Estrutura de Dados I

Listas, Filas e Pilhas

Profa.: Márcia Sampaio Lima

EST - UEA

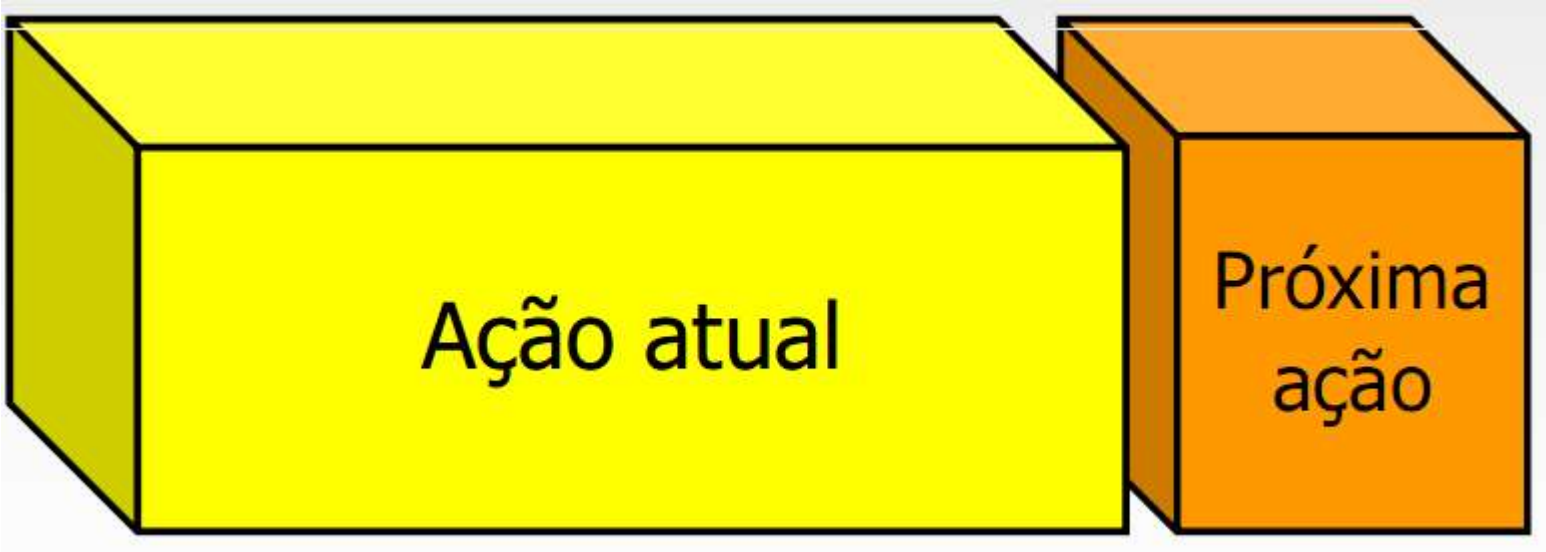
Listas Encadeadas

- Listas Encadeadas ou Listas Ligadas representam uma seqüência de objetos na memória do computador.

- Exemplo: Lista de atividades do dia...
  - Comprar uma lâmpada
  - Trocar uma lâmpada queimada
  - Procurar uma conta na gaveta
  - Pagar uma conta
  - Desligar computador
  - Dormir

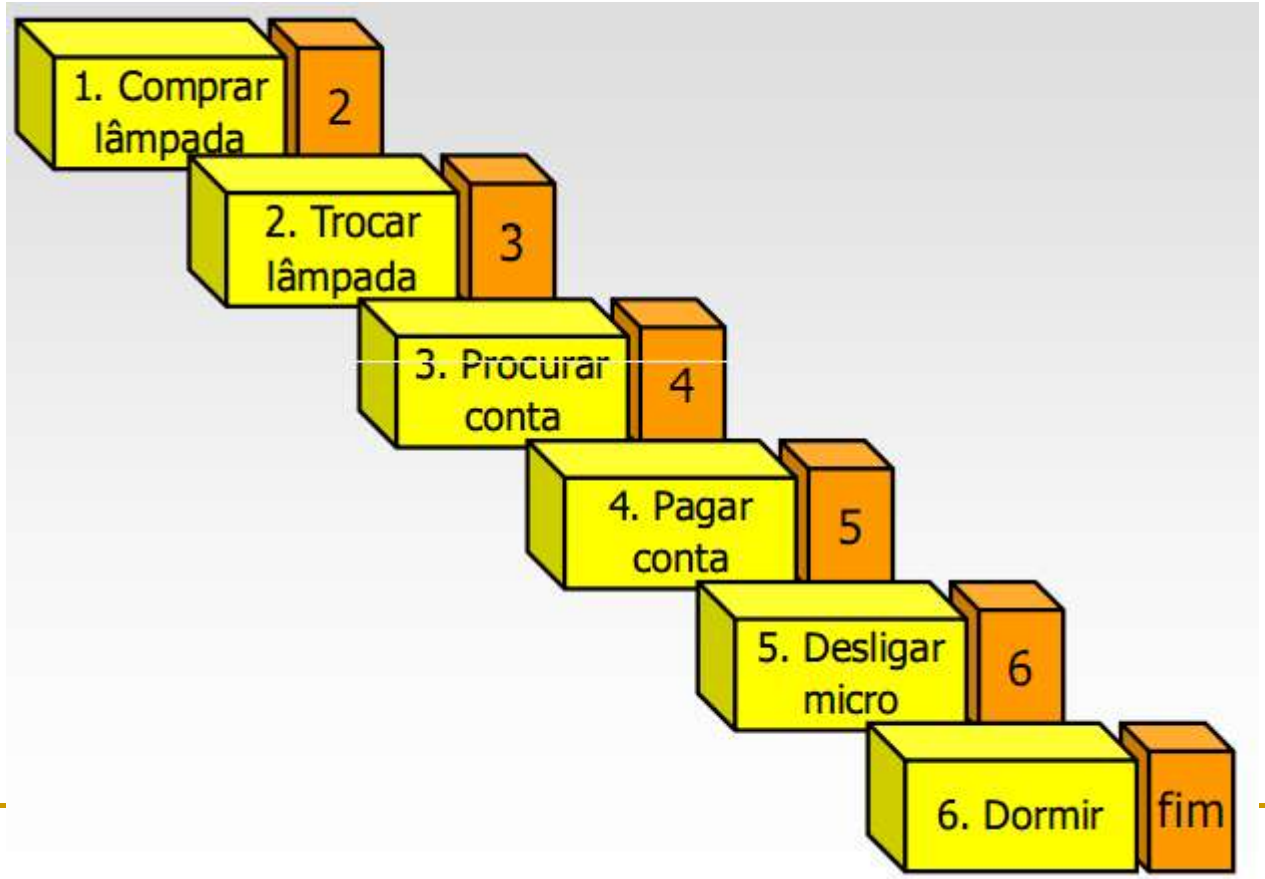
Listas Encadeadas

- Na lista de atividades anterior, uma tarefa dependia da execução da tarefa anterior.



Listas Encadeadas

- Lista Encadeada



Listas Lineares

- Exemplo de Listas:
  - Lista telefônica
  - Lista de presença
  - Lista de pacotes a serem transmitidos
  - Lista de clientes de uma agência bancária
  - Etc

Listas Lineares

- São estruturas formadas por um conjunto de dados de **forma a preservar a relação de ordem linear entre eles.**

- Uma das formas mais simples de interligar os elementos de um conjunto.

- Podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda.

