

Algoritmos e Estruturas de Dados I

Aula 04: Pilhas

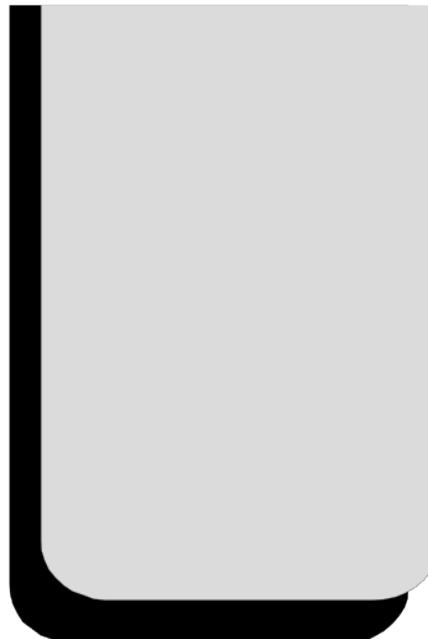
Prof. Márcio Porto Basgalupp

créditos: Prof. Jurandy G. Almeida Jr.

Universidade Federal de São Paulo
Departamento de Ciência e Tecnologia

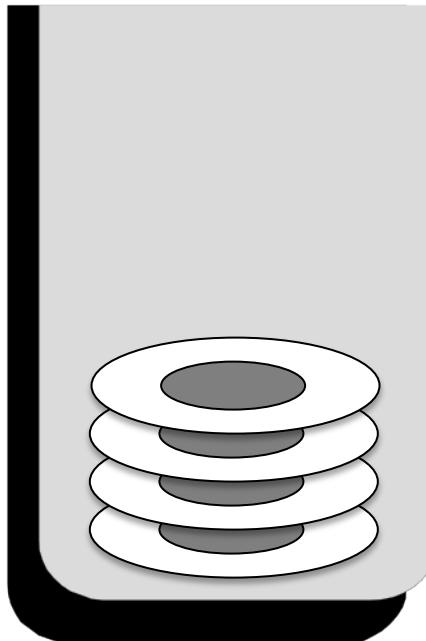
O que é uma pilha?

- Uma **pilha** é uma estrutura de dados na qual todas as inserções, retiradas e, geralmente, todos os acessos são efetuados numa única extremidade, denominada **topo**



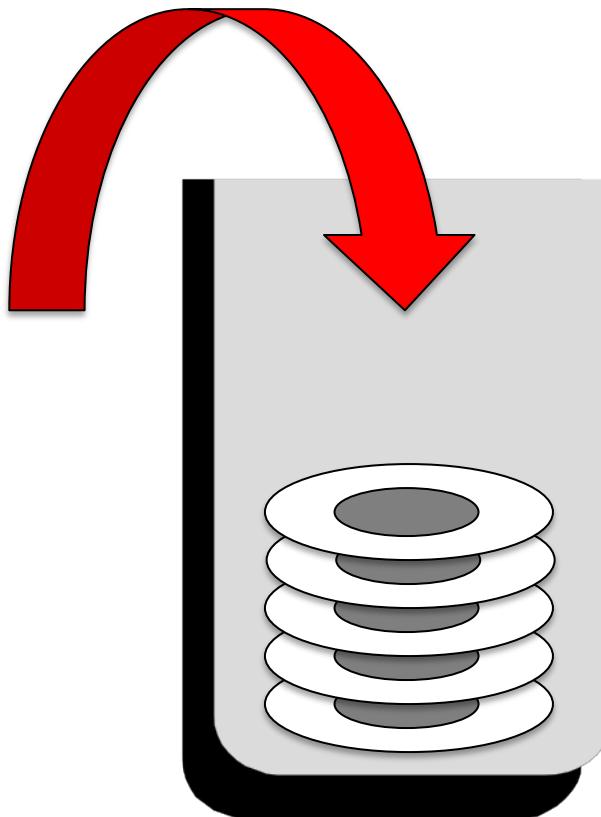
O que é uma pilha?

