

# ED62A-COM2A

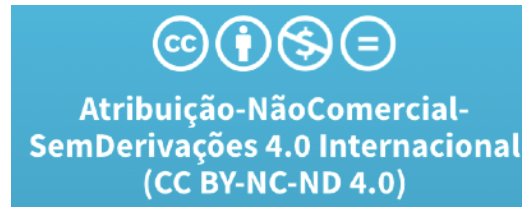
# ESTRUTURAS DE DADOS

Aula 02A - Pilha  
(Implementação estática)

Prof. Rafael G. Mantovani

# Licença

Este trabalho está licenciado com uma Licença CC BY-NC-ND 4.0:



maiores informações:

[https://creativecommons.org/licenses/by-nc-nd/4.0/deed.pt\\_BR](https://creativecommons.org/licenses/by-nc-nd/4.0/deed.pt_BR)

# Roteiro



- 1** Introdução
- 2** Pilhas
- 3** Operações
- 4** Implementação com memória estática
- 5** Síntese / Revisão
- 6** Referências

# Roteiro

- 1 Introdução**
- 2 Pilhas**
- 3 Operações**
- 4 Implementação com memória estática**
- 5 Síntese / Revisão**
- 6 Referências**

# Introdução

- Conjuntos são fundamentais para Computação / Matemática
  - na **Matemática** os conjuntos são invariáveis (inteiros, reais, racionais, etc)
  - já na **Computação** os conjuntos (de dados) são dinâmicos
- Conjuntos possuem **Operações**
  - podemos realizar diferentes operações em um conjunto
  - as mais comuns são operações de "dicionários"
  - **inserir, eliminar e verificar** a existência de um elemento
  - a melhor forma de implementar depende das operações

# Introdução

- Conjuntos são fundamentais para Computação / Matemática
  - na **Matemática** os conjuntos são invariáveis (inteiros, reais, racionais, etc)
  - já na **Computação** os conjuntos são dinâmicos
- Conjuntos para Computação
  - As Estruturas de Dados que estudaremos terão operações de dicionário:
    - **inserir, remover, verificar, etc**
  - podemos definir um conjunto
  - as mais comuns são operações de "dicionários"
  - **inserir, eliminar e verificar** a existência de um elemento
  - a melhor forma de implementar depende das operações

# Introdução

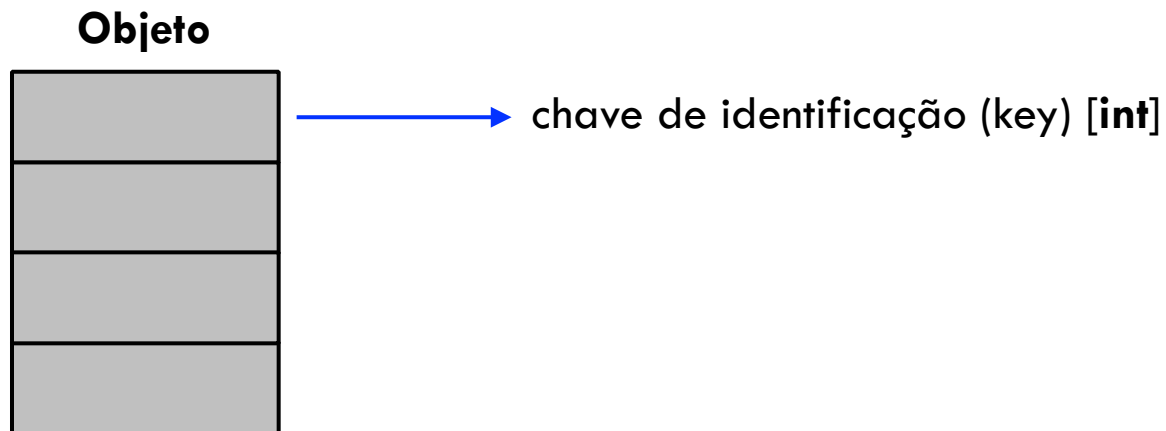
- Elemento (objeto) → vários atributos

## Objeto

Teoricamente podemos armazenar qualquer informação nas estruturas (int, float, char). Porém ... vamos assumir que manipulamos **objetos genéricos (structs)** com chaves/rótulos **inteiros (int)**.