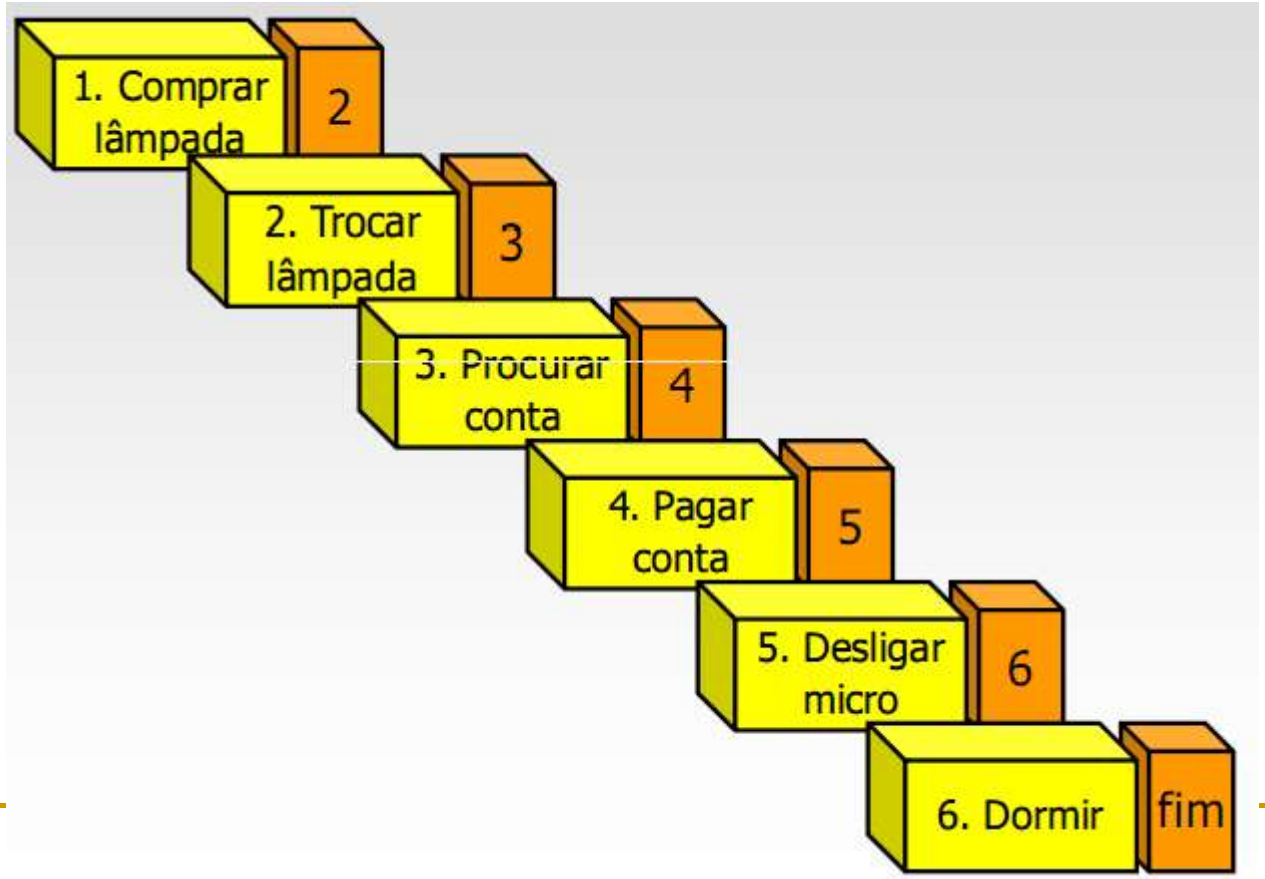
Listas Lineares

- Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.

- Uma lista é composta por nodos, os quais podem conter, cada um deles, um dado primitivo ou composto.

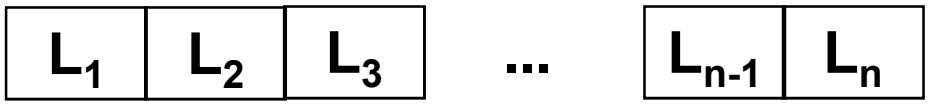
Listas Encadeadas

- Lista Encadeada



Listas Lineares

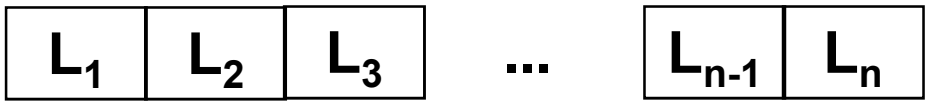
- Representação



- Sendo:
  - L_1 → primeiro elemento da lista
  - L_2 → sucessor de L_1
  - L_{n-1} → antecessor de L_n
  - L_n → último elemento da lista

Listas Lineares

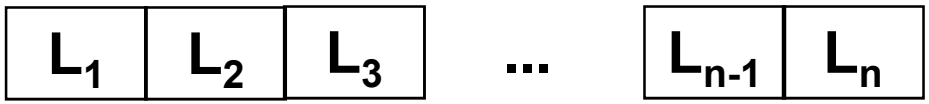
- Representação



  - Seqüência de zero ou mais itens: L_1 , ..., L_{n-1} , L_n
  - Na qual L_i é de um determinado tipo e n representa o tamanho da lista linear.

Listas Lineares

- Representação



  - Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão.
    - Assumindo n ≥ 1, L_1 o primeiro item da lista e *Ln* o último item da lista, temos que:
      - L_i precede L_{i+1} para i = 1, 2, · · · , n -1
      - L_i sucede L_{i-1} para i = 2, · · · , n
      - O elemento L_i é dito estar na i-ésima posição da lista.

Listas Lineares

- Operações:
  - Criar uma lista vazia
  - Verificar se a lista está vazia
  - Obter tamanho da lista
  - Inserir, retirar e localizar elementos.
  - Concatenar duas listas formando uma lista única.
  - Dividir uma lista e formar duas ou mais listas.
  - Exibir todos os elementos da lista

Listas Lineares

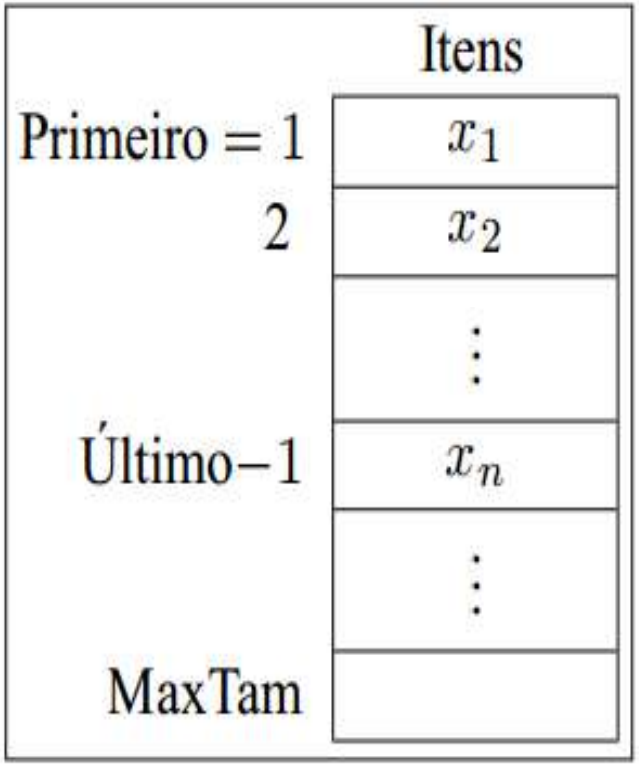
- Representação (parte 2):
  - Várias EDs podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares.
  - As duas mais utilizadas:
    - Implementações por meio de arranjos (vetores).
    - Implementações por meio de apontadores.