- Por exemplo, pense numa pilha de pratos comumente encontrada em restaurantes do tipo *self-service*



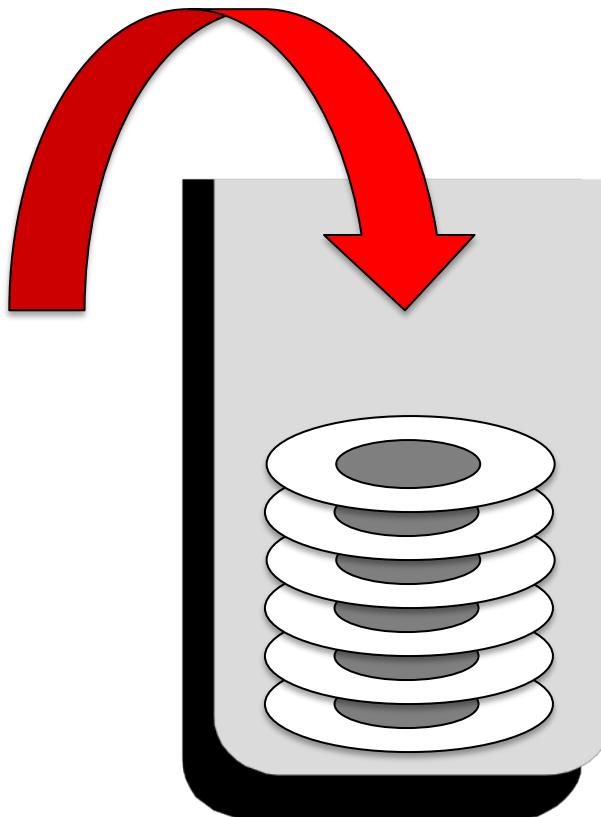
O que é uma pilha?

- Por exemplo, pense numa pilha de pratos comumente encontrada em restaurantes do tipo *self-service*
- Os pratos são colocados sempre no topo da pilha pelos empregados ...



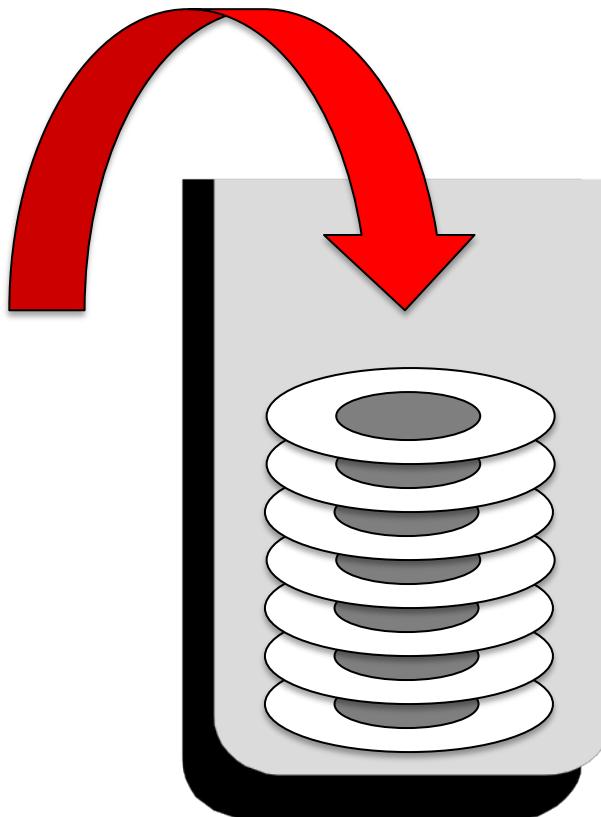
O que é uma pilha?

- Por exemplo, pense numa pilha de pratos comumente encontrada em restaurantes do tipo *self-service*
- Os pratos são colocados sempre no topo da pilha pelos empregados ...



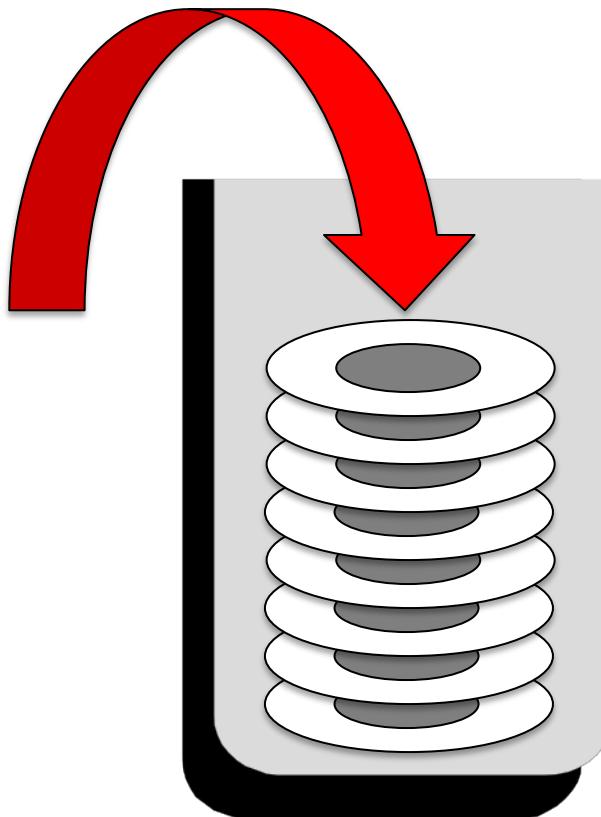
O que é uma pilha?