# Introdução

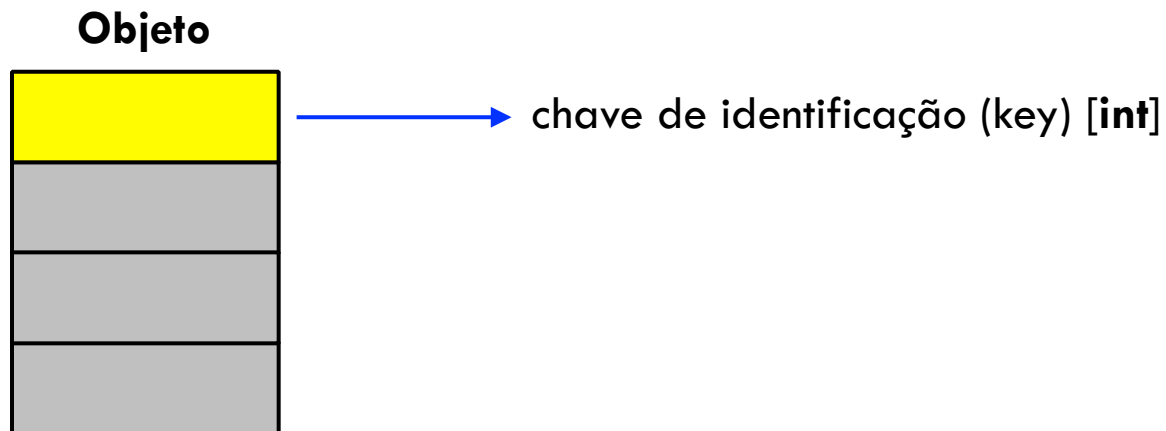
- Elemento (objeto) → vários atributos





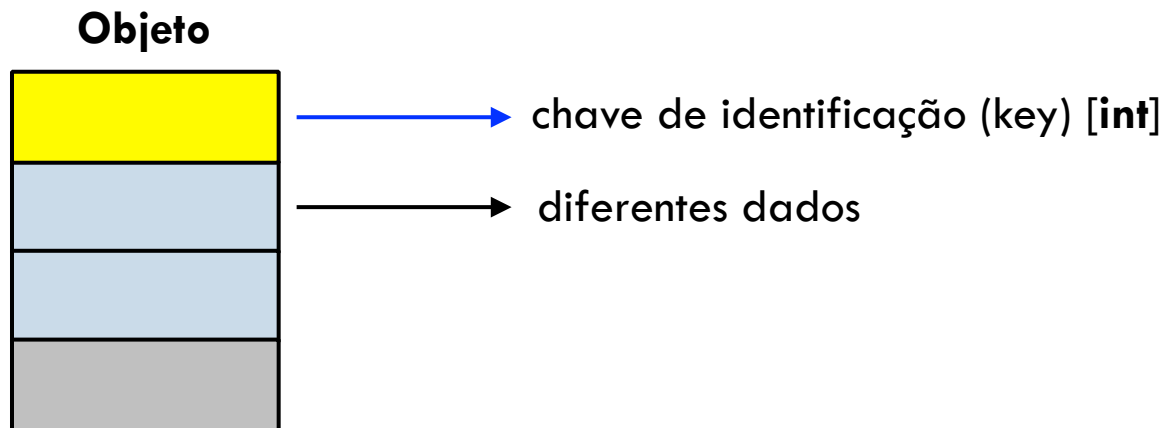
# Introdução

- Elemento (objeto) → vários atributos



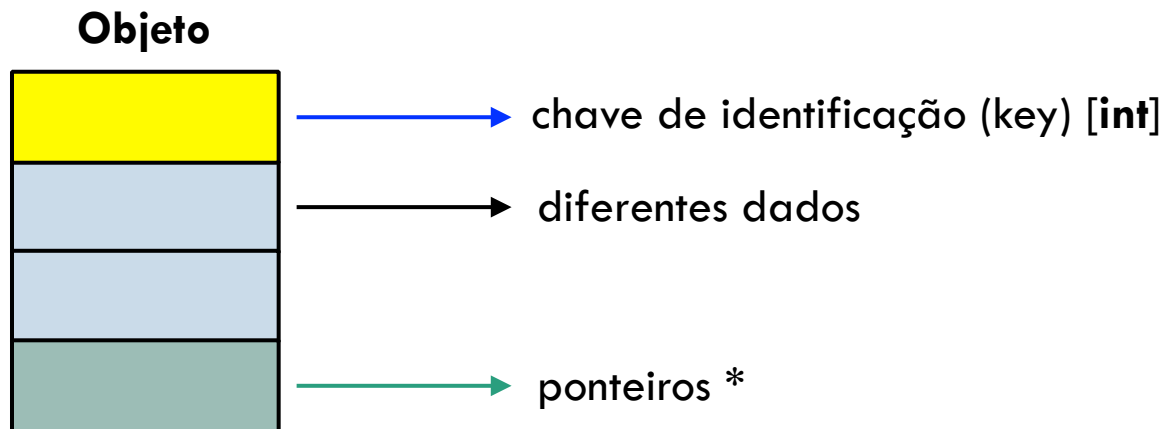
# Introdução

- Elemento (objeto) → vários atributos



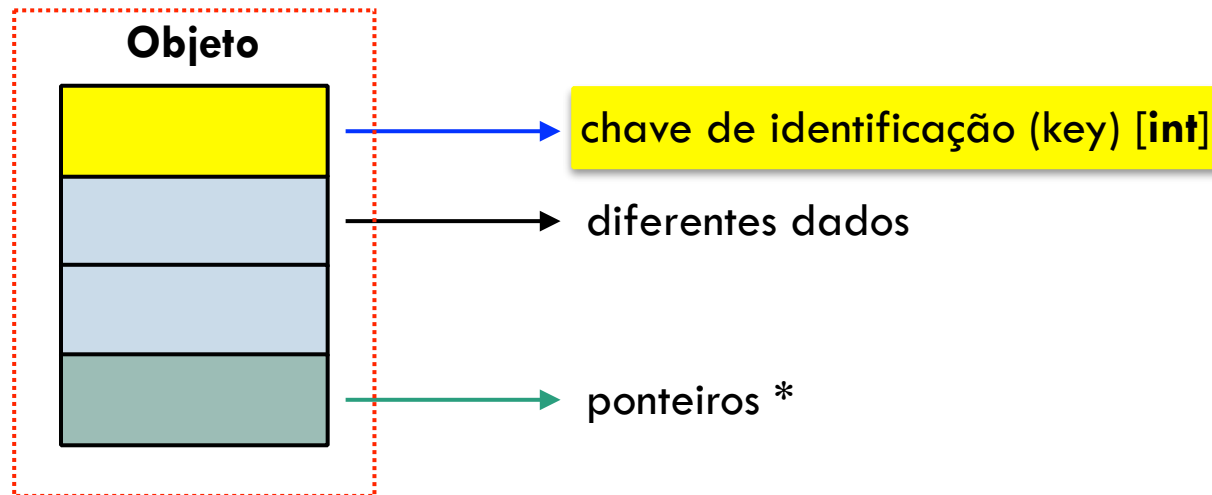
# Introdução

- Elemento (objeto) → vários atributos



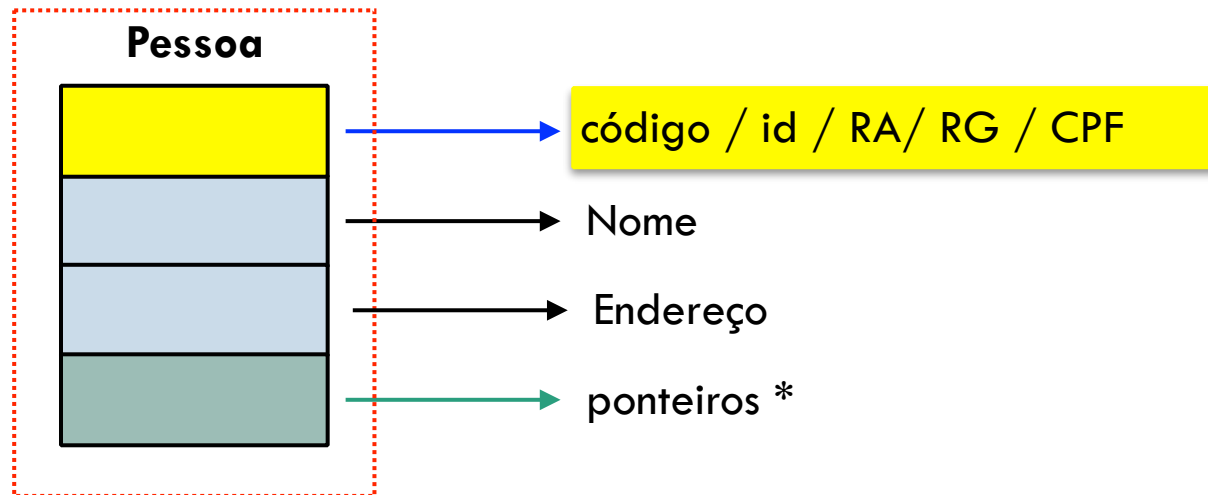
# Introdução

- Elemento (objeto) → vários atributos



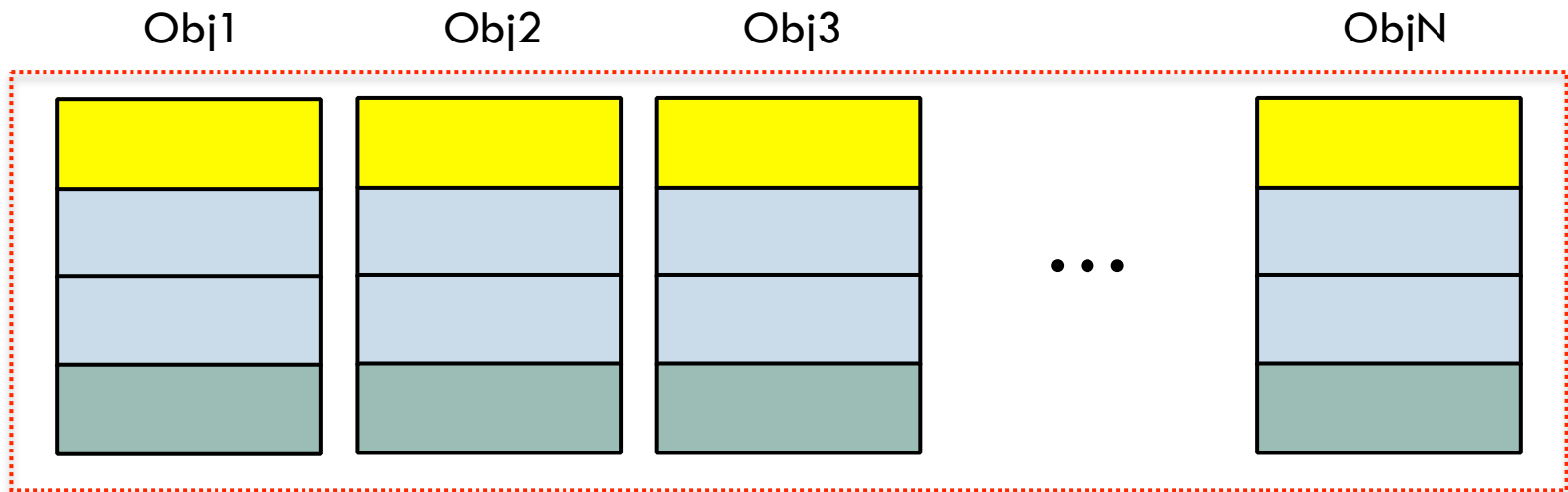
# Introdução

- Exemplo:



# Introdução

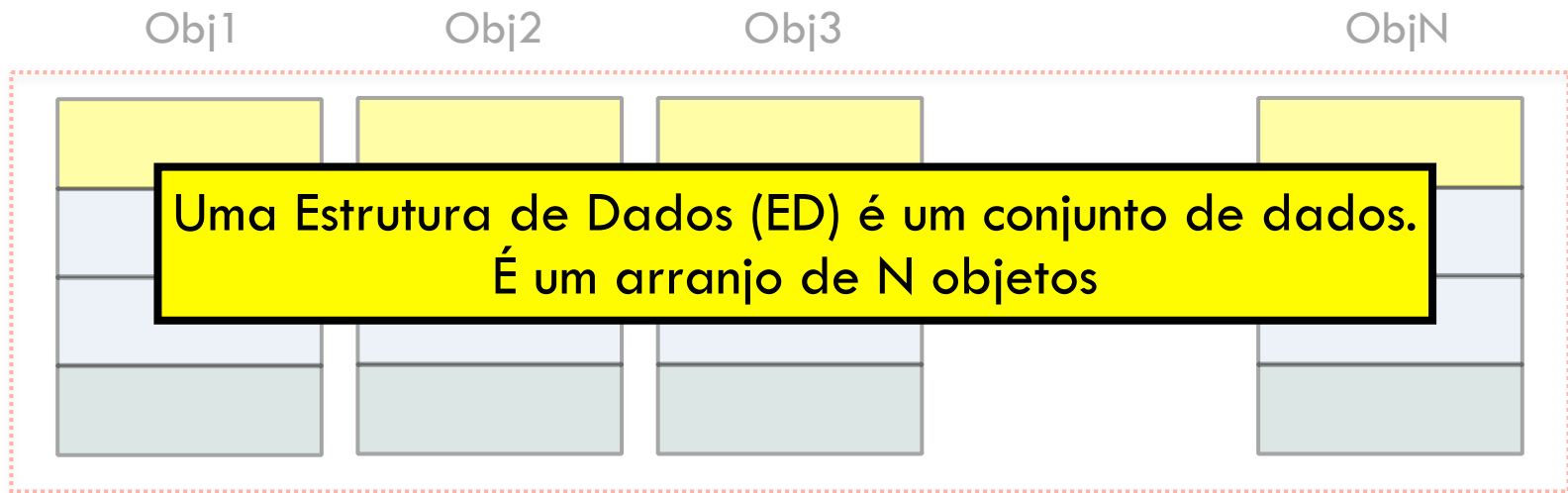
- Estrutura = Arranjo de N Objetos



**Estrutura de Dados**

# Introdução

- Estrutura = Arranjo de N Objetos



Estrutura de Dados

# Operações

Dada uma estrutura **S**, chave **k**, elemento **x**:

Podemos ter operações de **modificação**, e operações **adicionais** condizentes com o tipo da estrutura



# Operações

Dada uma estrutura **S**, chave **k**, elemento **x**:

Podemos ter operações de **modificação**, e operações **adicionais** condizentes com o tipo da estrutura

**S** é a estrutura de Dados

**k** é a chave, valor usado para organizar os elementos

**x** é o objeto a ser inserido na estrutura

# Operações

Dada uma estrutura **S**, chave **k**, elemento **x**:

iniciar (**S**, **x**)

Inserir (**S**, **k**)

Remover (**S**, **k**)

pesquisar (**S**, **k**)

destruir (**S**, **x**)

**Operações de  
modificação**

# Operações

Dada uma estrutura **S**, chave **k**, elemento **x**:

**iniciar (S, x)**

Cria e inicia a estrutura

**Inserir (S, k)**

Insere um novo elemento na estrutura

**Remover (S, k)**

Remove um elemento da estrutura

**pesquisar (S, k)**

Pesquisa um elemento existente na estrutura

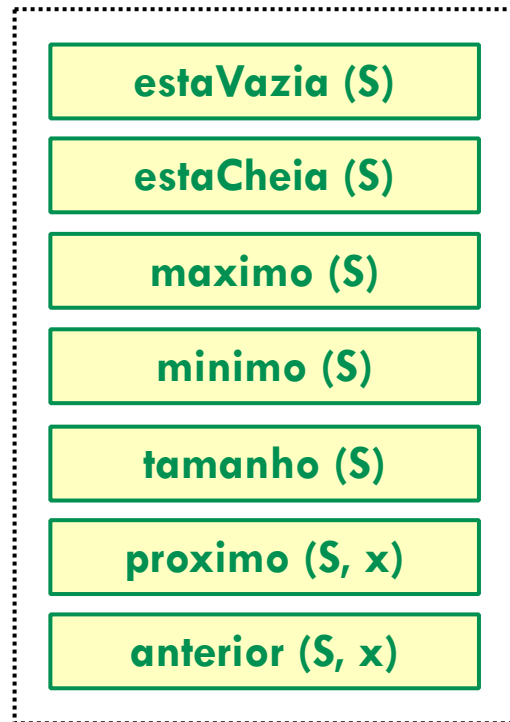
**destruir (S, x)**

Destrói a estrutura e desloca a memória

**Operações de  
modificação**

# Operações

Dada uma estrutura **S**, chave **k**, elemento **x**:



**Operações adicionais  
de consulta**

# Operações

Dada uma estrutura **S**, chave **k**, elemento **x**:

**estaVazia (S)**

Verifica se a estrutura está vazia

**estaCheia (S)**

Verifica se a estrutura está cheia

**maximo (S)**

Retorna o elemento de maior valor

**minimo (S)**

Retorna o elemento de menor valor

**tamanho (S)**

Retorna a quantidade de elementos na estrutura

**proximo (S, x)**

Retorna o próximo elemento (segundo critério)

**anterior (S, x)**

Retorna o elemento anterior (segundo critério)

**Operações adicionais  
de consulta**

# Roteiro

- 1 Introdução
- 2 Pilhas
- 3 Operações
- 4 Implementação com memória estática
- 5 Implementação com memória dinâmica
- 6 Síntese / Revisão
- 7 Referências

# Pilhas



# Pilhas



LIFO (Last In, First Out)

"Último elemento a entrar é o primeiro a sair"



# Pilhas



LIFO (Last In, First Out)

**Pilhas (*Stacks*)** são o tipo mais básico de estrutura que estudaremos :)

primeiro a sair”

# Pilhas

**Topo** da pilha  
(acessível)

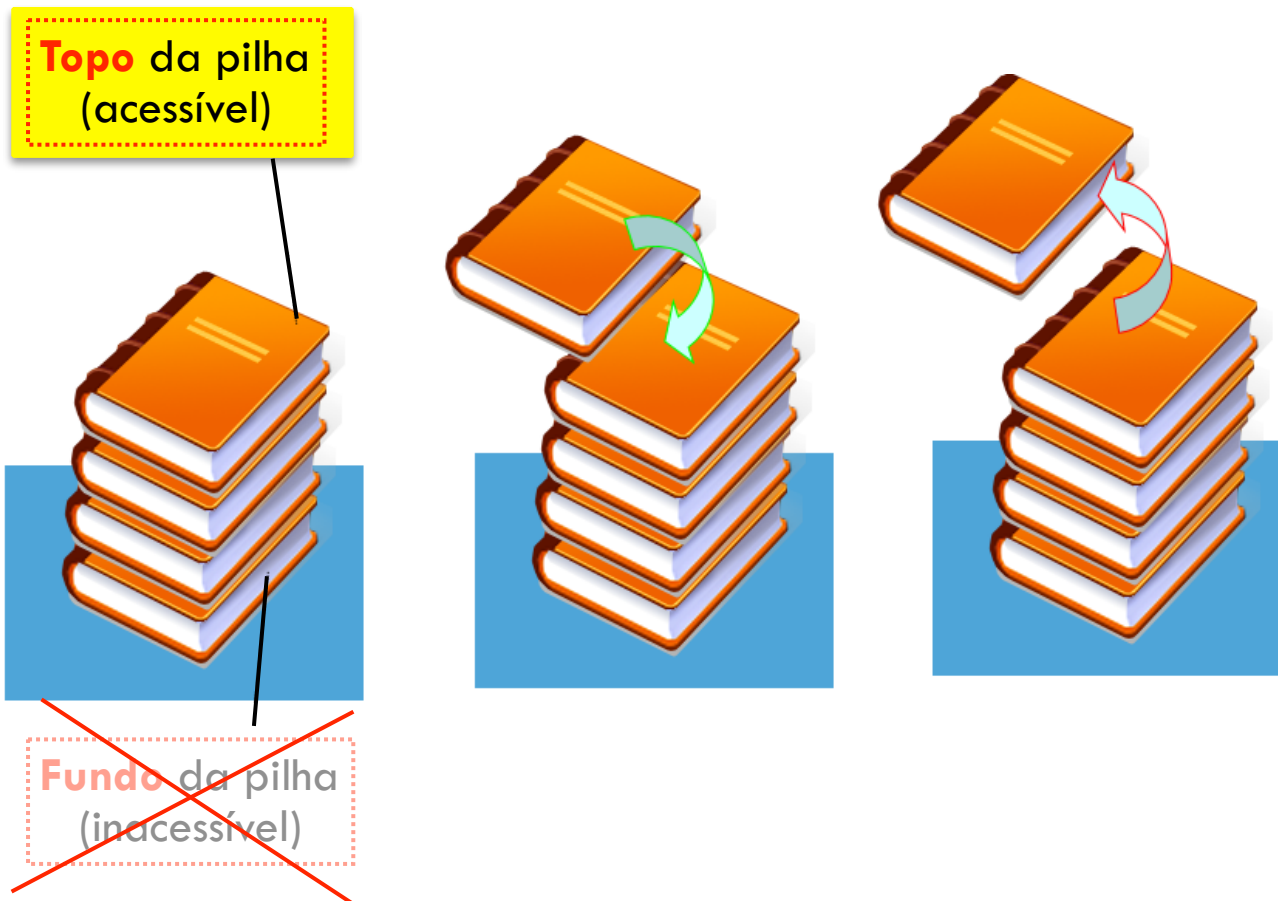


**Fundo** da pilha  
(inacessível)



Pilha de livros  
(Stack)

# Pilhas



# Pilhas

**Topo** da pilha  
(acessível)



Pilha de livros  
(Stack)

~~**Fundo** da pilha  
(inacessível)~~



Empilha um novo  
livro no topo  
(**push**)



Remove um livro  
do topo  
(**pop**)

# Pilhas

**Topo** da pilha  
(acessível)



Pilha de livros  
(Stack)

~~Fundo~~ da pilha  
(inacessível)

## Operações



Empilha um novo  
livro no topo  
(**push**)

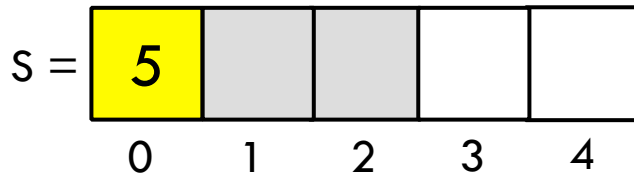


Remove um livro  
do topo  
(**pop**)

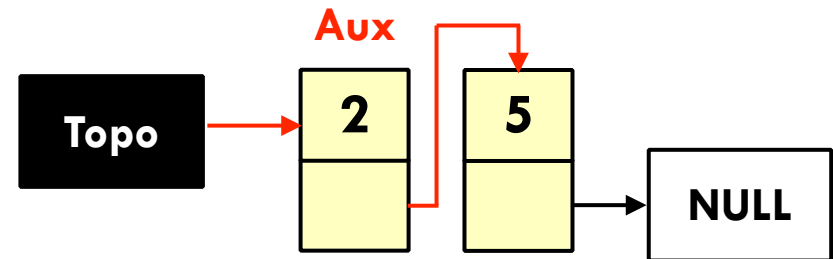
# Roteiro

- 1 Introdução
- 2 Pilhas
- 3 Operações
- 4 Implementação com memória estática
- 5 Síntese / Revisão
- 6 Referências