Listas Encadeadas

- Implementações por meio de arranjos (vetores ou matrizes).

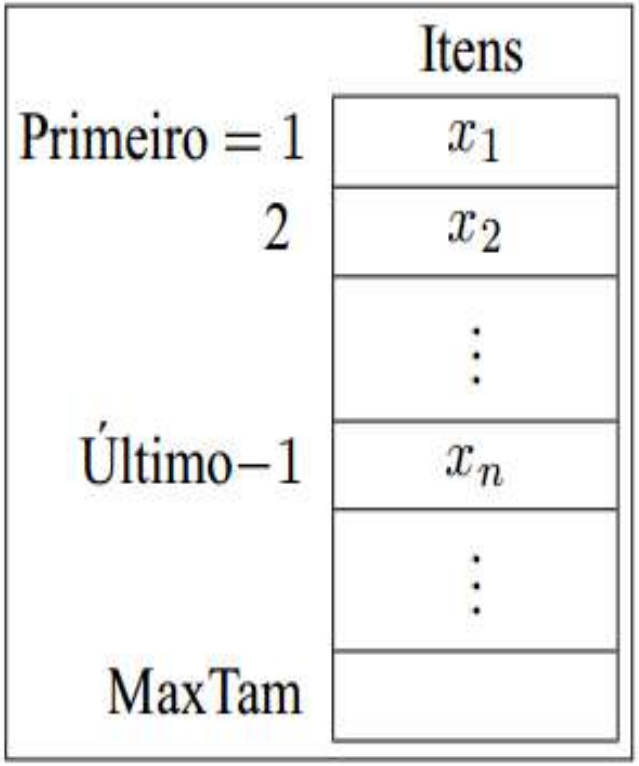
Listas Lineares

- Implementações por **ARRANJOS**
  - Os itens da lista são armazenados em posições contíguas de memória.
  - A lista pode ser percorrida em qualquer direção.
  - A inserção de um novo item pode ser realizada após o último item com custo constante.



Listas Lineares

- Implementações por **ARRANJOS**
  - A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
  - Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.



Lista Encadeada

- Inserir itens no meio da lista:

Listas Lineares

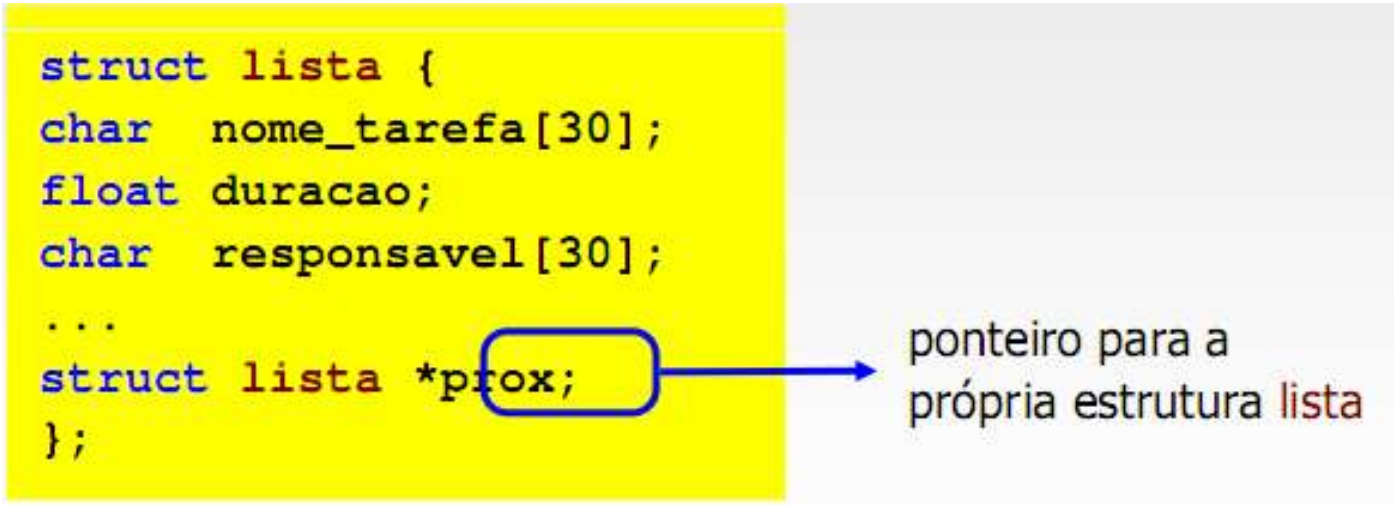
- Implementações por **ARRANJOS**
  - Características:
    - Itens são armazenados em posições contíguas de memória.
    - A lista pode ser percorrida em qualquer direção.
    - A inserção de um novo item pode ser realizada após o último item com custo constante.
    - A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
    - Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.

Listas Lineares

- Implementações por **ARRANJOS**
  - Vantagem:
    - Economia de memória (os apontadores são implícitos nesta estrutura).
  - Desvantagens:
    - Inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso.

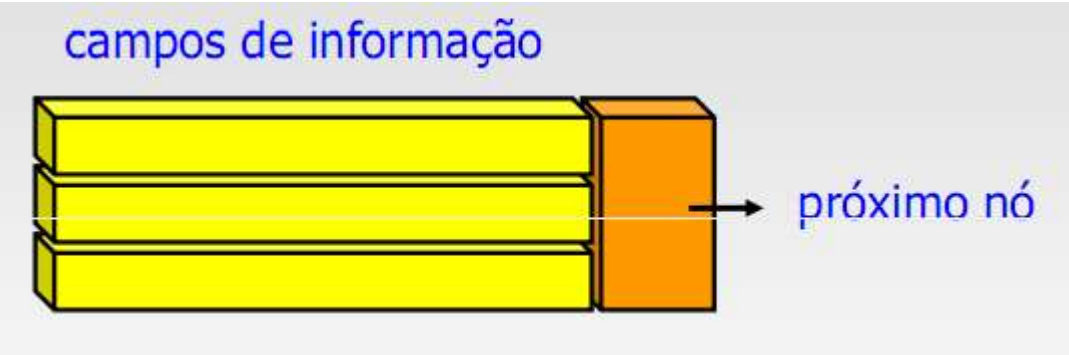
Listas Encadeadas

- Representação por **PONTEIROS**
  - **Estrutura de dados dinâmicas:** estrutura de dados que contém ponteiros para si próprias.



Listas Encadeadas

- Representação gráfica de um elemento da lista:



Listas Lineares

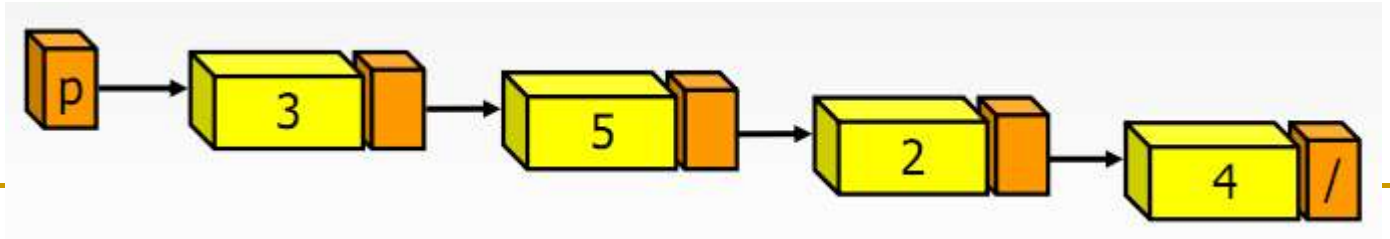
- Implementação por **APONTADORES**
  - Cada item é encadeado com o seguinte mediante uma variável do tipo Apontador (ponteiro).
  - Permite utilizar posições não contíguas de memória.
  - É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.

