# Estrutura de Dados
# TAD FILA

Professora Sandra Rovena Frigeri

# Sumário

- Características do TAD FILA
- Operações Básicas do TAD FILA

# Caracteríticas: TAD FILA


- Conjunto ordenado de elementos
- Critério de ordenação: ordem de inclusão
- Novos elementos: FIM da FILA
- Retirar elementos: INÍCIO da FILA
- Critério FIFO (First-In-First-Out)

# Aplicações do TAD FILA

- Fila de impressão
- Fila de solicitação de serviços
- Fila de processos
- Fila de dados
- Fila de pedidos

# Aplicações do TAD FILA


Solicitações de Impressão

FILA

- Solicitação 1
- Solicitação 2
- Solicitação x
- ....
- Solicitação N

# Operações TAD

- **Operações primitivas:**
  - Dependem da estrutura do TAD
  - Operações básicas do TAD

- **Operações não primitivas de um TAD:**
  - Não dependem da estrutura do TAD
  - Chamam as operações primitivas

# Operações Primitivas TAD FILA

Fila* **cria** ();

- Criar a estrutura de dados da FILA

- Inicializar:
  - Tamanho da FILA
  - Referências para o Início e Fim da FILA

# Operações Primitivas TAD FILA

- int **vazia** (Fila* f);

- Verificar se a FILA está vazia
  - Testar o tamanho da FILA (zero elementos)

  Ou
  - Ou referências para o Início ou Fim da FILA: NULL

# Operações Primitivas TAD FILA

void **libera** (Fila* f);

- Liberar o espaço de memória ocupado pela FILA

# Operações Primitivas TAD FILA

int **tamanho** (Fila* f);

- Consultar e retornar a quantidade de elementos que estão na FILA

# Operações Primitivas TAD FILA

void **insere** (Fila* f, float v);

- Incluir um novo elemento na FILA
  - Novos elementos são incluídos no Fim da FILA

# Operações Primitivas TAD FILA

float **remove** (Fila* f);

- Consultar e Remover elemento da FILA
  - O elemento a ser removido é aquele que está no início da FILA

# TAD FILA
# Implementação com Vetor

- Vetores: características

- Vetores x TAD FILA

# Vetores: características

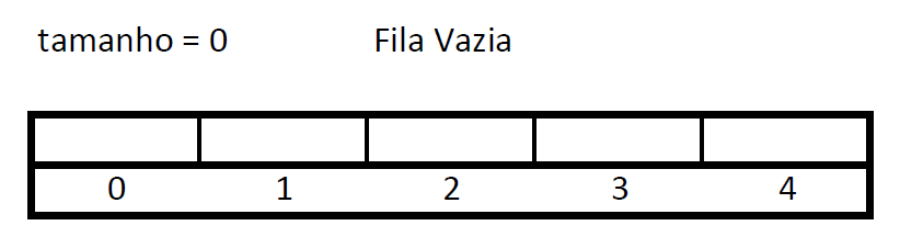
- Conjunto de elementos
- Indexados
- Tamanho predefinido
- Todos elementos do mesmo tipo

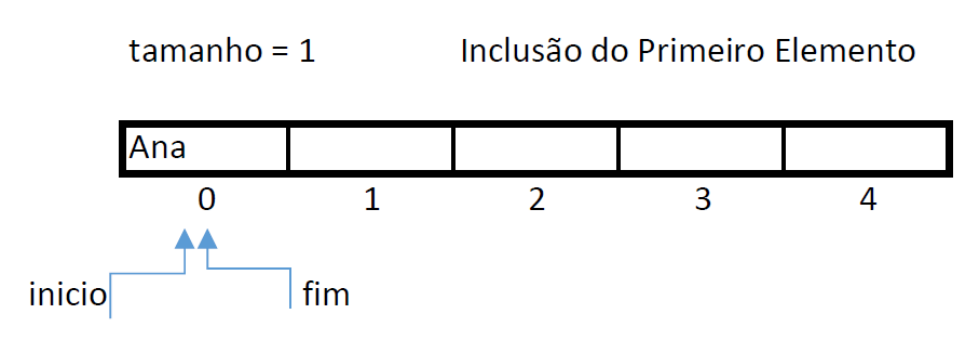
| Conteúdo | A | J | T | D | B | C | L |
|---|---|---|---|---|---|---|---|
| Índice | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