- Por exemplo, pense numa pilha de pratos comumente encontrada em restaurantes do tipo *self-service*
- Os pratos são colocados sempre no topo da pilha pelos empregados ...



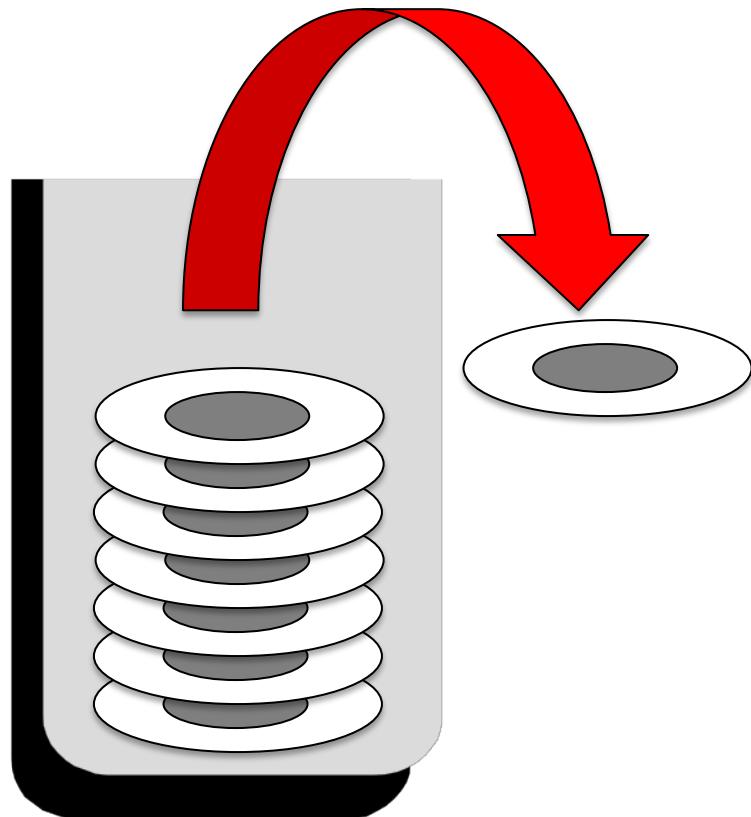
O que é uma pilha?

- Por exemplo, pense numa pilha de pratos comumente encontrada em restaurantes do tipo *self-service*
- Os pratos são colocados sempre no topo da pilha pelos empregados ...



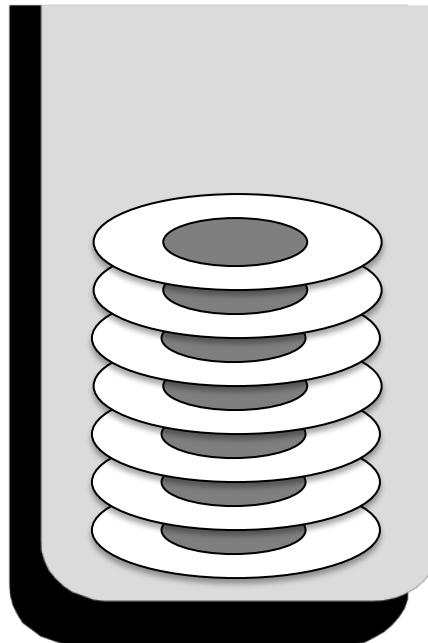
O que é uma pilha?

- Por exemplo, pense numa pilha de pratos comumente encontrada em restaurantes do tipo *self-service*
- Os pratos são colocados sempre no topo da pilha pelos empregados e são retirados a partir do topo pelos clientes



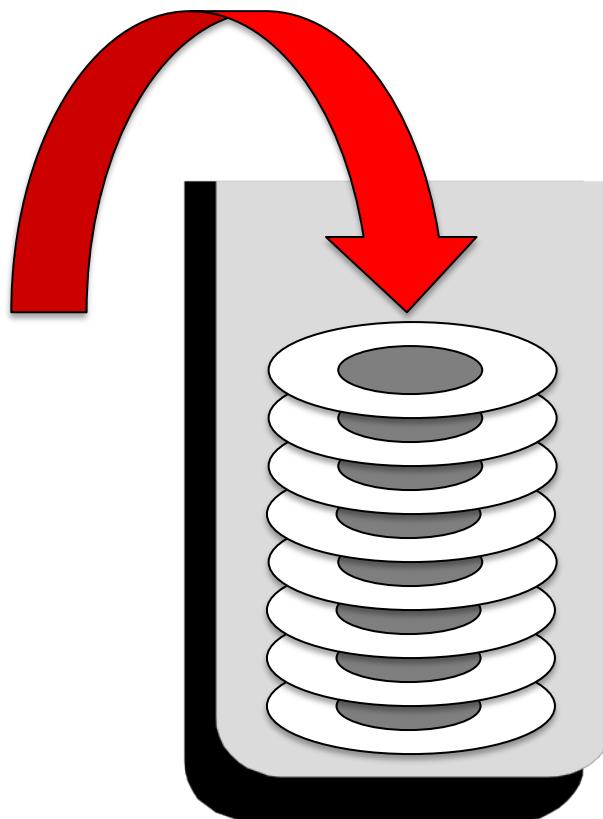
O que é uma pilha?

