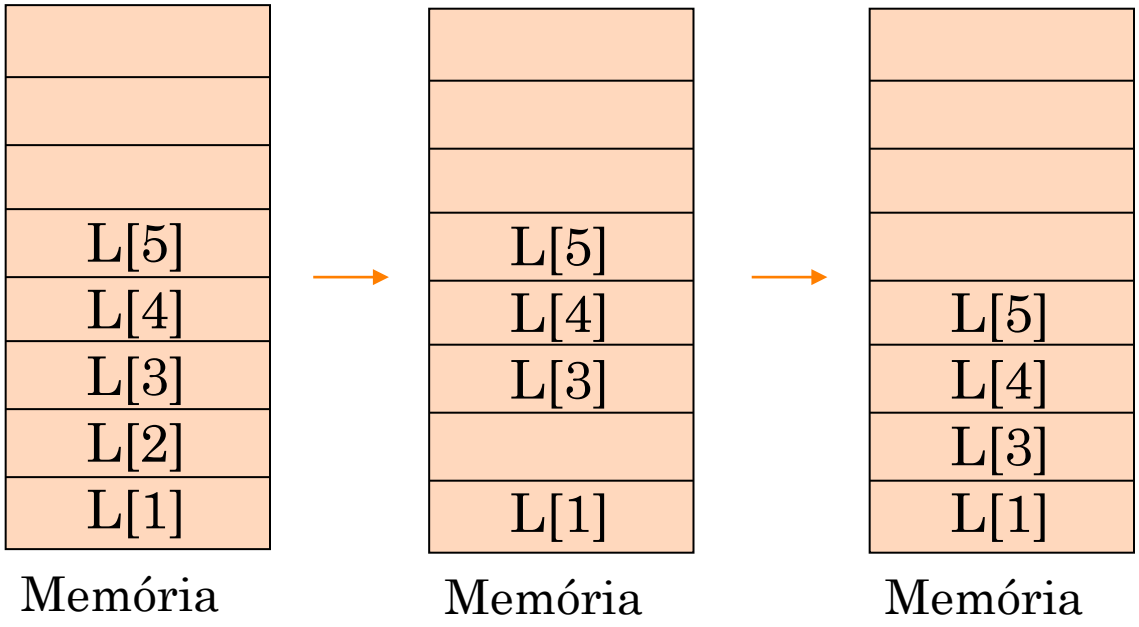


Exemplo de Implementação de Listas com Vetores

Listas Sem Restrições - Vetores

- Inserções e remoções acarretam a “movimentação dos nós”, já que estas operações podem ocorrer com os elementos que estão no “meio” da estrutura

- Exemplo:
  - Remover L[2]



LISTAS

- Implementação de uma lista de inteiros ordenada crescentemente:

```
#define MAX 100;

struct LISTA {
  int dado[MAX];
  int total;
};

typedef struct LISTA lista;
```

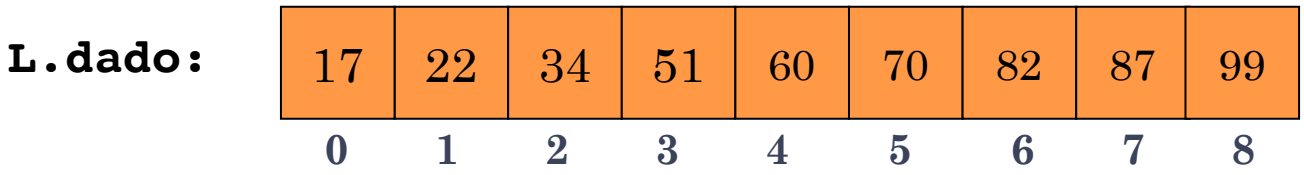
```
int main(void) {

    lista L;
    L.total = 0;
    ...
}
```

Exercícios

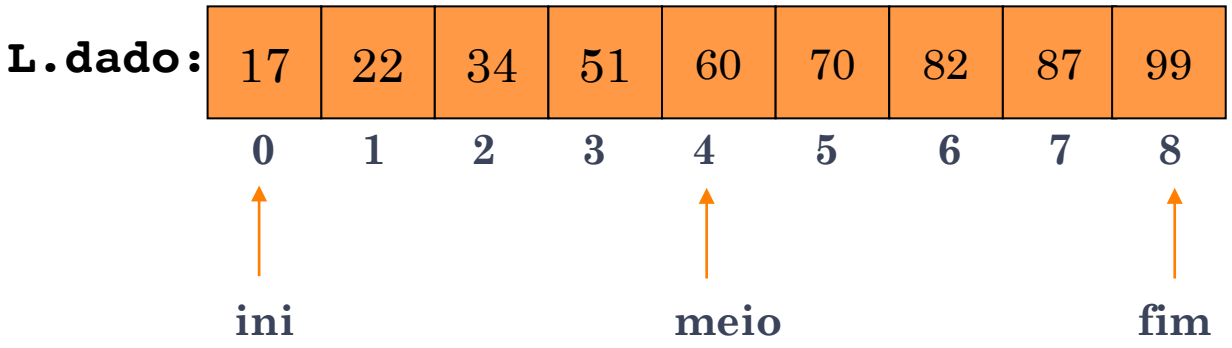
- Faça um programa que crie uma lista L com números inteiros:
  - A lista deve estar ordenada de maneira crescente
  - Crie uma função para Inserir Elementos
    - `void insere (&L, elem)`
  - Crie uma função para Remover Elementos
    - `void remove (&L, elem)`
  - Crie uma função para Buscar Elementos
    - `int busca (&L, elem, &pos)`