Listas Lineares

- Implementação por **APONTADORES**
  - A lista é constituída de células.
  - Cada célula contém um item da lista e um apontador para a célula seguinte.
  - P é um apontador para a célula cabeça da lista.

Listas Encadeadas

- Cada item da lista pode ser chamado:
  - Elemento
  - Nó
  - Célula
  - Item
- O apontador para o início da lista é tratado como se fosse uma célula (cabeça).
- O símbolo / representa ponteiro nulo (NULL), indica final da lista.



Listas Lineares

- Implementação por **APONTADORES**

- Vantagens:
  - Permite inserir ou retirar itens do meio da lista a um custo constante (importante quando a lista tem de ser mantida em ordem).
  - Bom para aplicações em que não existe previsão sobre o crescimento da lista (o tamanho máximo da lista não precisa ser definido a priori).
- Desvantagem:
  - Utilização de memória extra para armazenar os apontadores.

Listas Encadeadas

- Operações sobre listas encadeadas:
  - Criar a lista
  - Inserir itens
  - Remover itens
  - Buscar itens

  - Para manter a lista ordenada, após realizar algumas dessas operações, será necessário apenas movimentar alguns ponteiros.

Listas Encadeadas

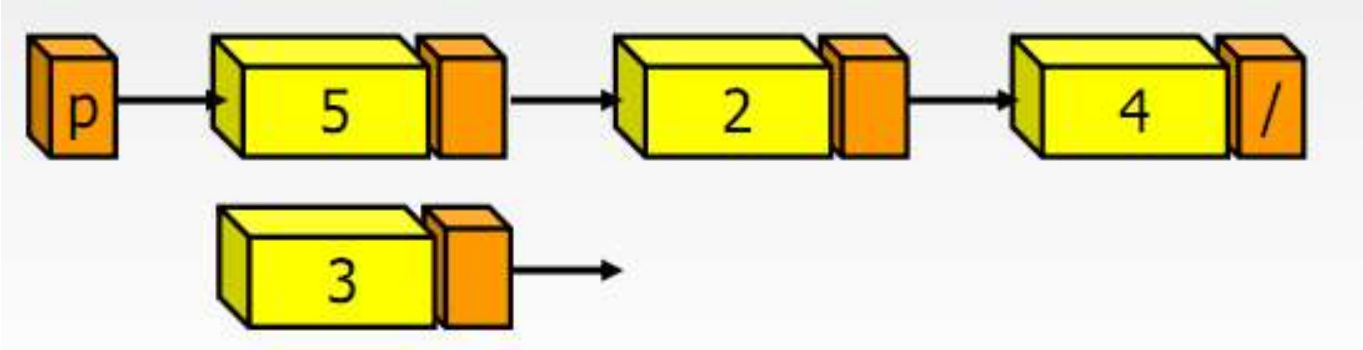
- Outras operações:
  - Destruir uma lista
  - Ordenar uma lista
  - Intercalar duas listas
  - Concatenar duas listas
  - Dividir uma lista em duas
  - Copiar uma lista em outra

Listas Lineares

- Inserir elemento na Lista:
  - Inserção início
  - Inserção meio
  - Inserção fim

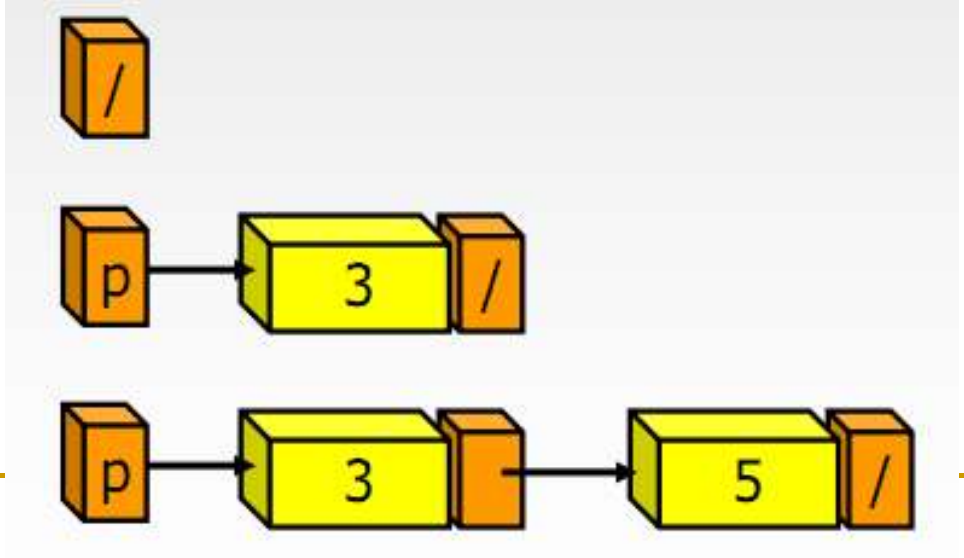
Listas Encadeadas

- Inserção de itens no início:
  - O endereço armazenado pelo ponteiro p deve ser alterado para o endereço do item a ser acrescido à lista.



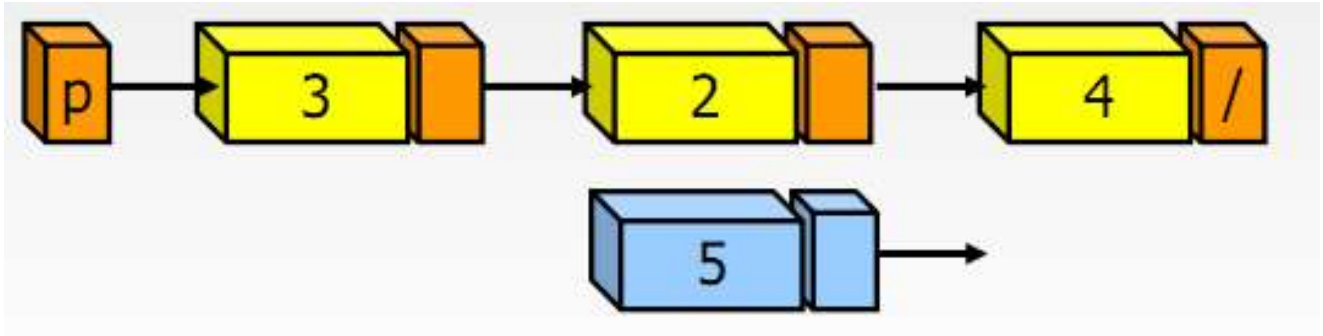
Listas Encadeadas

- Inserção de itens no final:
  - O endereço armazenado pelo ponteiro p será alterado caso a lista esteja vazia ou
  - O campo prox do último item será alterado.



Listas Encadeadas

- Inserção de itens no meio:
  - O campo prox do item a ser inserido recebe o campo prox do item posterior.
  - O campo prox do item antecessor recebe o endereço do item a ser inserido.