- O prato localizado no fundo da pilha foi o primeiro a ser colocado na pilha e é o último a ser retirado



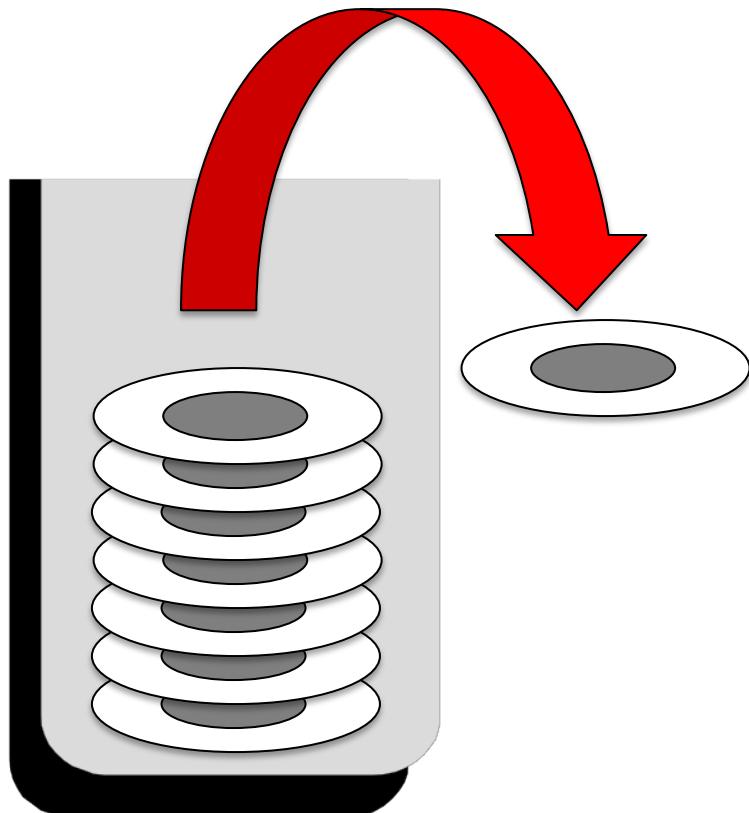
Operações Básicas

- Quando um item é adicionado numa pilha, usa-se a operação **Empilhar** (inserir)



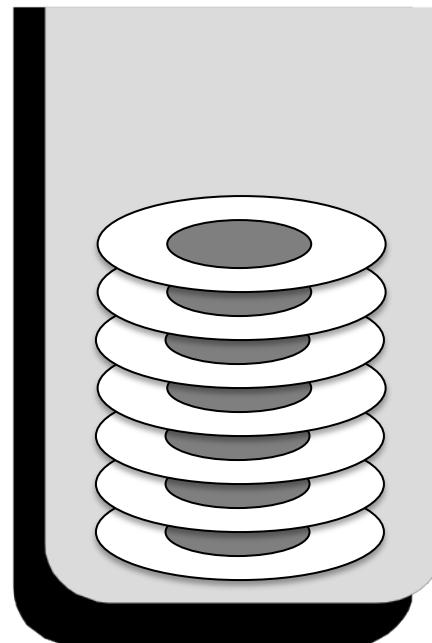
Operações Básicas

- Quando um item é adicionado numa pilha, usa-se a operação **Empilhar** (inserir)
- Quando um item é retirado de uma pilha, usa-se a operação **Desempilhar** (retirar)