Busca Binária

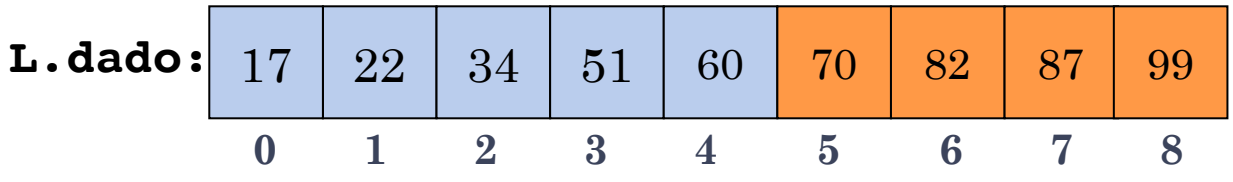
- Como estamos implementando a lista utilizando vetor, e os elementos estão ordenados, é possível implementar um tipo de busca mais eficiente → Busca Binária
- Considere uma lista L com 9 elementos (L.total = 9)



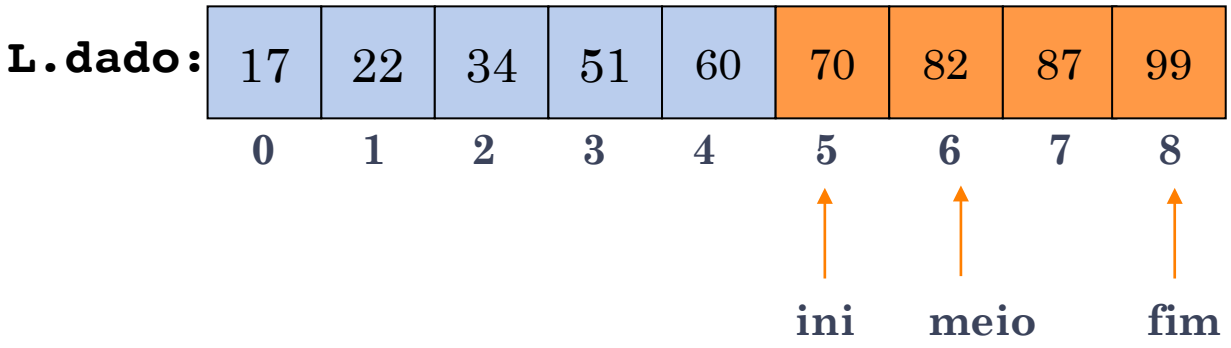
Busca Binária (procurar elem = 70)



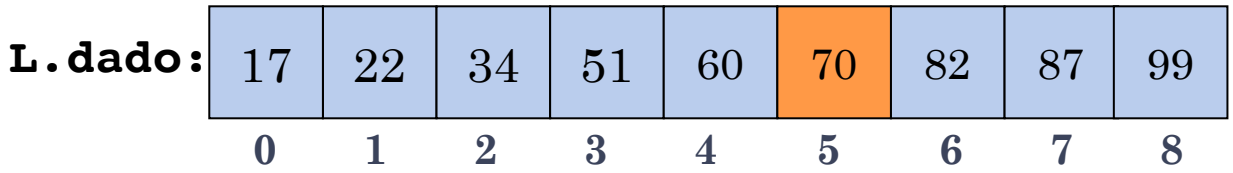
- Compara elem com o meio
- meio = (ini + fim)/2
- `L.dado[meio] == elem` ???
- Neste exemplo `elem > L.dado[meio]`, então a busca prossegue na parte do vetor maior do que meio



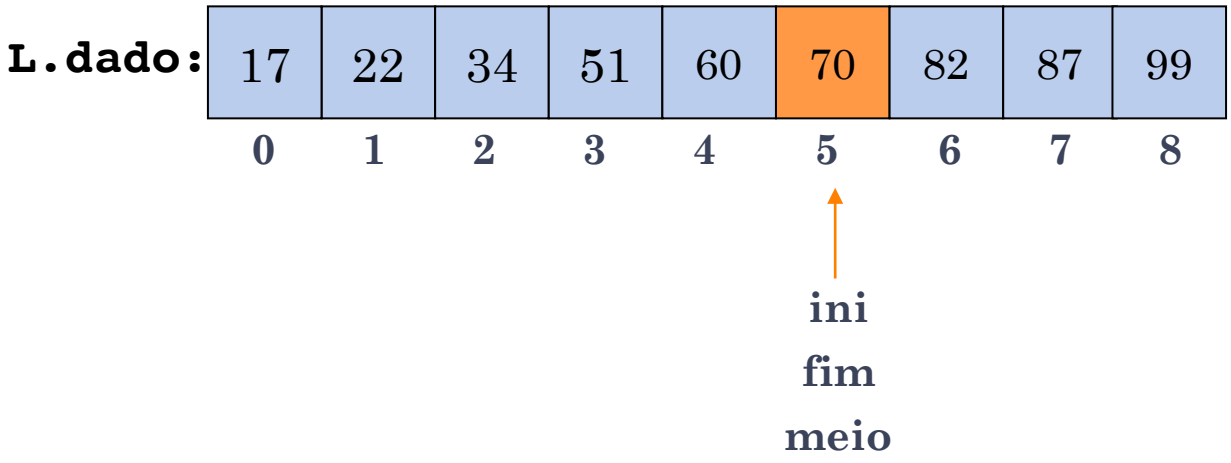
Busca Binária (procurar elem = 70)



- Compara elem com o meio
- meio = (ini + fim)/2
- `L.dado[meio] == elem` ???
- Neste exemplo `elem < L.dado[meio]`, então a busca prossegue na parte do vetor menor do que meio



Busca Binária (procurar elem = 70)



- Compara elem com o meio
- meio = (ini + fim)/2
- `L.dado[meio] == elem` ???
- Neste exemplo `elem == L.dado[meio]`, então a busca termina