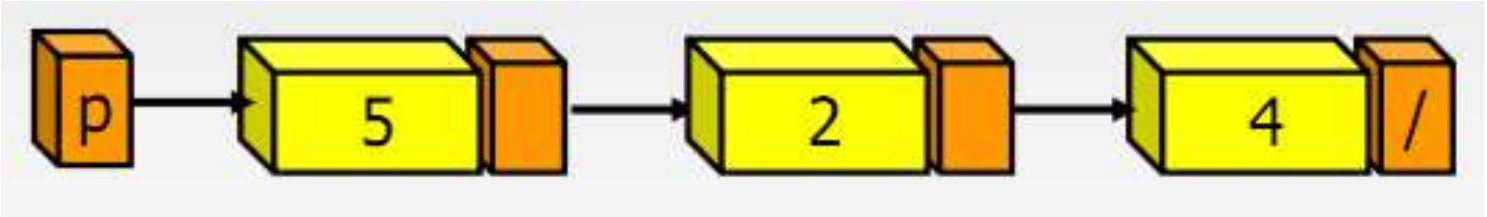


Listas Lineares

- Excluir elemento
  - Exclusão início
  - Exclusão meio
  - Exclusão fim

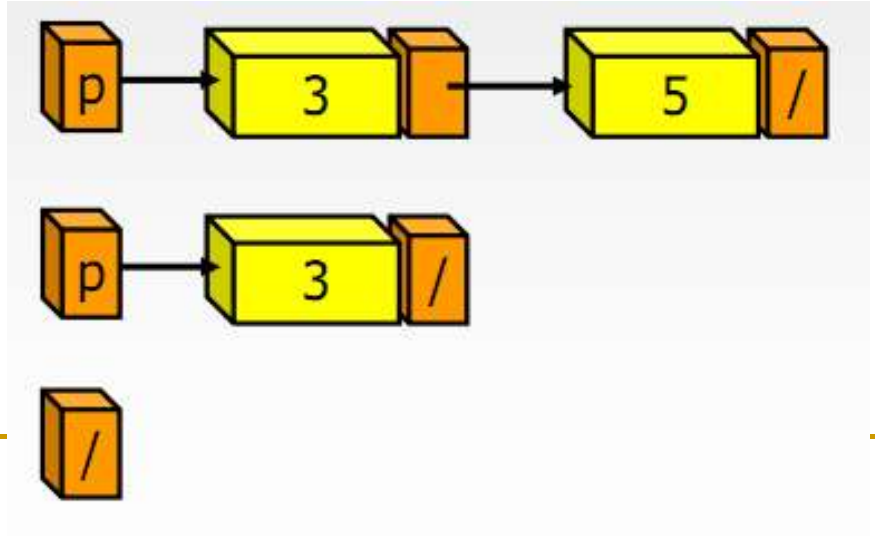
Listas Encadeadas

- Remoção de itens no início:
  - O endereço armazenado pelo ponteiro p deve ser alterado para o endereço do item que segue o primeiro item da lista.



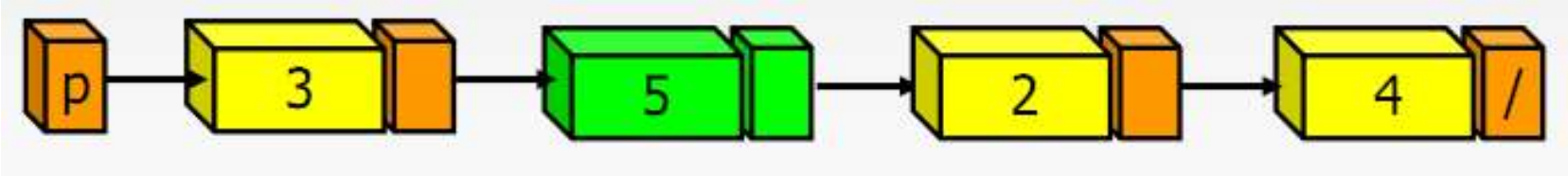
Listas Encadeadas

- Remoção de itens no final:
  - O campo prox do último item será alterado caso a lista contenha mais de um item ou
  - O endereço armazenado em p será alterado para NULL.



Listas Encadeadas

- Remoção de itens no meio:
  - Item antecessor recebe o campo prox do item a ser removido.

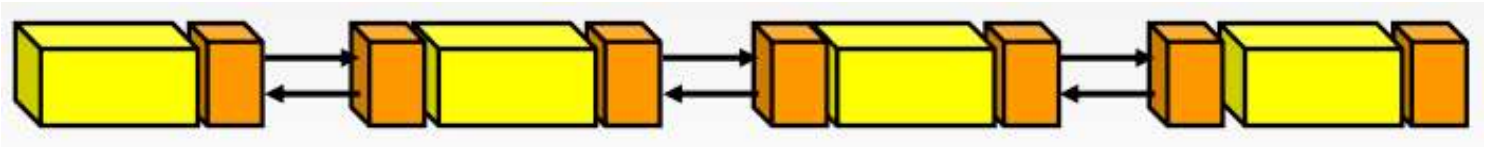


Listas Encadeadas

- Variações da Lista Encadeada:
  - Lista Duplamente Encadeada
  - Listas Circulares

Listas Encadeadas

- Lista Duplamente Encadeada
  - Cada elemento da lista é ligado ao seu sucessor e ao seu predecessor.
  - Possibilita um trajeto em ambos os sentidos, simplificando o gerenciamento da lista.



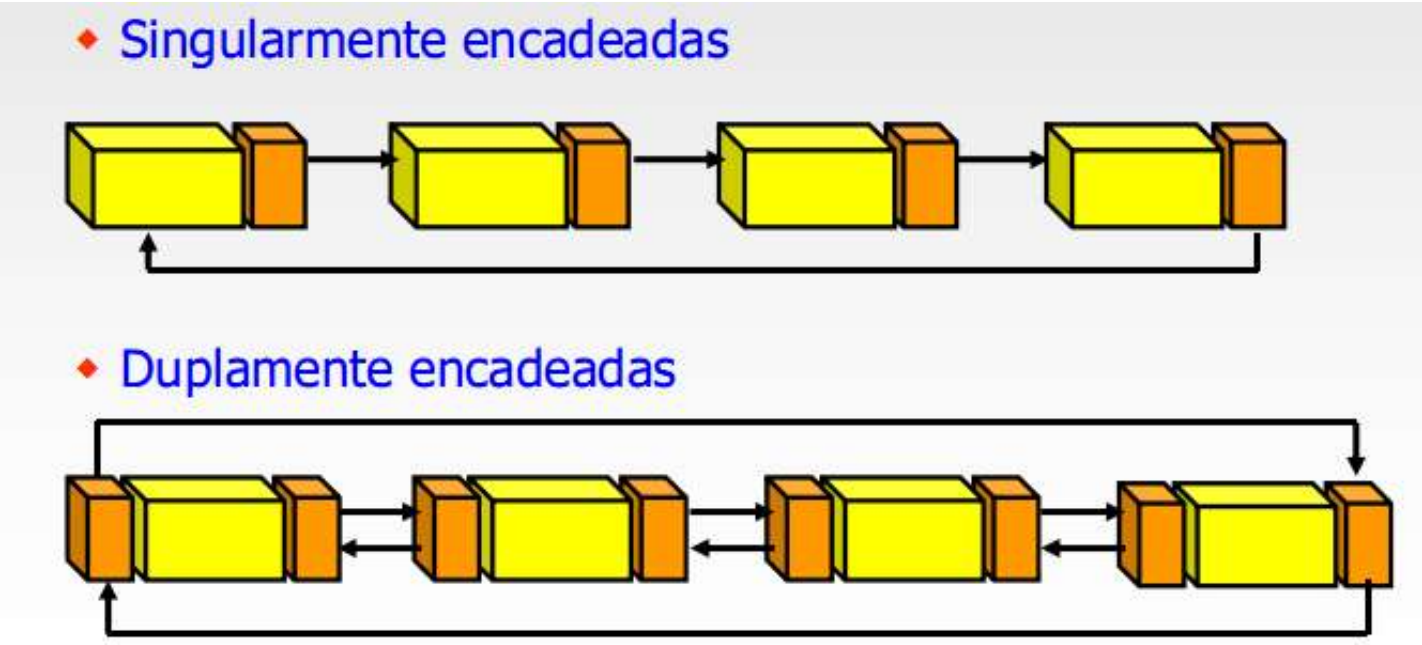
Listas Encadeadas

- Lista Duplamente Encadeada
  - A estrutura de dados de um nó de uma lista duplamente encadeada recebe um novo campo: um ponteiro para o nó antecessor.

```
typedef struct noh {
   int          info;
   struct noh   *ant;
   struct noh   *prox;
} tipoNode;
```

Listas Encadeadas

- Listas Circulares
  - São listas encadeadas cujo fim aponta para o seu início, formando um circulo que permite uma trajetória contínua na lista. Podem ser:



Listas Encadeadas

- Listas Circulares
  - Estrutura do nó permanece a mesma.Dependerá apenas se o encadeamento da lista é duplo ou singular.
  - Não há necessidade de dois ponteiros: um para o início e outro para o fim da lista.
  - Basta marcar um nó da lista para evitar loops.