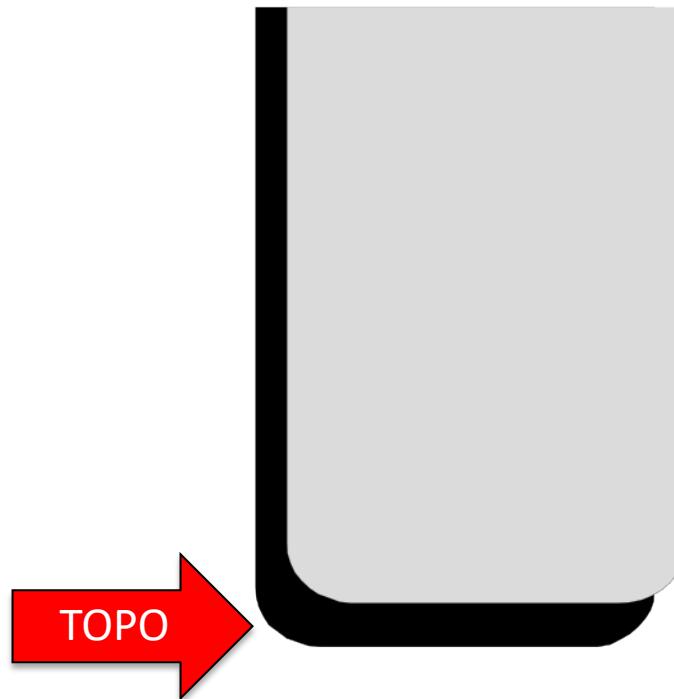
Propriedades

- O **último** item inserido na pilha é sempre o **primeiro** a ser retirado
- Essa propriedade é denominada *Last In, First Out* ou **LIFO**
- Existe uma ordem linear, do **mais recente para o menos recente**



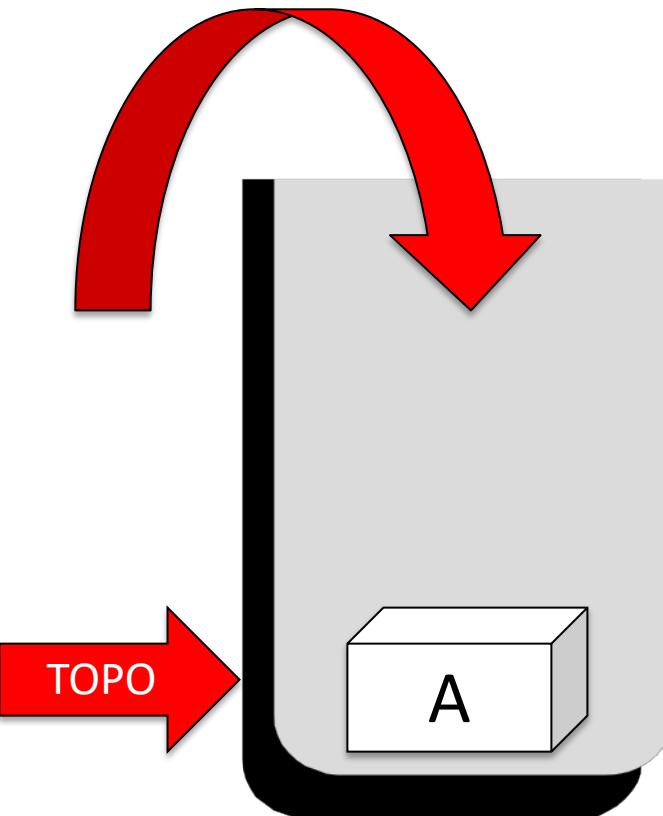
Exemplo

- pilha inicialmente vazia



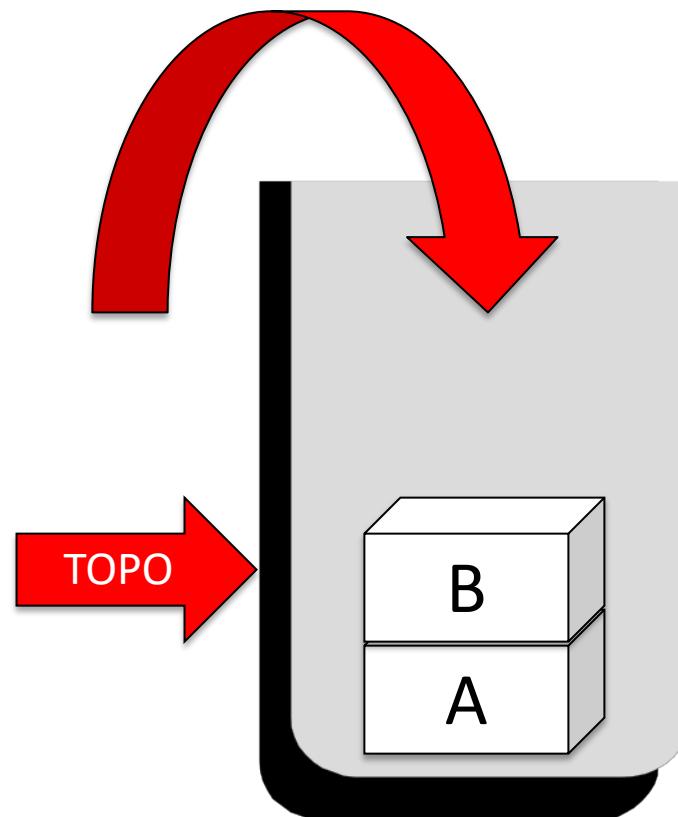
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A