# TAD FILA

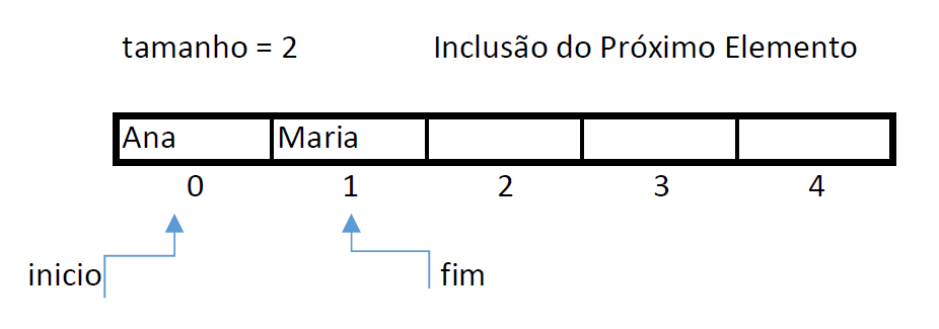

- Conjunto ordenado de elementos
- Critério de ordenação: ordem de inclusão
- Novos elementos: FIM da FILA
- Retirar elementos: INÍCIO da FILA
- Critério FIFO (First-In-First-Out)

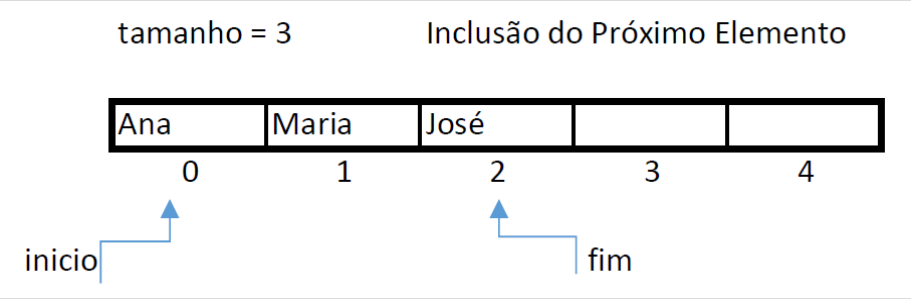
# Vetores x TAD FILA



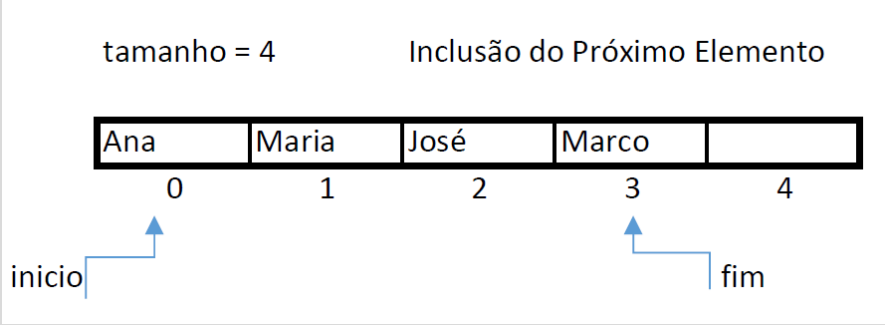



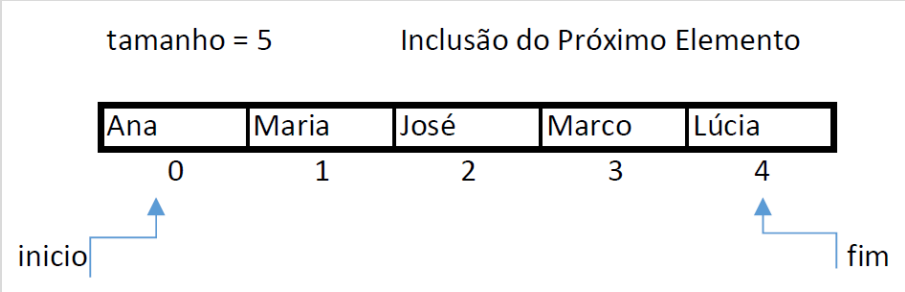
# Vetores x TAD FILA



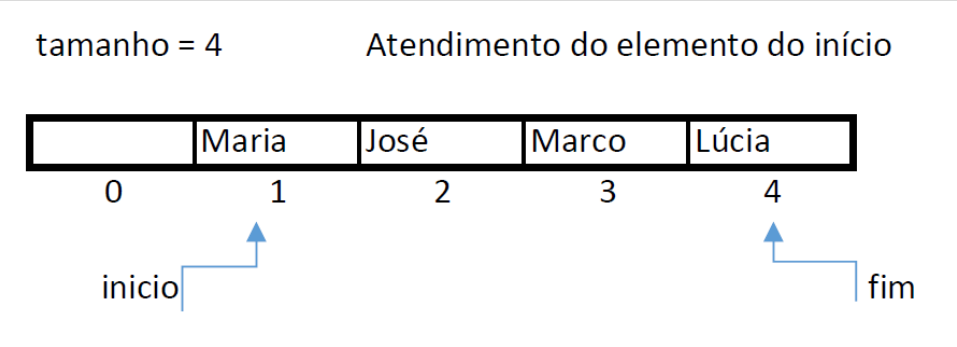