Pilhas e Filas

Pilhas e Filas

- São **casos especiais** de **Listas Encadeadas**.
- Possuem **regras rigorosas** para acessar seus dados.
- A **operação de recuperação** de dados são **destrutivas**:
  - Para acessar dados intermediários dentro dessas estruturas é necessário destruir seqüencialmente dados anteriores.

Pilhas

- É uma das Eds mais simples.
- É uma lista linear em que todas as inserções, retiradas e acessos são feitos em apenas um extremo da lista: topo.

Pilhas

- Exemplo:
  - Pilha de pratos.
  - Pilha de livros.

- A retirada ou adição de pratos/livros deve ser feita na parte superior == desempilhar itens da pilha.

Pilhas

- Os itens são colocados um sobre o outro.
- O item inserido mais recentemente está no topo.
- O item inserido menos recentemente no fundo.
- **LIFO (last-in-first-out)** → o último item inserido é o primeiro item que pode ser retirado da lista.

Pilhas

- Os elementos da pilha são retirados na ordem inversa à ordem em que foram introduzidos: o primeiro que sai é o último que entrou → LIFO.

Pilhas

- Existe uma ordem linear para pilhas, do “mais recente para o menos recente”.

- Uma pilha contém uma seqüência de obrigações adiadas. A ordem de remoção garante que as estruturas mais internas serão processadas antes das mais externas.

Pilhas

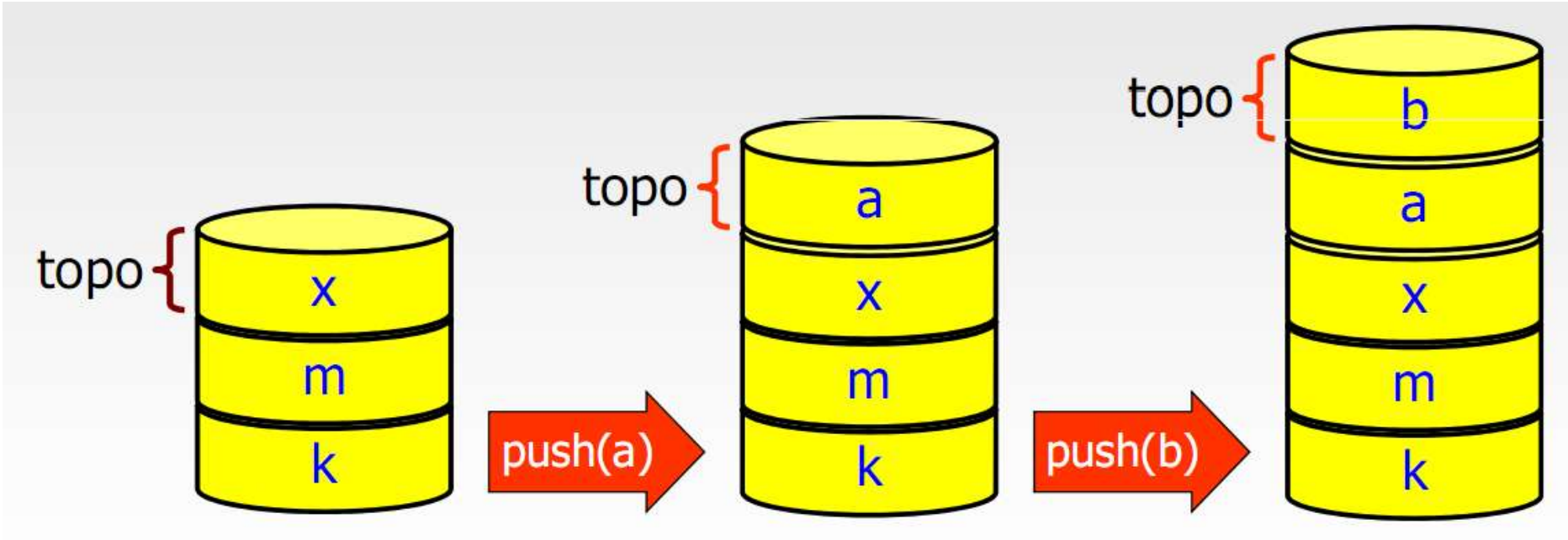
- Operações básicas:
  - **Empilha(x, Pilha).** Insere o item x no topo da pilha.
    - **PUSH(x, Pilha)**
  - **Desempilha(Pilha, x).** Retorna o item x no topo da pilha, retirando-o da pilha.
    - **POP(x, Pilha)**

Pilhas

- Outras Operações:
  - Faz a pilha ficar vazia: FPVazia(Pilha).
  - Vazia(Pilha): Retorna true se a pilha está vazia; caso contrário, retorna false.
  - Tamanho(Pilha). Esta função retorna o número de itens da pilha.

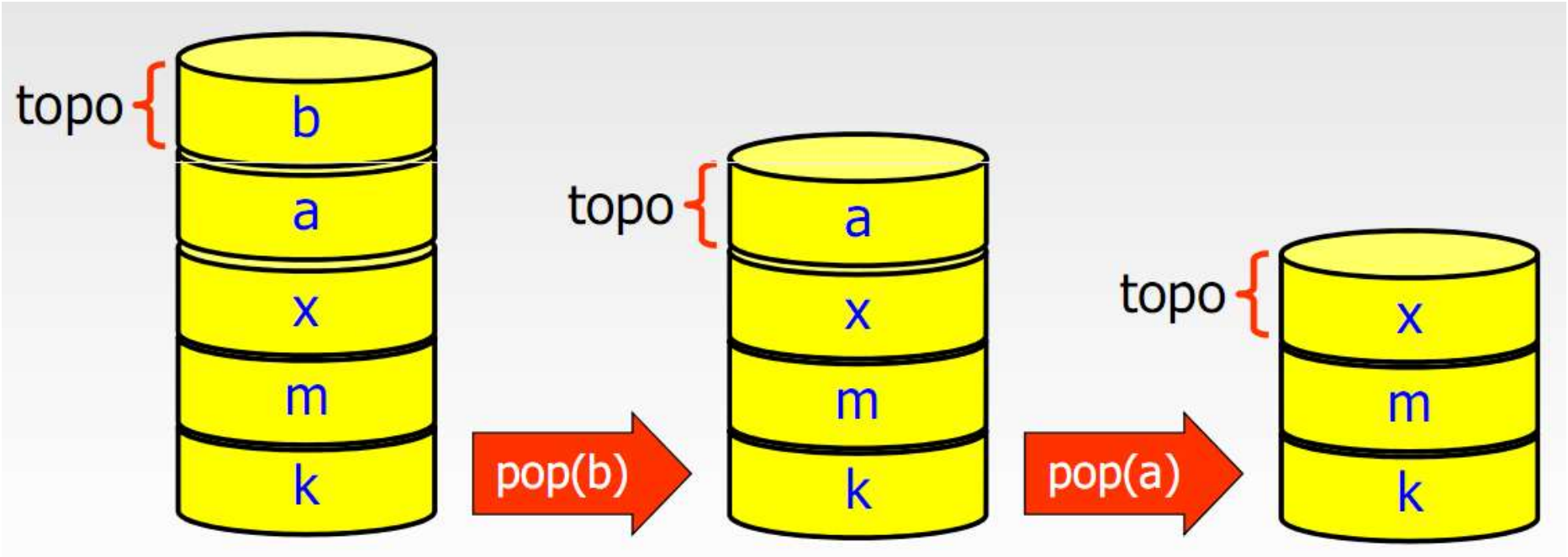
Pilhas

- Push()