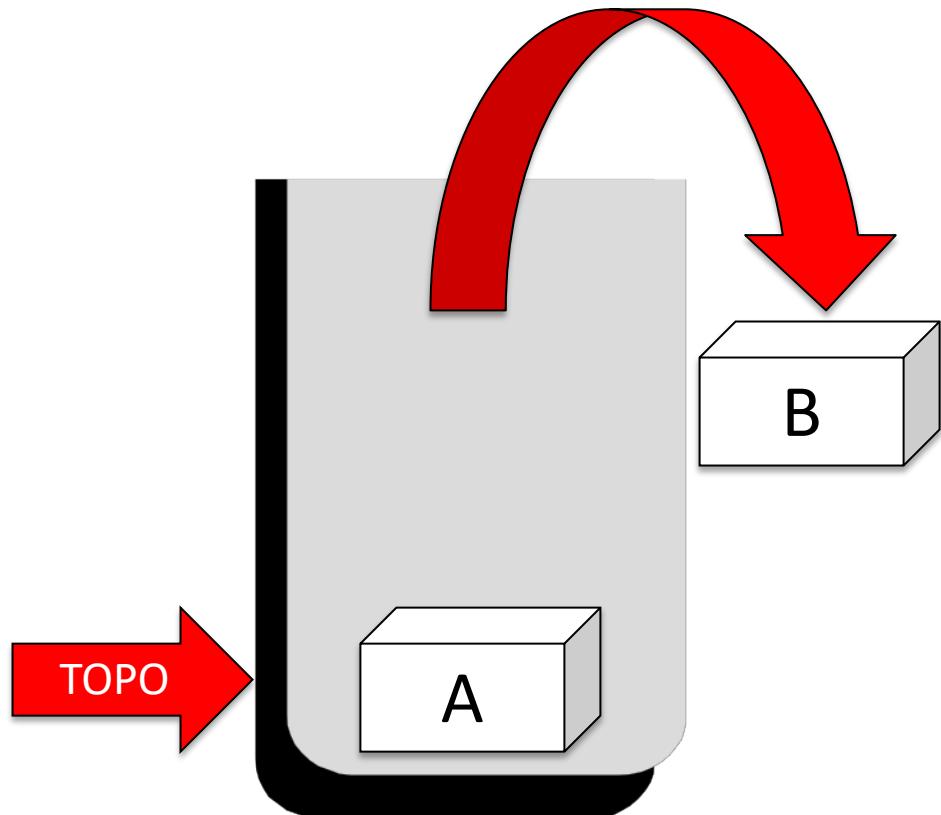
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B



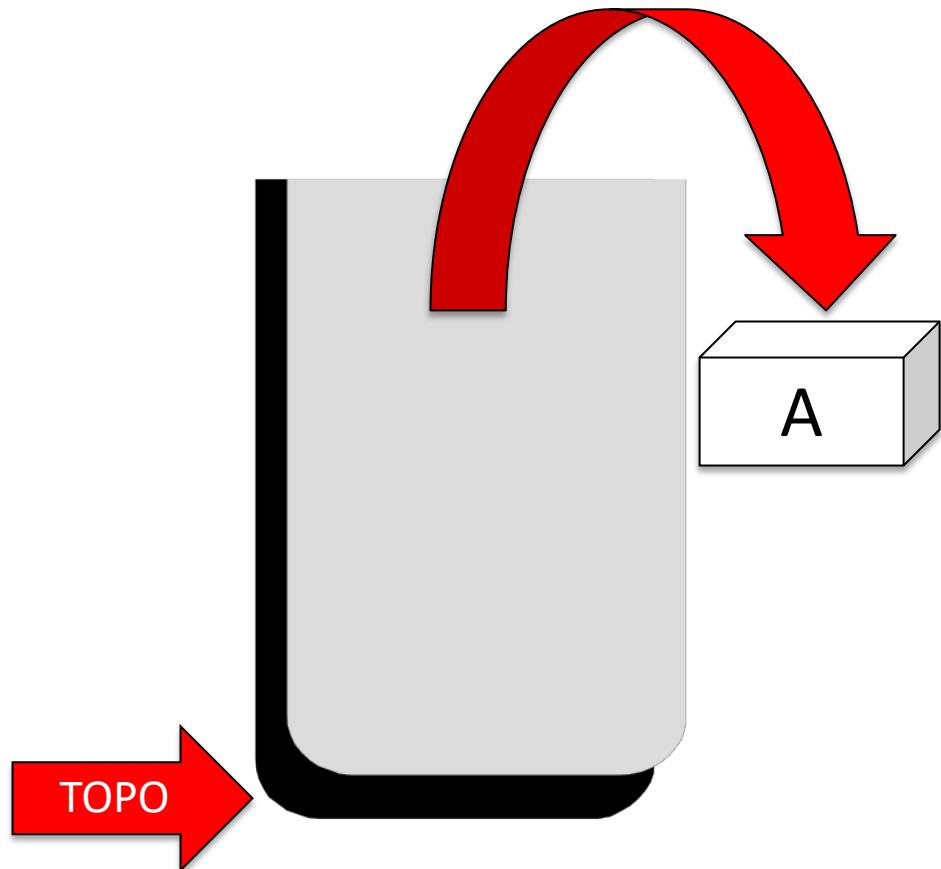
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item