# Implementação

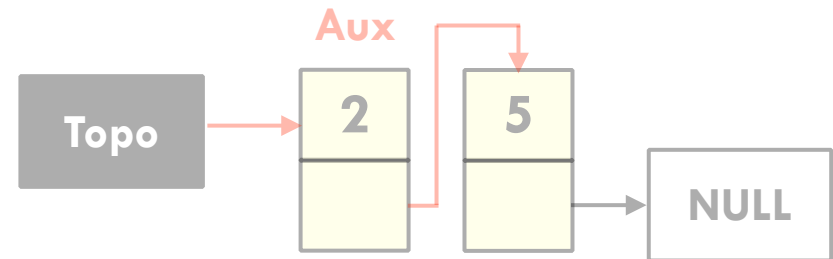
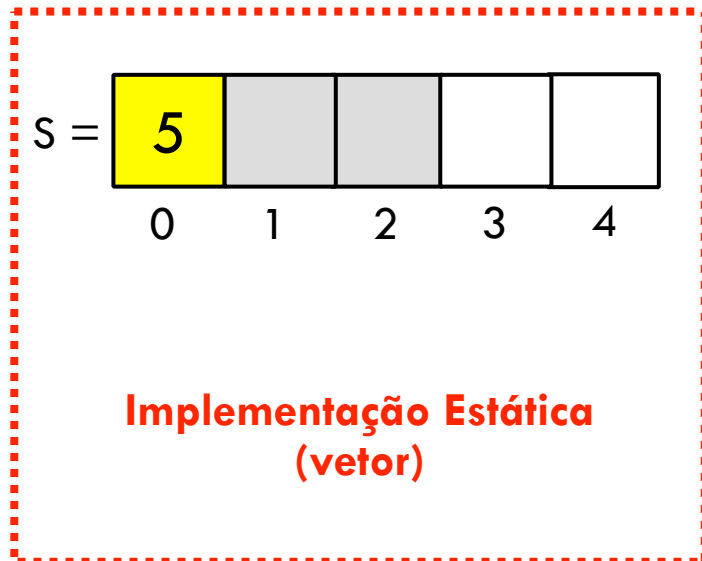


**Implementação Estática  
(vetor)**



**Implementação Dinâmica  
(ponteiros)**

# Implementação



**Implementação Dinâmica  
(ponteiros)**

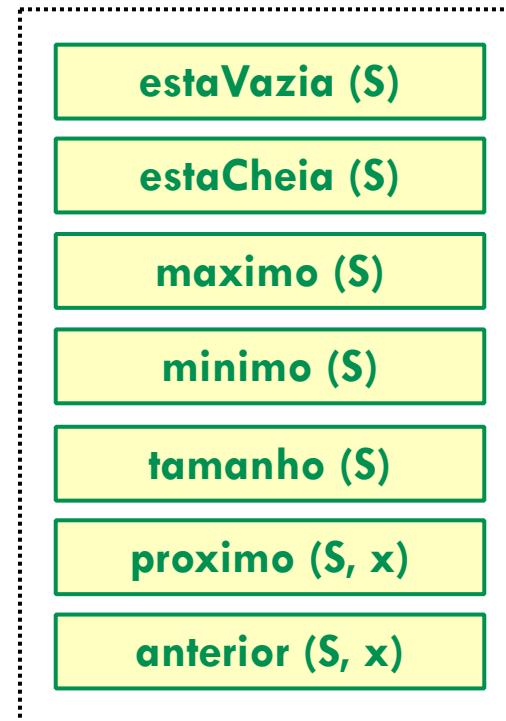


# Operações em Pilhas Estáticas

Dada uma estrutura  $S$ , chave  $k$ , elemento  $x$ :



**Operações de  
modificação**



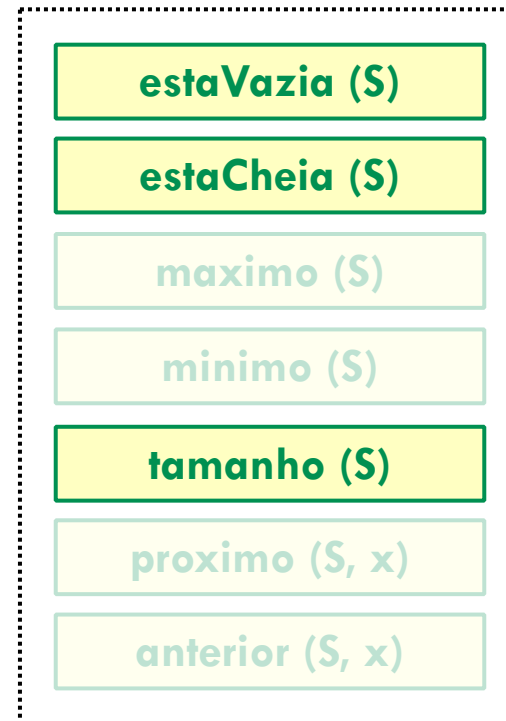
**Operações adicionais  
de consulta**

# Operações em Pilhas Estáticas

Dada uma estrutura  $S$ , chave  $k$ , elemento  $x$ :



**Operações de  
modificação**



**Operações adicionais  
de consulta**

# Operações em Pilhas Estáticas

**iniciar (S)**

Inicializa a pilha e suas variáveis

**Inserir (S, k)**

Inserir objeto na pilha (empilhar)

**Remover (S, k)**

Remover objeto da pilha (desempilhar)

**Topo (S)**

Retorna o objeto do topo, sem remover

**estaVazia (S)**

Retorna booleano indicando se a pilha está vazia

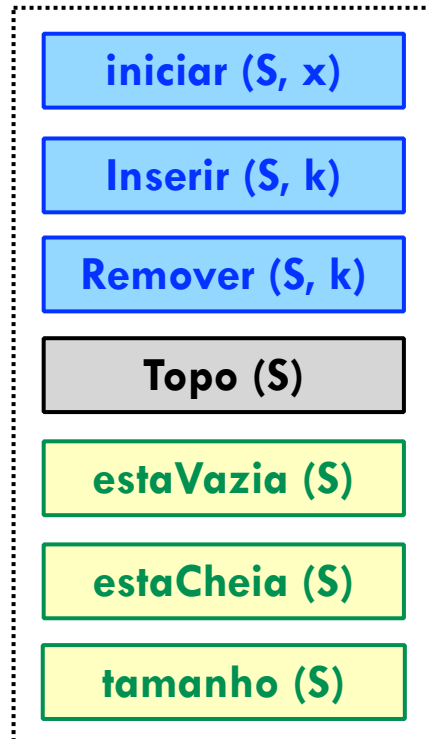
**estaCheia (S)**

Retorna booleano indicando se a pilha está cheia

**tamanho (S)**

Retorna a quantidade de elementos na pilha

# Operações em Pilhas Estáticas



Inicializa a pilha e suas variáveis

Inserir objeto na pilha (empilhar)

Remover objeto da pilha (desempilhar)

Retorna o objeto do topo, sem remover

Retorna booleano indicando se a pilha está vazia

Retorna booleano indicando se a pilha está cheia

Retorna a quantidade de elementos na pilha

**Pilhas estáticas terão todas estas operações !**

# Pseudocódigos

**Função (param1, ...)**

**1. Instrução 1**

**2. ...**

**3. Instrução N**

**4. return (x)**

# Pseudocódigos

Função (P  
1. Instru  
2. ...  
3. Instru  
4. return

Definiremos todas as operações em termos  
de **Pseudocódigo**: independente de  
linguagem de programação. Vamos ver a  
notação:

# Pseudocódigos

**Função** (param1, ...)

1. Instrução 1
2. ...
3. Instrução N
4. **return** (x)

Inputs (entradas)  
- parâmetros

# Pseudocódigos

Função (param1, ...)

1. Instrução 1
2. ...
3. Instrução N
4. return (retornos)

Sequência de  
Instruções



# Pseudocódigos

Função (param1, ...)

1. Instrução 1

2. ...

