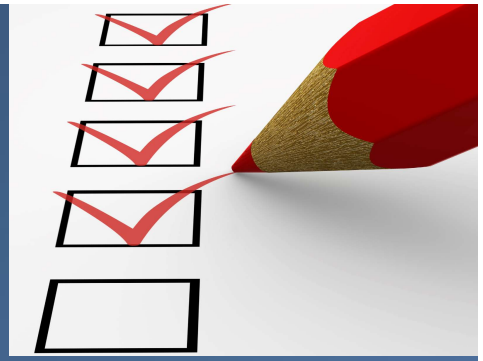
Listas Lineares



Definições:

- Lista Linear: sequência de zero ou mais elementos $a_1, a_2, \ldots, a_n$ sendo
  - a_i elementos de um mesmo tipo
  - *n* o tamanho da lista linear ($n = 0 \Rightarrow$ lista vazia)

- Propriedade fundamental: os elementos têm relações de ordem na lista
  - a_i precede a_{i+1} (e a_i sucede a_{i-1})
  - a_1 é o primeiro elemento da lista
  - a_n é o último elemento da lista

- Representação de Listas Lineares: TAD com elementos organizados de maneira sequencial.
  - Os tipos mais comuns de listas lineares são as *pilhas* e as *filas*

Operações:

- Uma lista linear define uma série de operações sobre seus elementos:
- Criação: Iniciar a lista como sendo vazia
- Inserção: Inserir um elemento numa dada lista
- Remoção: Remover um elemento de uma dada lista
- Consulta: Obter informações sobre um dado elemento de uma lista

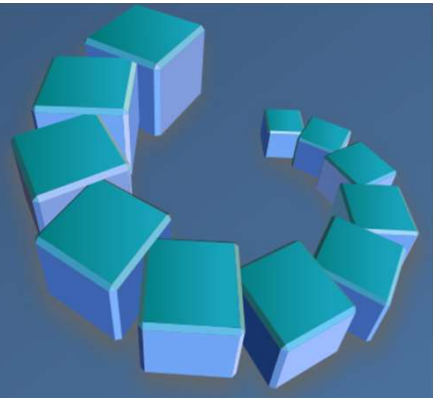
Implementação:

- A implementação de uma Lista, usando uma linguagem de programação como o C, pode ser feita por meio de:

1. Vetores (Alocação Sequencial de Memória)

ou

2. Lista Encadeada (Alocação Dinâmica de Memória)

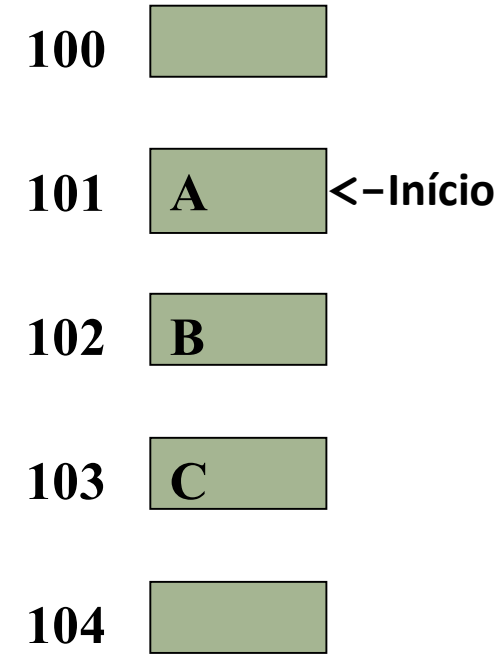


Listas Sobre Vetores

Implementação sobre Vetores:

- Quando definidas sobre vetores:
  - Elementos são alocados em sequência
  - Sequência “física”

Alocação Seqüencial: Memória



Implementação sobre Vetores:

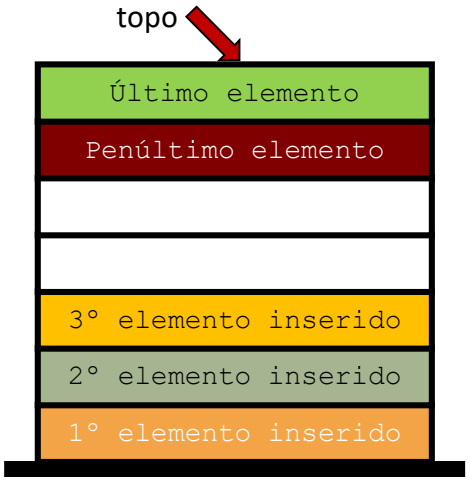
Características:

- Dados armazenados em um vetor
- A lista estará alocada em posições contíguas na memória
  - a lista sempre terá um início no índice 0 (zero) e um fim que nem sempre coincidirá com o final do vetor
  - Necessário controlar o final da lista

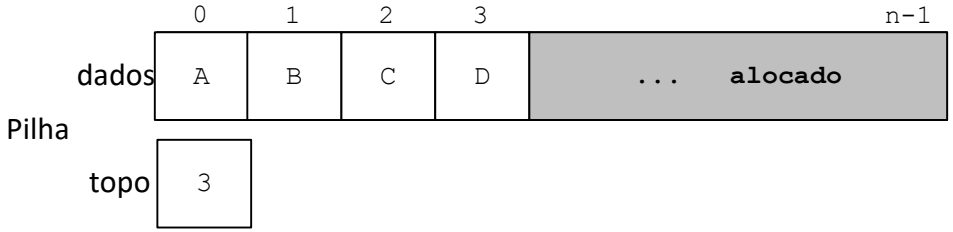
- Listas sobre vetores podem ser criadas de forma estática, ou por meio de alocação dinâmica
- Adicionar ou remover elementos no meio da lista requer deslocamentos dos demais elementos (evitar espaços vazios)
- Na sequência, estudaremos a construção sobre vetores de dois tipos muito comuns de lista: *Pilhas* e *Filas*

Pilhas: conceito

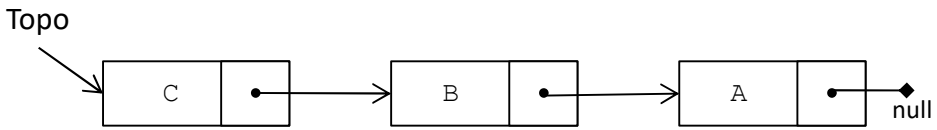
- Pilhas: conceito e representações
  - Tipo de lista com critérios rígidos de acesso: permite operações em apenas uma das extremidades (topo),
  - Respeita a política LIFO (*Last In – First Out*) de acesso



Vetor:



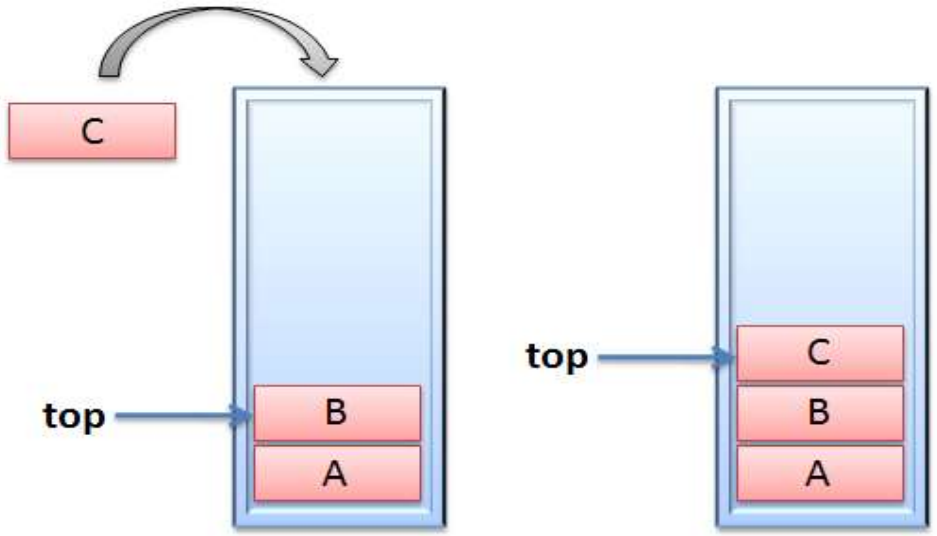
Encadeada:



Pilhas: inserção

- Pilhas: a inserção só pode ser feita no Topo (*push*)

```
Push (P, x)
    P.topo = P.topo + 1
    P.dados[P.topo] = x
```

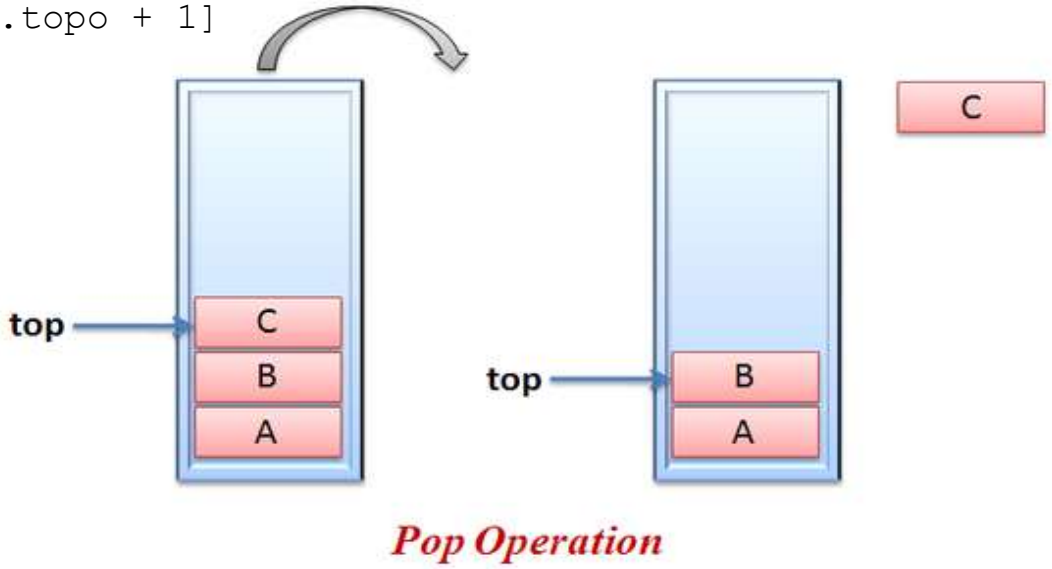


Push Operation

Pilhas: remoção

- Pilhas: a remoção só pode ser feita no Topo (*pop*)

```
Pop(P)
   if P.topo == 0
      error "pilha vazia"
   else
      P.topo = P.topo - 1
      return P.dados[P.topo + 1]
```



Pilhas: inserção e remoção

- Observação: é importante notar que as duas operações sobre Pilhas, *push* e *pop*, são operações executadas em tempo $O(1)$

```
Push (P, x)
   P.topo = P.topo + 1
   P.dados[P.topo] = x


Pop(P)
   if P.topo == 0
      error "pilha vazia"
   else
      P.topo = P.topo - 1
      return P.dados[P.topo + 1]
```

Pilhas: consulta

- Pilhas: apesar de ser possível varrer uma pilha, conceitualmente devemos ter acesso apenas ao Topo. Consulta em uma pilha seria verificar qual o elemento do topo, sem removê-lo (*peek*)

```
Peek(P)
   if P.topo == 0
      error "pilha vazia"
   else
      return P.dados[P.topo]
```

Práticas:

10. Como exercício, vamos implementar um TAD conhecido, construído sobre uma lista estática: Pilha de inteiros

    - Atenção: apesar de ser uma lista baseada em vetor, atente para a necessidade de gerenciar o tamanho do vetor. Ele deve crescer de acordo com a demanda

    - Para isso, considere que a alocação será realizada em blocos de 512 posições e as funções próprias da pilha devem verificar se mais espaço é necessário ou algum espaço pode ser desalocado (realloc)

    - Vamos estudar, brevemente, como funciona uma Pilha

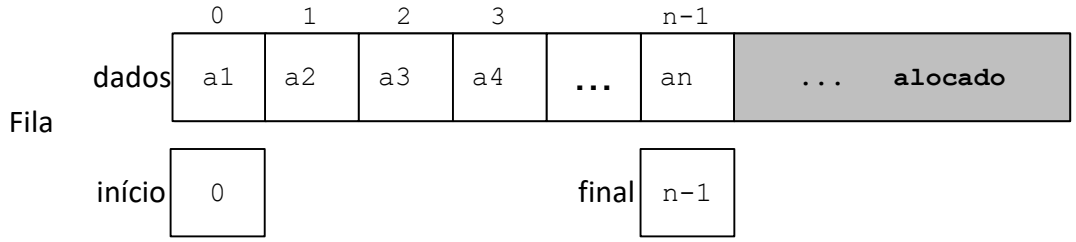
Filas: conceito

- Filas: outro tipo de lista muito utilizada
  - Tipo de lista que implementa a política FIFO (*First In – First Out*) de acesso
  - A ordem de inserção dos dados afeta a ordem de remoção

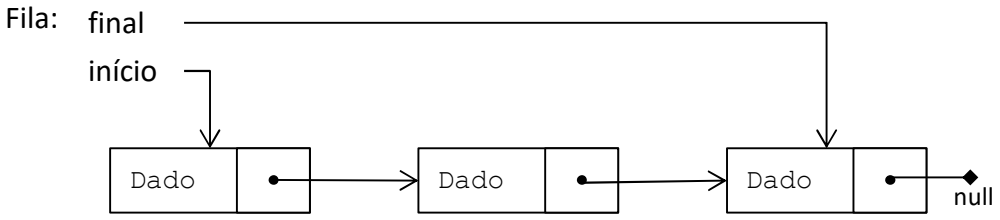
Filas: conceito

- Filas: representação
  - Organização sequencial de dados (relação de ordem)
  - Acesso apenas nas extremidades, respeitando *FIFO*

Vetor:



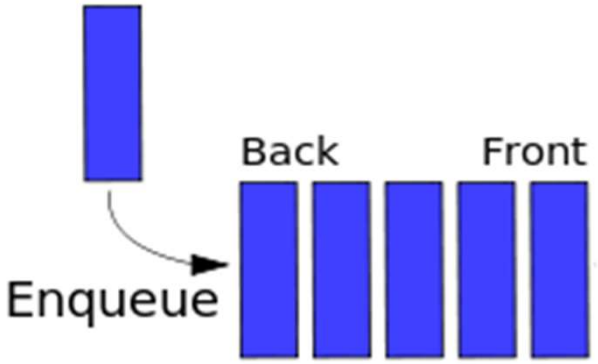
Encadeada:



Filas: inserção

- Fila: inserção só pode ser feita em uma das extremidades, chamada Fim da Fila (*enqueue*)

```
Enqueue(F, x)
  if (is_full(F))
    error "overflow"
  F.dados[F.final] = x
  F.final = (F.final + 1) % n
```