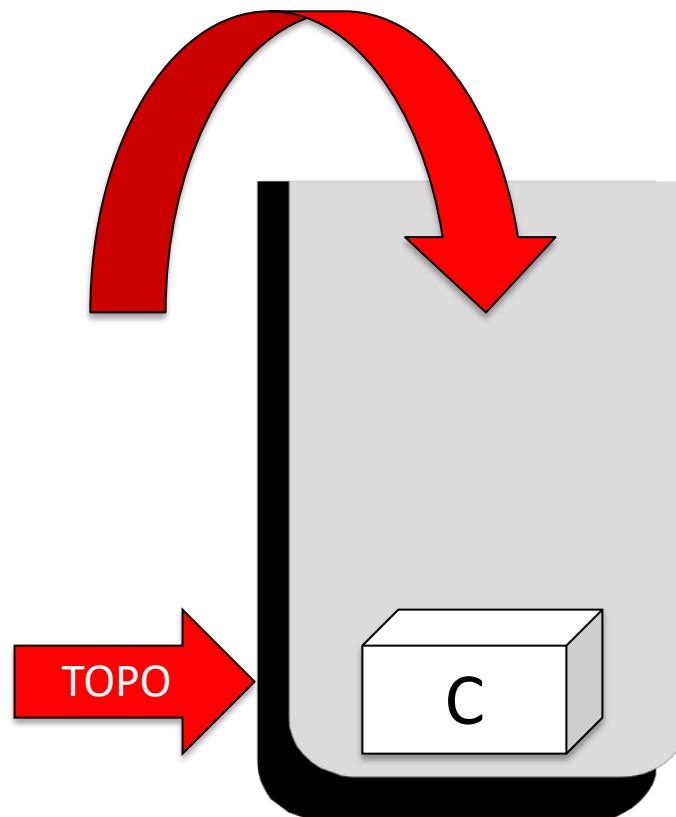
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item
- retirar (desempilhar) item



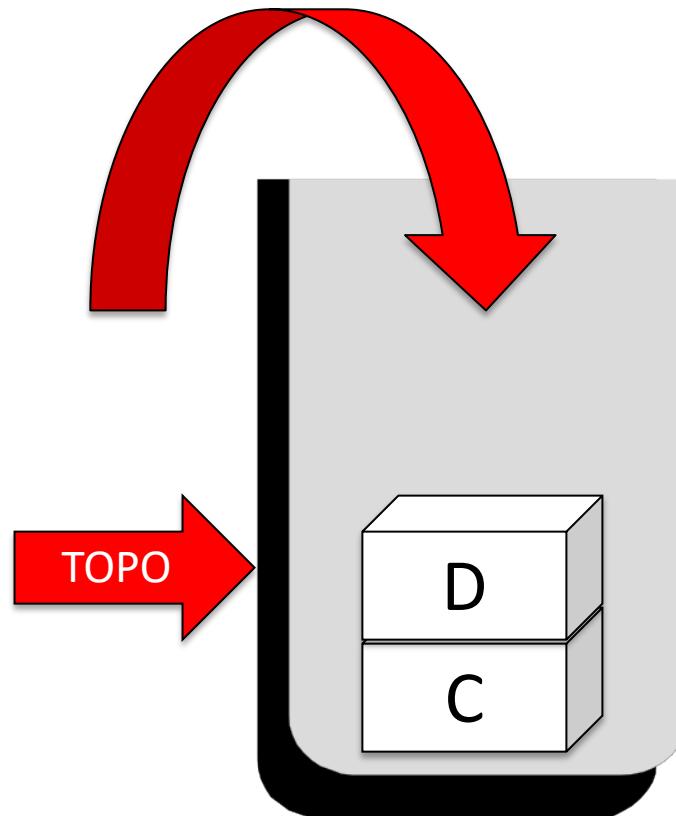
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item
- retirar (desempilhar) item
- inserir (empilhar) item C