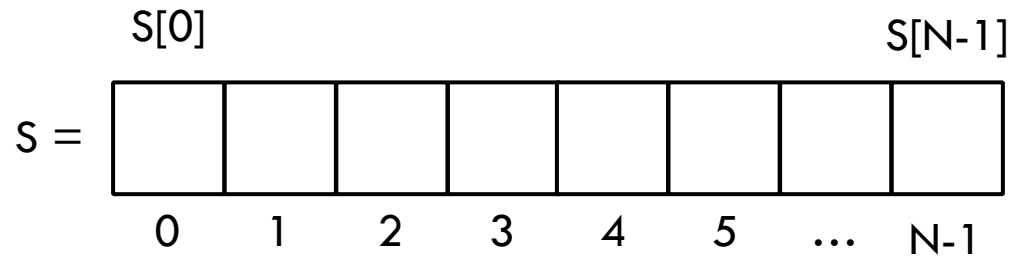
3. Instrução N

4. **return (x)**

Se necessário,  
retornos da função

# Pilhas

$S$  = Arranjo de  $N$  elementos



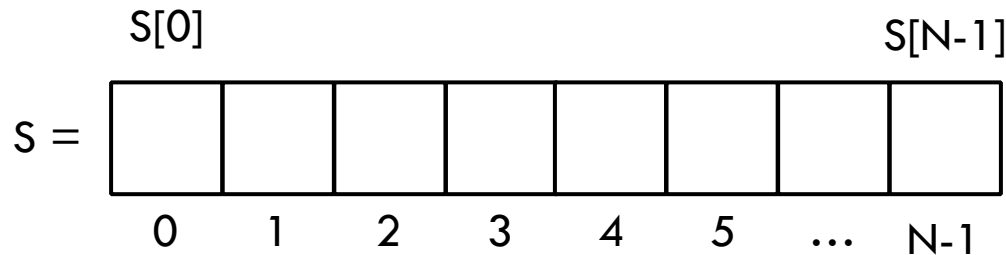
**Topo**

Indexa a posição  
disponível para inserção

# Pilhas

$S$  = Arranjo de  $N$  elementos

Pilha = vetor estático

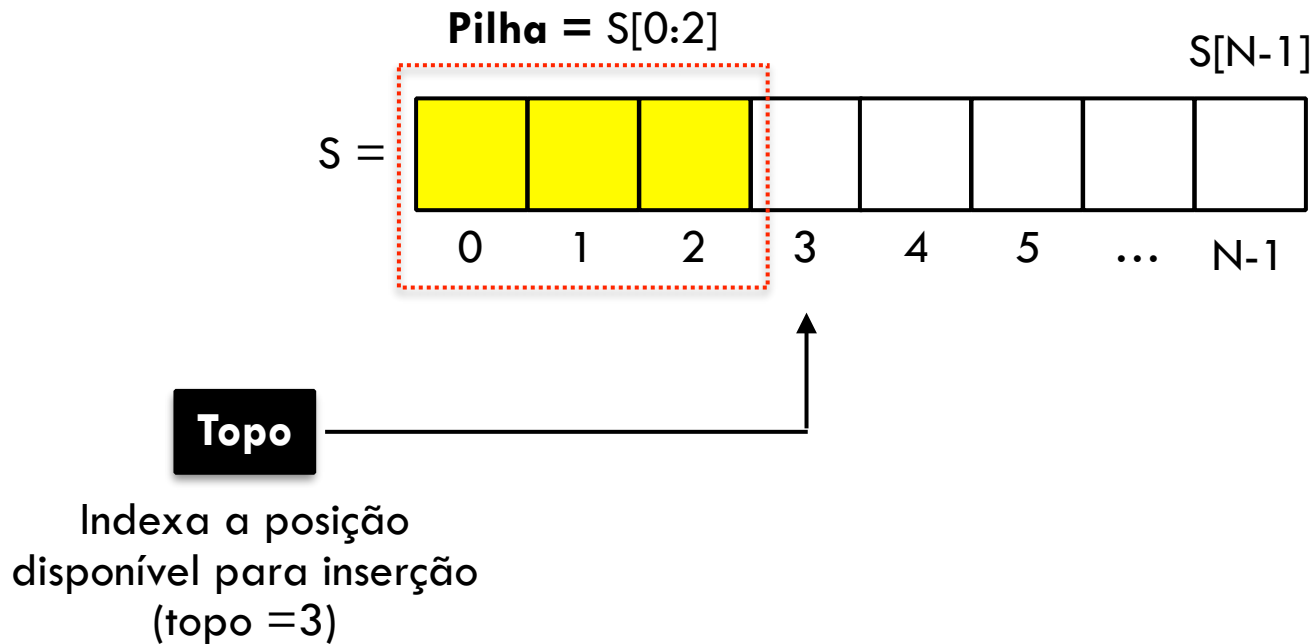


**Topo**

Indexa a posição  
disponível para inserção

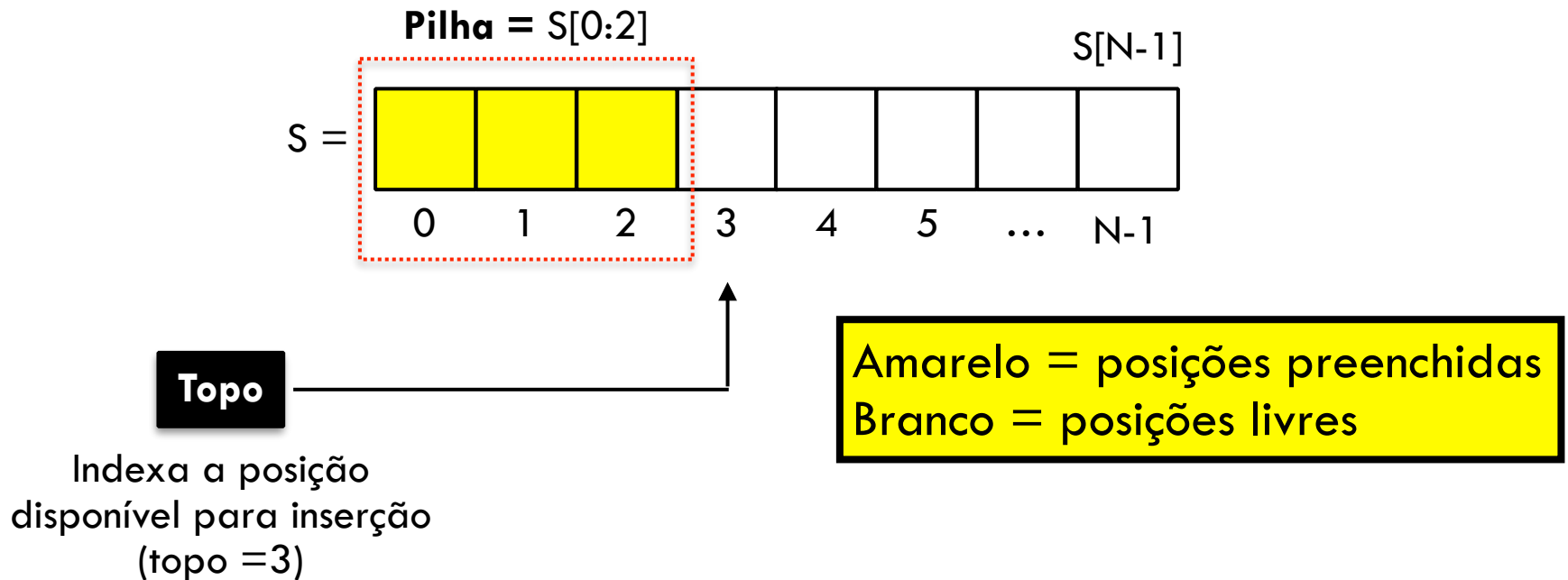
# Pilhas

$S$  = Arranjo de  $N$  elementos



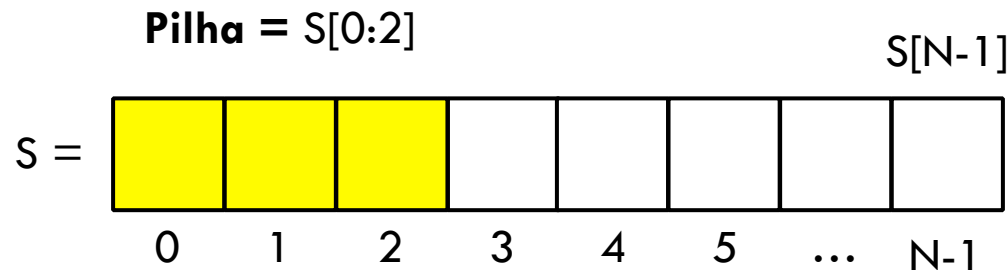
# Pilhas

$S$  = Arranjo de  $N$  elementos



# Pilhas

$S$  = Arranjo de  $N$  elementos



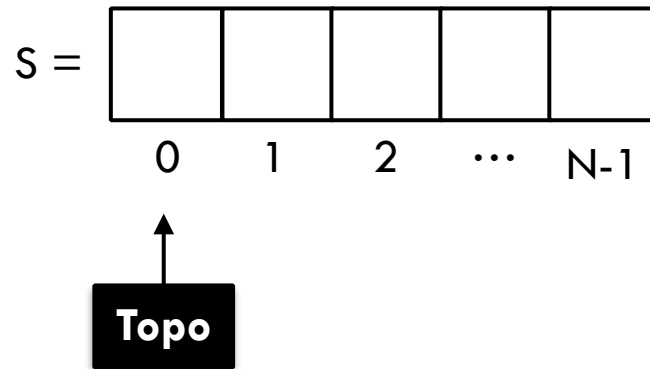
**Topo**

Indexa a posição  
disponível para inserção  
( $\text{topo} = 3$ )

Controle é feito por uma  
variável inteira (**topo**) que  
especifica qual posição  
podemos operar