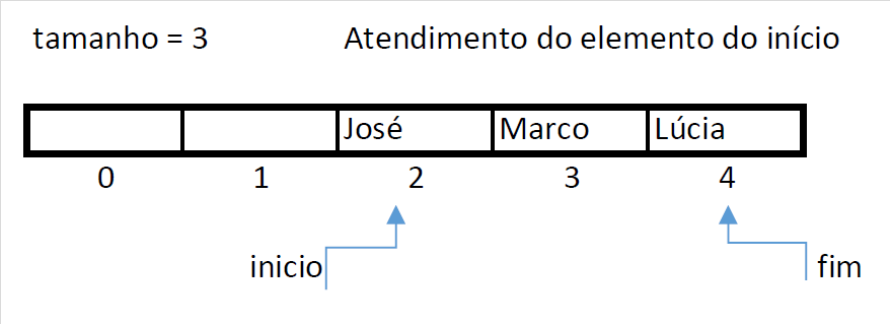
# Vetores x TAD FILA



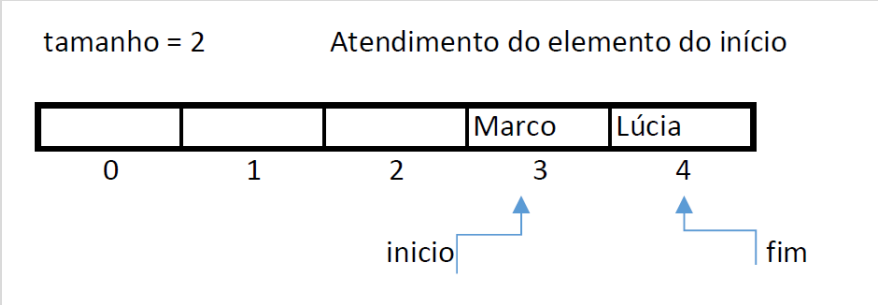



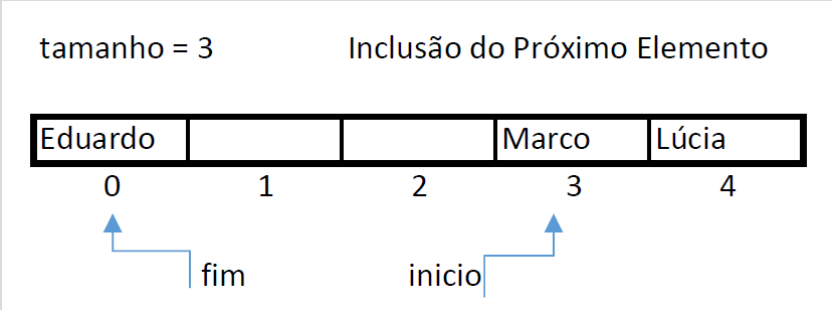
# Vetores x TAD FILA



# Vetores x TAD FILA



# Vetores x TAD FILA

- Vetores não são flexíveis
- Vetores tem tamanho predefinido
- Início e Fim da FILA não coincidem com o início e fim do vetor
- Complexidade para gerenciar esses limites

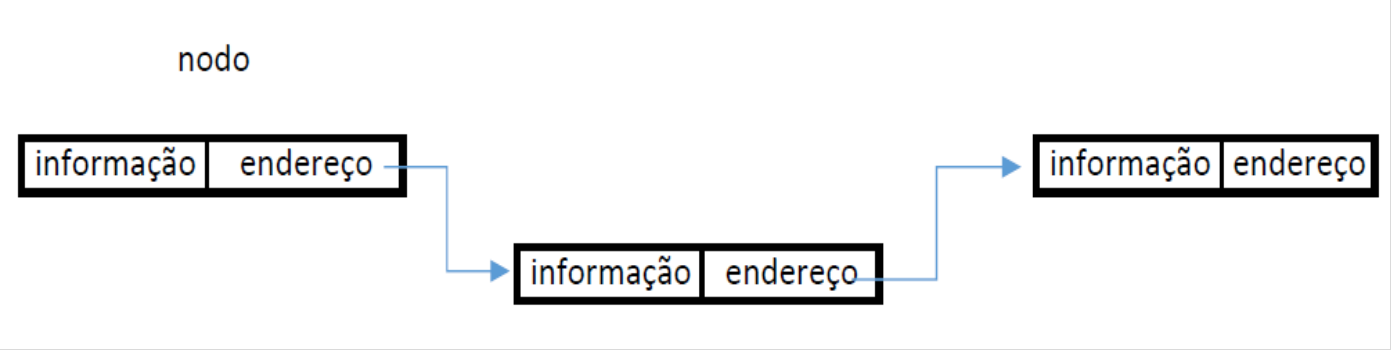
# TAD FILA
# Implementação com Encadeamento