Pilhas

- Pop()

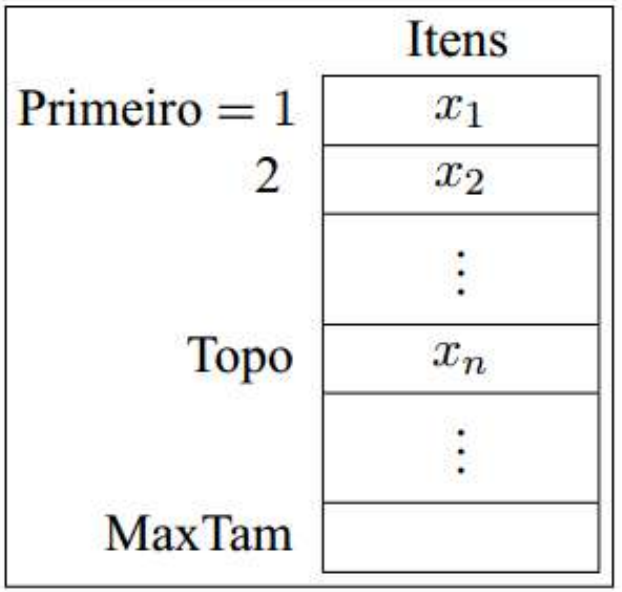


Pilhas

- Representação:
  - As duas mais utilizadas:
    - Implementações por meio de arranjos.
    - Implementações por meio de apontadores.

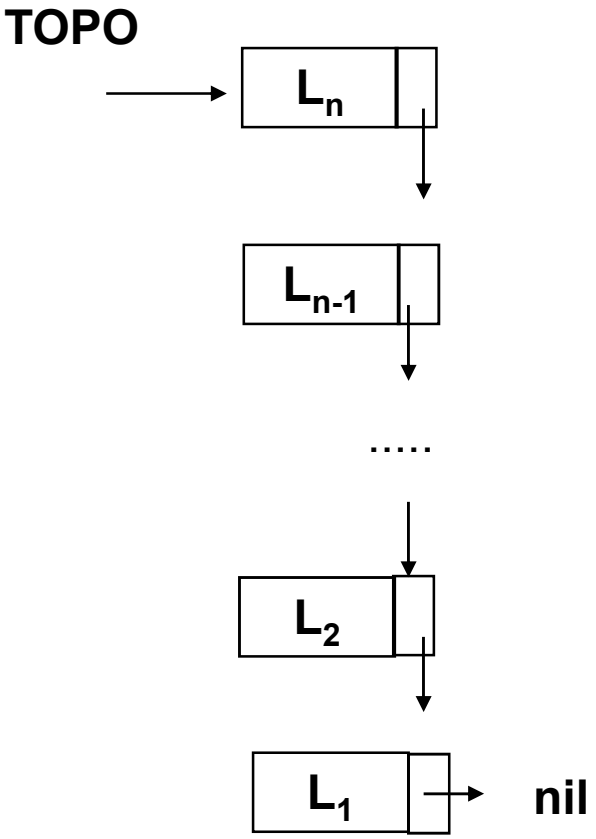
Pilhas

- Representação por **ARRANJOS**
  - Os itens da pilha são armazenados em posições contíguas de memória.
  - Como as inserções e as retiradas ocorrem no topo da pilha, um cursor chamado Topo é utilizado para controlar a posição do item no topo da pilha.



Pilhas

- Representação por **APONTADORES**
  - Para desempilhar o item *Ln* basta fazer o topo apontar para o proximo elemento da pilha.
  - Para empilhar um novo item, basta criar uma nova célula e ajustar os apontadores: topo e apontador da nova celula.



Pilhas

- Implementação de pilha usando estruturas:

```
typedef struct noh {
    float       info;
    struct no   *prox;
} tipoNo;

struct pilha {
tipoNo* topo;
} tipoPilha;
```

Pilhas

- Criar uma estrutura de pilha

```
tipoPilha* criar(void)
{
tipoPilha *p;
p = (tipoPilha*) malloc(sizeof(tipoPilha));
p->topo = NULL;
return p;
}
```

Pilhas

- Insere elemento no topo – push()

```
tipoPilha push(tipoPilha *p, float v)
{
tipoNo* aux;
aux = (tipoNo) malloc(sizeof(tipoNo));
aux -> info = v;
aux -> prox = p->topo;
return aux;
}
```

Pilhas – POP()

```
void pop(tipoPilha *p)
{
float v;
tipoNo* aux;
if (vazia(p))
{
   printf("Pilha vazia.");
   exit(1); /*aborta o programa*/
}
v     = p->topo->info;
aux   = p->prox;
free(p);
printf("Retirou o elemento %d", v);
}
```

Pilhas

- Verifica se pilha esta vazia

```
int vazia(tipoPilha *p)
{
return (p->topo == NULL);
}
```

Filas

Fila

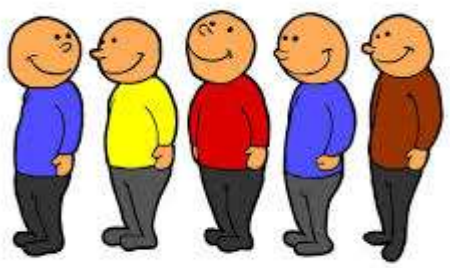
- Conceito
  - É uma lista linear em que todas as inserções são realizadas em um extremo da lista, e todas as retiradas são realizados no outro extremo da lista.
  - Exemplo:
    - Fila de banco
    - Fila de carros

Fila

- Conceito
  - O modelo intuitivo:**FIFO**
    - Pessoas no início da fila são servidas primeiro e as pessoas que chegam entram no fim da fila.
    - First-in-First-out
    - Primeiro a chegar será o primeiro a sair
  - São utilizadas quando desejamos processar itens de acordo com a ordem “primeiro-que-chega, primeiro-atendido”.
  - SO utilizam filas para regular a ordem na qual tarefas devem receber processamento e recursos devem ser alocados a processos.

Filas

- A consulta na Fila é feita desenfileirando elemento a elemento até encontrar o elemento desejado ou chegar no final da fila.



Filas

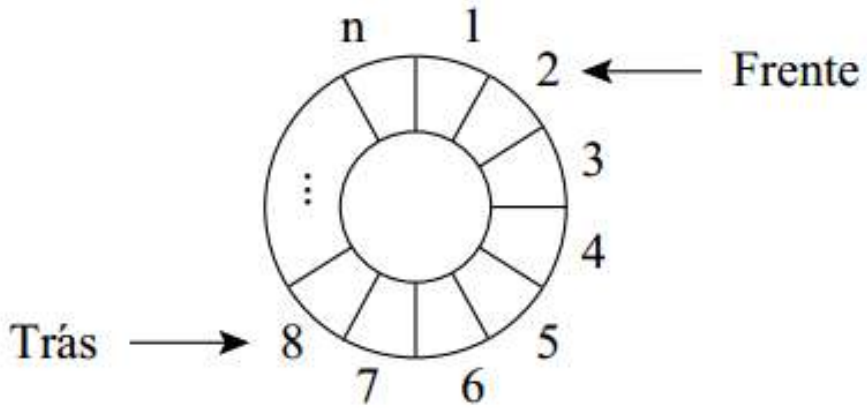
- Aplicações:
  - Fila de documentos para impressão (spooler de impressão).
  - Atendimento de processos requisitados ao SO.
  - Ordenação do encaminhamento de pacotes em um roteador.
  - Buffer para gravação de dados em mídia.