# Inicializar Pilha

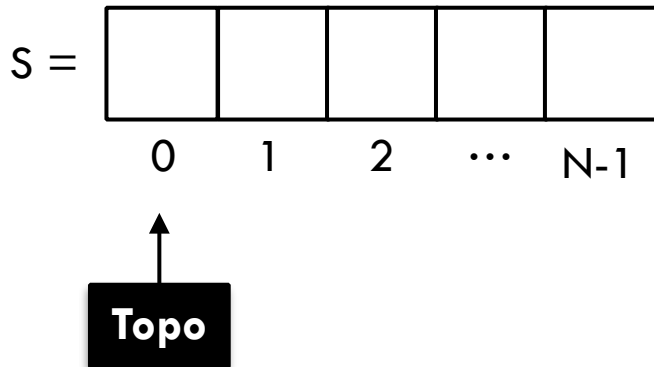
- $S[S.topo] = 0 \rightarrow$  1a posição válida



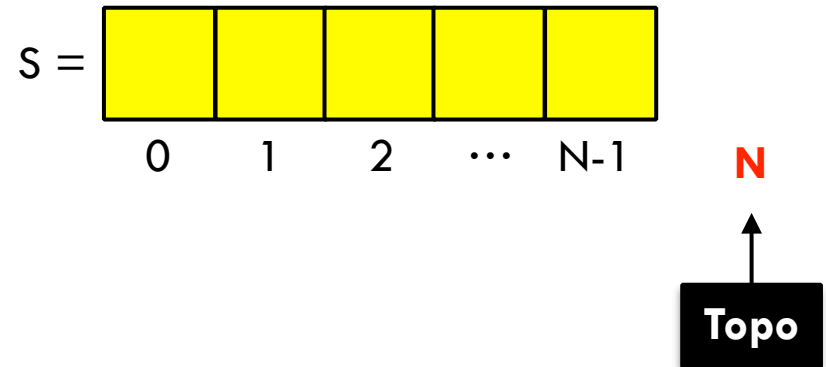
```
iniciar(S)  
1.  $S.topo = 0$ 
```

# Estados: vazio e cheio

- $S[S.topo] == 0 \rightarrow$  pilha está vazia
- $S[S.topo] == N \rightarrow$  pilha está cheia



**estaVazia (S)**  
1. `return(S.topo == 0)`

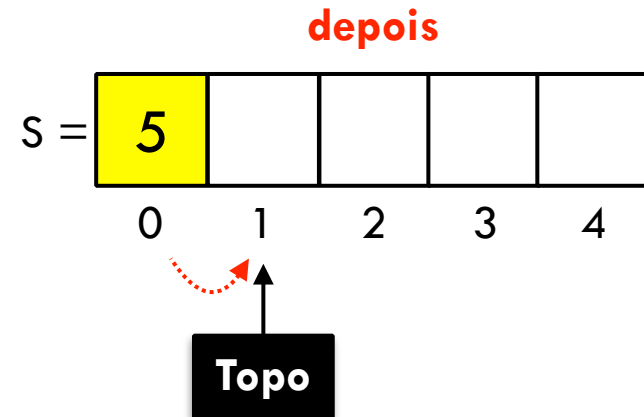
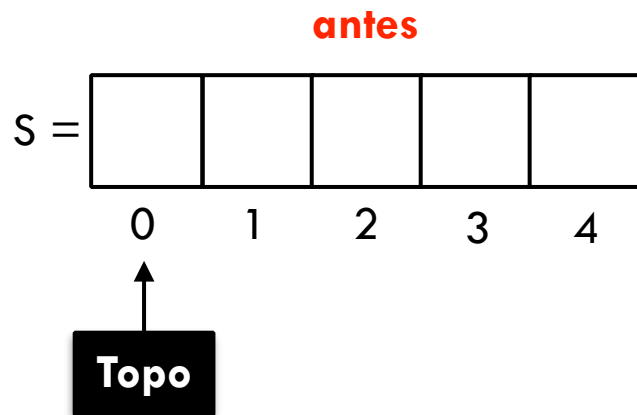


**estaCheia (S)**  
1. `return(S.topo == N)`



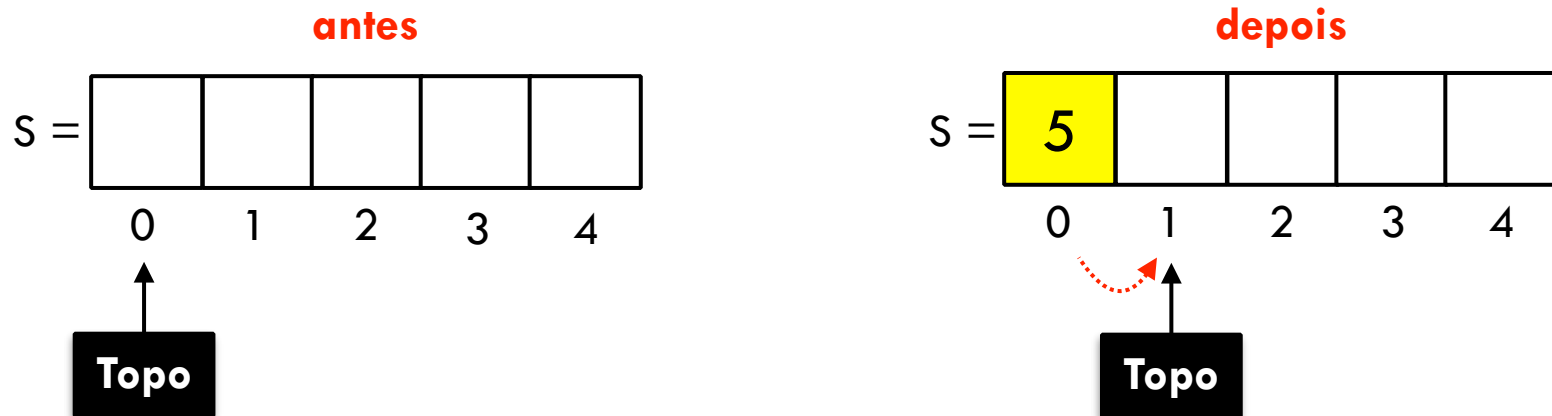
# Empilhar

- Empilhar (inserir) elemento  $x = 5$



# Empilhar

- Empilhar (inserir) elemento  $x = 5$

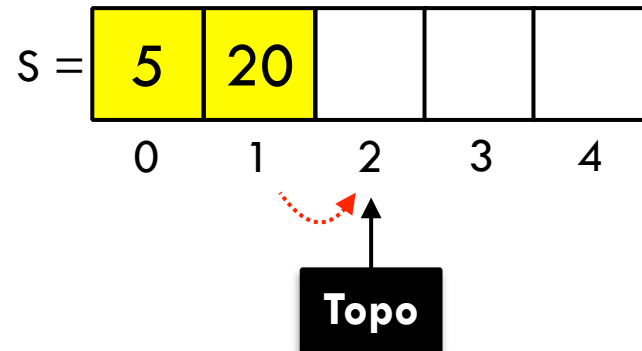
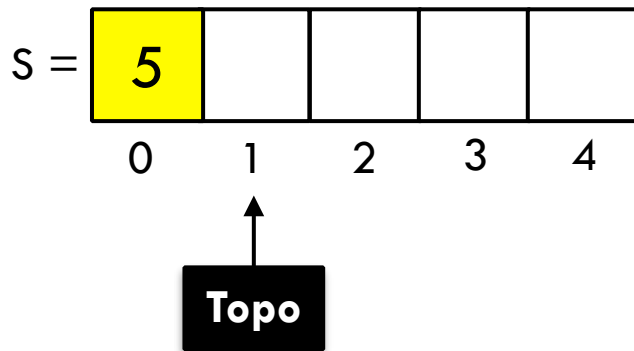


- Pseudocódigo**

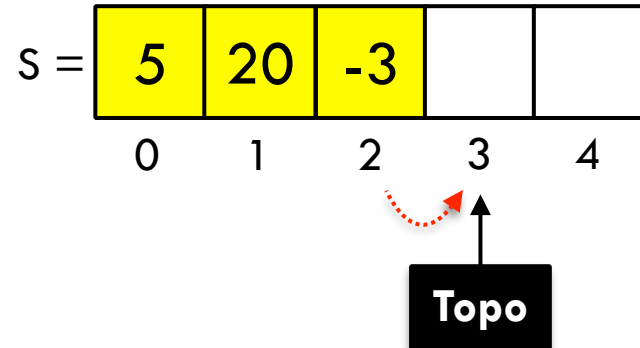
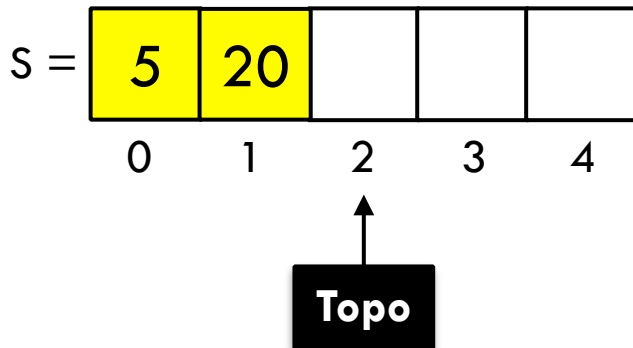
```
Empilha (S, x)
1. SE estaCheia(S) == FALSE
2.   S[S.topo] = x;
3.   S.topo = S.topo + 1;
```