- Implementação com Encadeamento
- Estrutura de Dados TAD FILA
- Operações Primitivas TAD FILA

# TAD FILA - Encadeamento

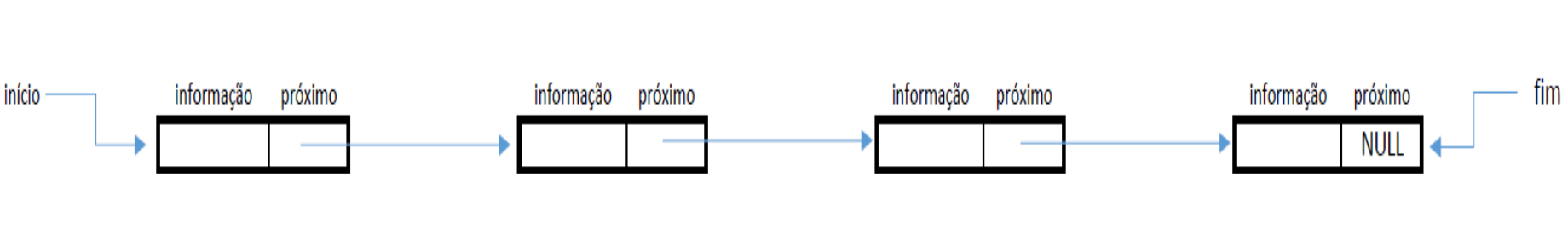

- Encadeamento
  - Alocação Dinâmica de Memória
  - Ligação entre os elementos

# Encadeamento




```
typedef struct nodo{
   float  informacao;
   nodo* endereco;
};
```

# TAD FILA - Encadeamento




```
typedef struct nodo{
   float  elemento;
   nodo* próximo;
};
```

# Estrutura de Dados

```
struct fila {
  int tamanho;
  nodo* inicio, fim;
};
typedef struct fila Fila;
```

```
typedef struct nodo {
   float  elemento;
   nodo* próximo;
};
```

# Operações Primitivas

```
Fila* cria (){
    Fila* f = (Fila) malloc (sizeof (Fila));
    f->tamanho = 0;
    f->inicio = NULL;
    f->fim = NULL;
    return(f);
}
```

```
struct fila {
  int tamanho;
  nodo* inicio, fim;
};
typedef struct fila Fila;
```

# Operações Primitivas

```
int vazia (Fila* f){
    if (f->inicio == NULL)
      return(1);
    else
      return(0);
}
```

```
void libera (Fila* f){
    if (f != NULL)
      free(f);
}
```

```
int tamanho (Fila* f){
    if (f != NULL)
      return(f->tamanho);
    else
      return(0);
}
```

# Operações Primitivas

```
void insere (Fila* f, float v){
  nodo* anterior;
  nodo* novo = (nodo) malloc (sizeof (nodo));
  if (novo == NULL)
     printf (“Memoria insuficiente!!”);
  else {
    novo -> elemento = v;
    novo -> proximo = NULL;
    f->tamanho = f->tamanho + 1;
```

```
 if (f->inicio == NULL){
       f->inicio = novo;
       f->fim = novo;  }
    else {
    anterior = f->fim;
    anterior->próximo = novo;
    f->fim = novo;
  }
}
```

# Operações Primitivas

```
float remove (Fila* f){
  nodo* primeiro;
  float v;
  if (vazia(f)){
     printf (“Fila Vazia!!”);
     return(0);
  }
```

```
else {
        primeiro = f->inicio;
        v = primeiro->elemento;
        f->inicio = primeiro->proximo;
        f->tamanho = f->tamanho – 1;
        if (f->inicio == NULL)
            f->fim = NULL;
        free(primeiro);
        return(v);
   }
}
```

# Atividades

- Ver no moodle!