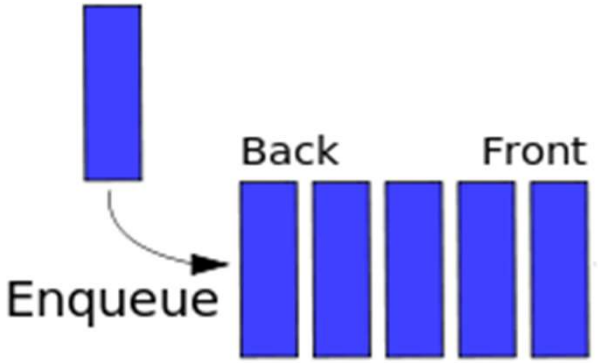


Filas: inserção

- Fila: inserção só pode ser feita em uma das extremidades, chamada Fim da Fila (*enqueue*)

```
Enqueue(F, x)
  if (is_full(F))
    error "overflow"
  F.dados[F.final] = x
  F.final = (F.final + 1) % n
```

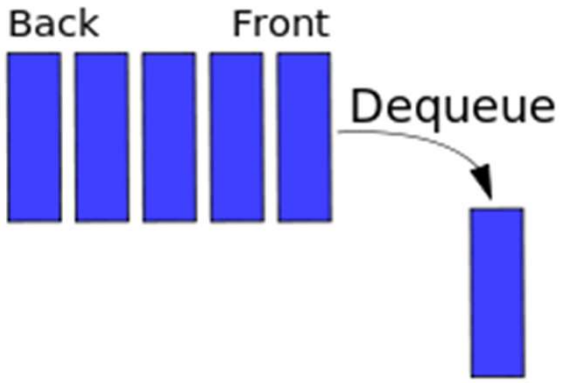
Observação: para evitar o deslocamento de dados no vetor, utilizamos aqui uma implementação em lista circular



Filas: remoção

- Fila: remoção só pode ser feita na extremidade chamada de Início da Fila (*dequeue*)

```
Dequeue(F)
  if (is_empty(F))
    error "fila vazia"
  x = F.dados[F.inicio]
  F.inicio = (F.inicio + 1) % n
  return x
```

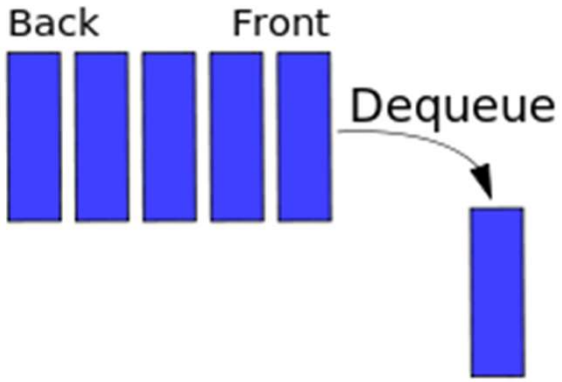


Filas: remoção

- Fila: remoção só pode ser feita na extremidade chamada de Início da Fila (*dequeue*)

```
Dequeue(F)
  if (is_empty(F))
    error "fila vazia"
  x = F.dados[F.inicio]
  F.inicio = (F.inicio + 1) % n
  return x
```

Observação: para evitar o deslocamento de dados no vetor, utilizamos aqui uma implementação em lista circular



Filas: inserção e remoção

- Observações:
  - Como inicialização, consideramos, que o tamanho do vetor é ***n*** que *inicio* = 0 e *final* = 0.
  - Com o uso de uma lista circular, notamos que as duas operações sobre Filas, *enqueue* e *dequeue*, são executadas em tempo $O(1)$
  - Esta implementação desperdiça uma posição do vetor (verificar!)

```
Enqueue(F, x)
  if (is_full(F))
    error "overflow"
  F.dados[F.final] = x
  F.final = (F.final + 1) % n
```

```
is_full(F)
  return (F.final+1) % n == F.ini
```

```
Dequeue(F)
  if (is_empty(F))
    error "fila vazia"
  x = F.dados[F.inicio]
  F.inicio = (F.inicio + 1) % n
  return x
```

```
is_empty(F)
  return F.final == F.ini
```