# Empilhar (Push)

- Empilhar (inserir) elemento  $x = 20$

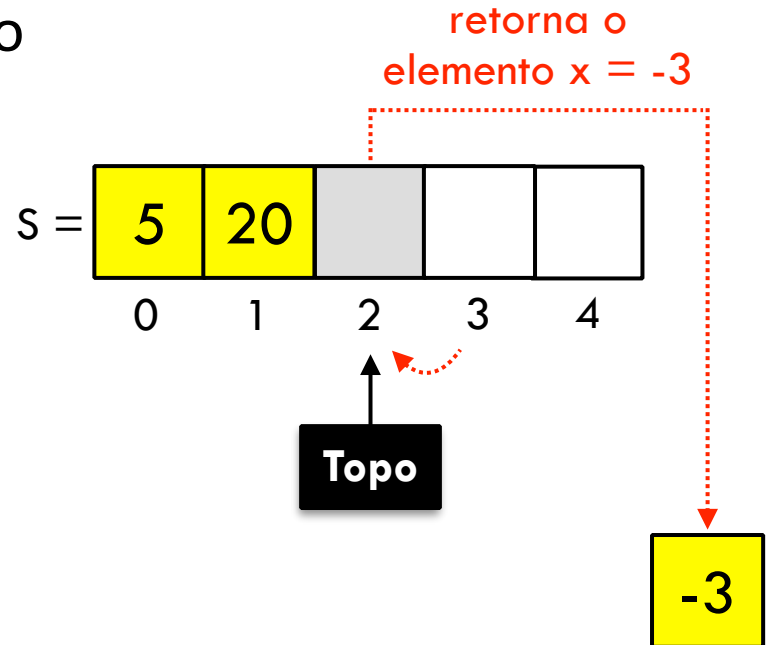
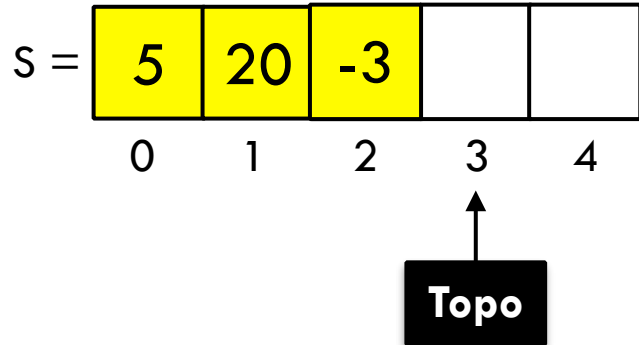


- Empilhar (inserir) elemento  $x = -3$



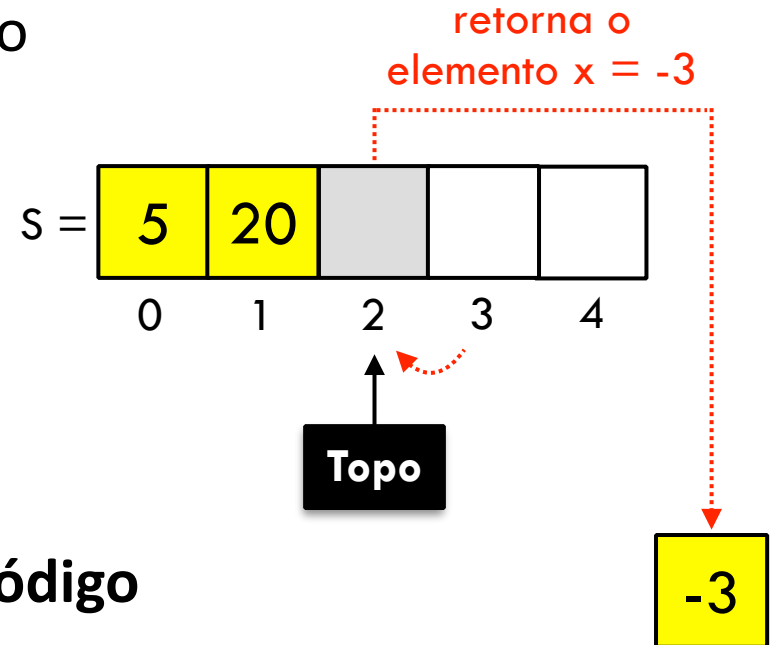
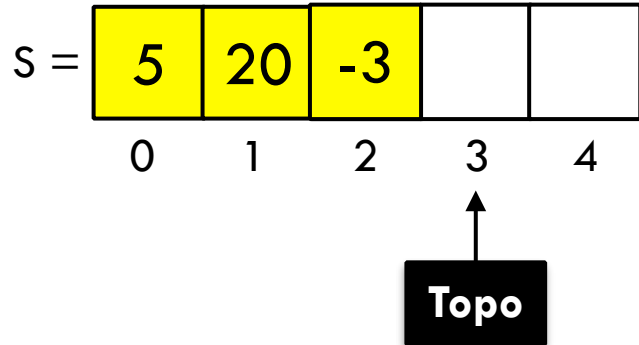
# Desempilhar

- desempilhar (remove) elemento



# Desempilhar

- desempilhar (remover) elemento



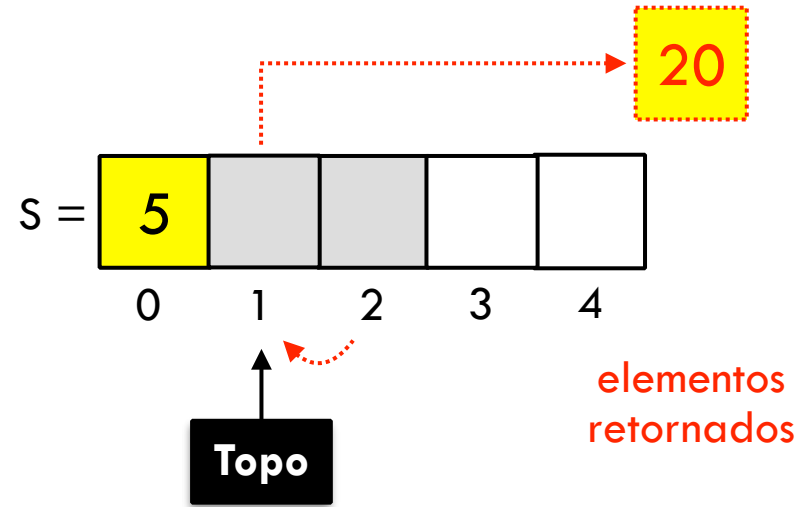
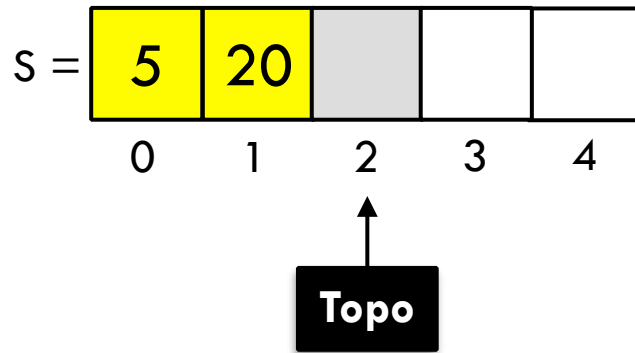
- Pseudocódigo

Desempilha (S)

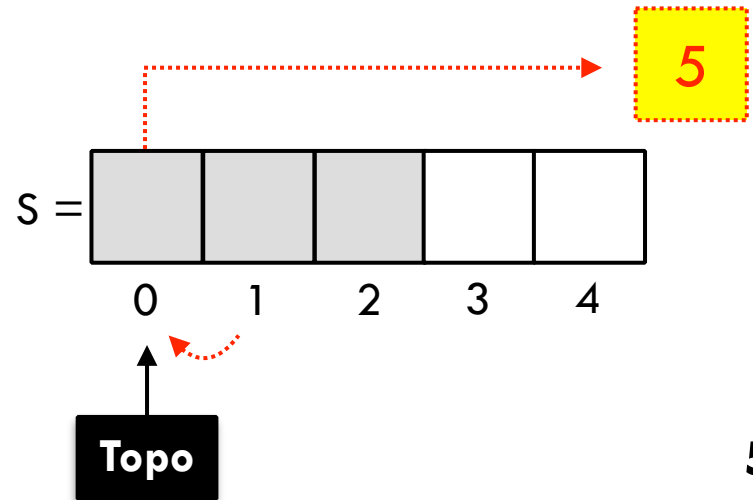
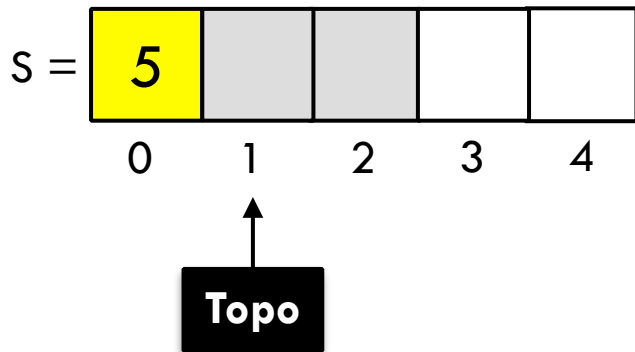
```
1. SE estaVazia(S) == FALSE
2.   x = S[S.topo - 1];
3.   S.topo = S.topo - 1;
4.   return(x);
```