Filas

- Operações:
  - Cria a fila vazia: FFVazia(Fila).
  - Enfileira(x, Fila). Insere o item x no final da fila.
  - Desenfileira(Fila, x). Retorna o item x no início da fila, retirando-o da fila.
  - Vazia(Fila). Esta função retorna true se a fila está vazia; senão retorna false.

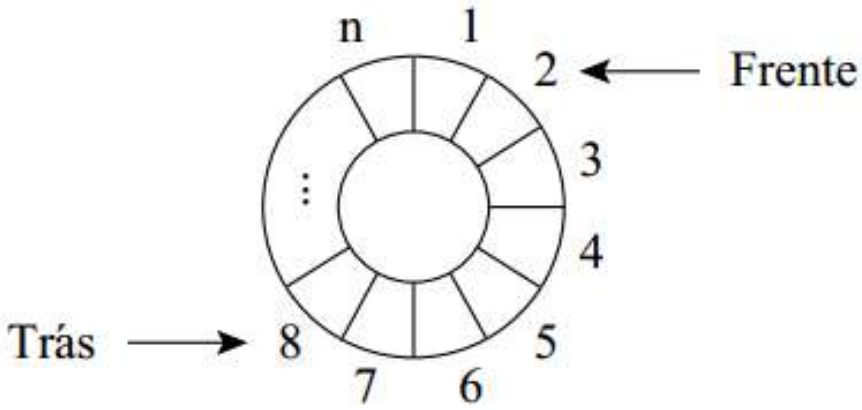
Filas

- Implementação por **ARRANJO**
  - Os itens são armazenados em posições contíguas de memória.
  - Enfileirar expande a parte de trás da fila.
  - Desenfileirar contrai a frente da fila.



Filas

- Implementação por **ARRANJO**
  - Com poucas inserções e retiradas, a fila vai ao encontro do limite do espaço da memória alocado para ela.
  - Representação: imaginar o array como um círculo. A primeira posição segue a última.



Filas

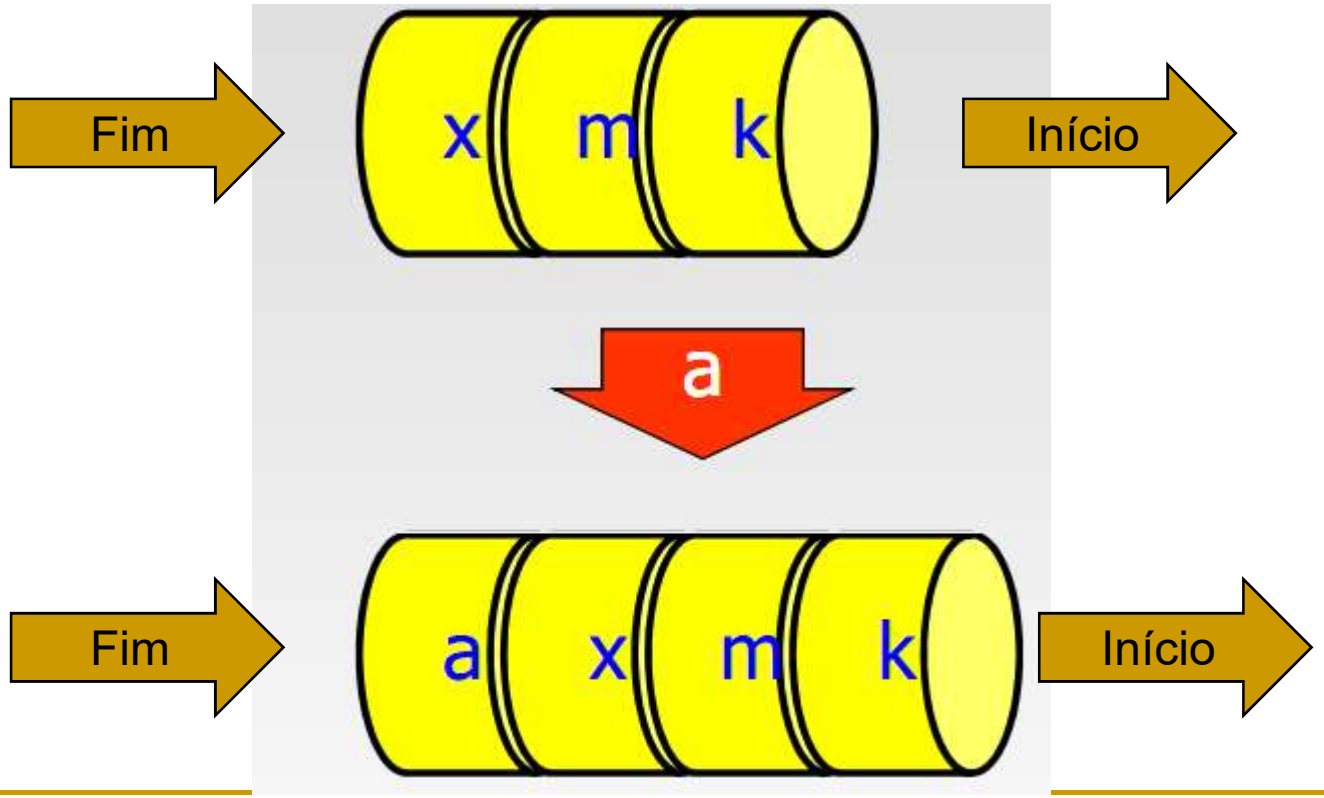
- Implementação por **APONTADOR**
  - Existem dois apontadores para controle de estrutura: frente e Trás.
  - Quando a fila está vazia, os apontadores Frente e Trás apontam para a nil.

Filas

- Implementação por **APONTADOR**
  - Para enfileirar um novo item, basta criar uma célula nova, ligá-la após a ultima celula da fila e atualizar o apontador Trás.
  - Para desenfileirar o primeiro item basta desligar a célula cabeça da lista e atualizar o apontador Frente.

Filas

- Inserção de elementos na Fila



Filas

- Remoção de elementos da Fila

Filas

- Definição Tipo Fila

```
typedef struct {
   int info;
   /* outros componentes*/
} tipoItem;

typedef struct no {
   tipoItem  item;
   struct no *prox;
} tipoNo;

typedef struct {
   tipoNoh frente;
tipoNoh tras;
} tipoFila;
```

```
void ffvazia(tipoFila *fila) {
   fila->frente = (tipoNo) malloc(sizeof(tipoNo));
   fila->tras   = fila->frente;
   fila->frente->prox = NULL;
}

int vazia(tipoFila fila) {
    return (fila.frente == fila.tras);
}

void enfileira(tipoItem x, tipoFila *fila) {
     fila->tras->prox = (tipoNo) malloc(sizeof(tipoNo));
     fila->tras       = fila->tras->prox;
     fila->tras->item = x;
     fila->tras->prox = NULL;
}
```

```
void desenfileira(tipoFila *fila) {
   ptr p;
   if (vazia(*fila))
{
      printf("Erro: Fila vazia.\n");
      return;
}
   p           = fila->frente;
   fila->frente = fila->frente->prox;
   free(p);
   printf("Item da fila excluido com sucesso.");
   getch();
}
```