# Desempilhar

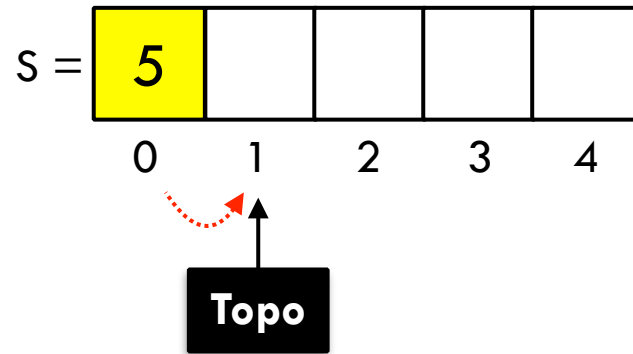
- desempilhar (remove) elemento



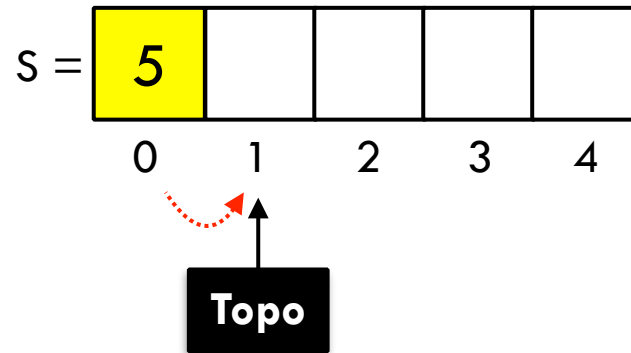
- desempilhar (remove) elemento



# Acessar topo (sem remoção)



# Acessar topo (sem remoção)



- Pseudocódigo

**Topo (S)**

```
1. x = [S.topo-1];  
2. return(x);
```



# Exercício 01

- Ilustre cada estado de uma pilha após realizar as seguintes operações (em ordem)
  - (A) `Push(S, 4)` *// empilha o elemento 4*
  - (B) `Push(S, 1)` *// empilha o elemento 1*
  - (C) `Push(S, 3)` *// empilha o elemento 3*
  - (D) `Pop(S)` *// desempilha o valor do topo*
  - (E) `Push(S, 8)` *// empilha o elemento 8*
  - (F) `Pop(S)` *// desempilha o valor do topo*
- Considere que a pilha está inicialmente vazia e é armazenada em um arranjo `S [1 .. 6]`

# Roteiro

- 1 Introdução
- 2 Pilhas
- 3 Operações
- 4 Implementação com memória estática
- 5 Síntese / Revisão
- 6 Referências

# Implementação (Estática)

- Dois tipos abstratos de dados

```
#define N 100

typedef struct {
    int key;
    /* pode ter mais elementos */
} Objeto;
```

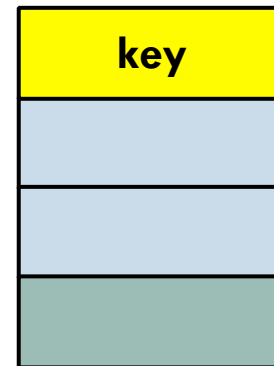
# Implementação (Estática)

- Dois tipos abstratos de dados

```
#define N 100

typedef struct {
    int key;
    /* pode ter mais elementos */
} Objeto;
```

implementa o nosso  
objeto



# Implementação (Estática)

- Dois tipos abstratos de dados

```
#define N 100

typedef struct {
    int key;
    /* pode ter mais elementos */
} Objeto;
```

.....→ implementa o nosso  
objeto

```
typedef struct {
    Objeto vetor[N];
    int topo;
} PilhaEstatica;
```

# Implementação (Estática)

- Dois tipos abstratos de dados

```
#define N 100

typedef struct {
    int key;
    /* pode ter mais elementos */
} Objeto;
```

implementa o nosso  
objeto

```
typedef struct {
    Objeto vetor[N];
    int topo;
} PilhaEstatica;
```

implementa o TAD  
para Pilha Estática  
(armazena Objetos)

# Implementação (Estática)

- Dois tipos abstratos de dados

```
void iniciaPilha(PilhaEstatica *pilha);
bool estaVazia(PilhaEstatica *pilha);
bool estaCheia(PilhaEstatica *pilha);
void push(Objeto obj, PilhaEstatica *pilha);
void pop(PilhaEstatica *pilha, Objeto *obj);
int size(PilhaEstatica *pilha);
Objeto top(PilhaEstatica *pilha);
void print(PilhaEstatica *pilha);
```

## Exercício 02

- Mãos a obra: implemente um TDA para Pilha com alocação estática, e as funções de manipulação.



# Roteiro

- 1 Introdução
- 2 Pilhas
- 3 Operações
- 4 Implementação com memória estática
- 5 Síntese / Revisão
- 6 Referências

# Revisão



- Pilhas
  - o que é
  - operações
  - implementação estática

# Próximas Aulas

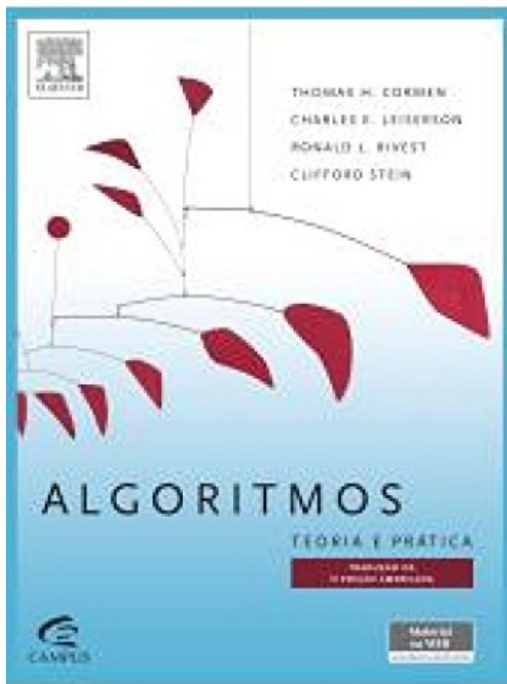


- Pilhas → implementação dinâmica
- Filas/ Deques
- Implementação de Listas Lineares
  - single-linked
  - double-linked

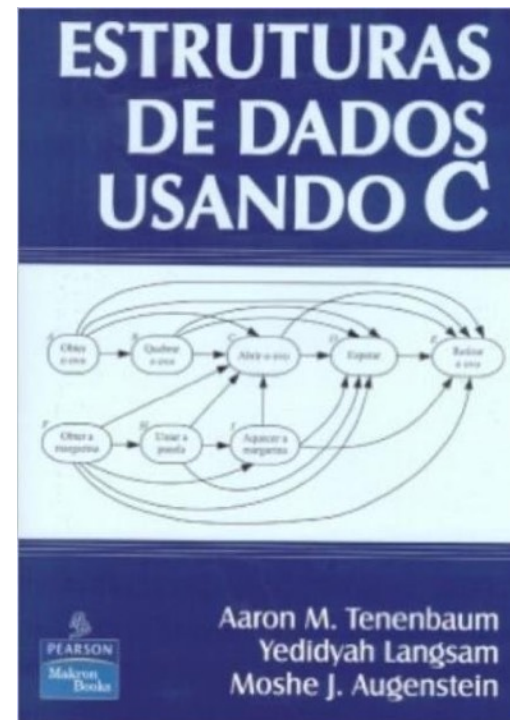
# Roteiro

- 1 Introdução
- 2 Pilhas
- 3 Operações
- 4 Implementação com memória estática
- 5 Síntese / Revisão
- 6 Referências

# Referências sugeridas

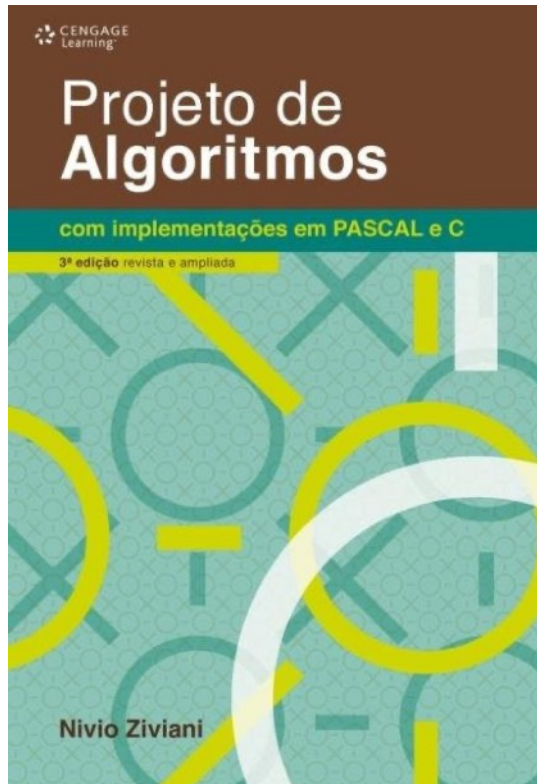


[Cormen et al, 2018]



[Tenenbaum et al, 1995]

# Referências sugeridas



[Ziviani, 2010]



[Drozdek, 2017]

# Perguntas?

Prof. Rafael G. **Mantovani**

[rafaelmantovani@utfpr.edu.br](mailto:rafaelmantovani@utfpr.edu.br)