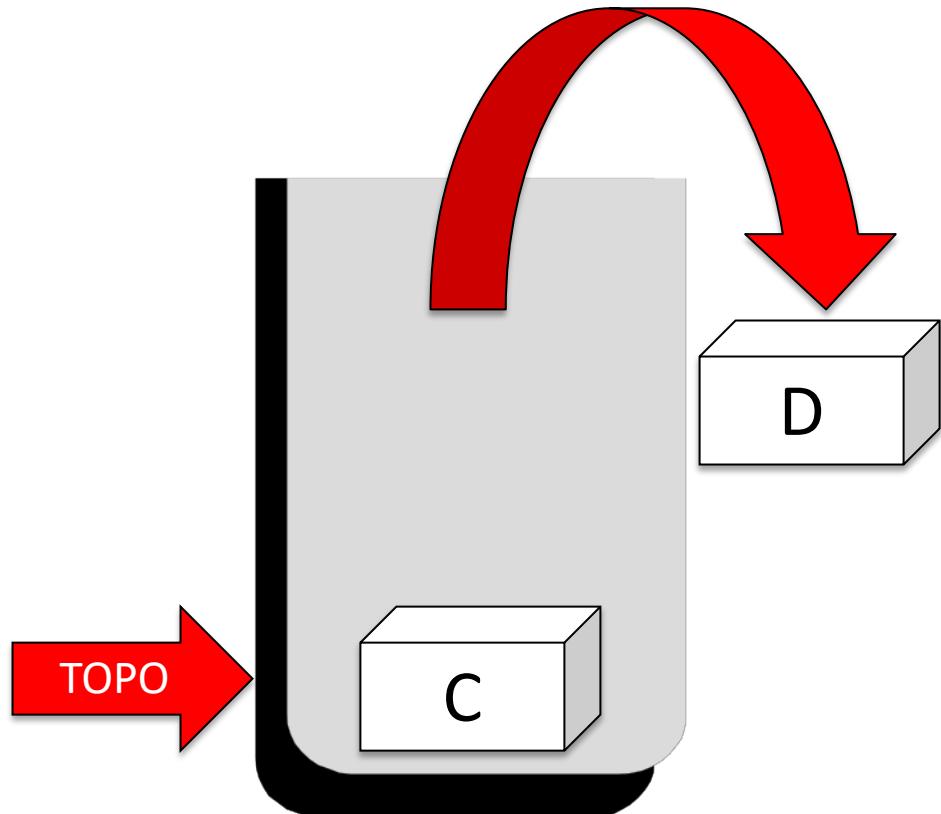
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item
- retirar (desempilhar) item
- inserir (empilhar) item C
- inserir (empilhar) item D



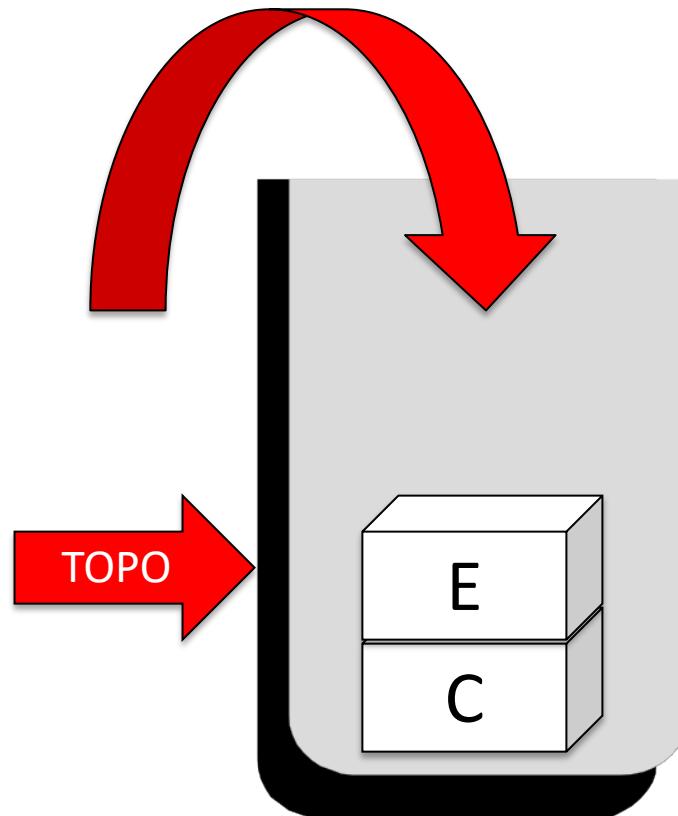
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item
- retirar (desempilhar) item
- inserir (empilhar) item C
- inserir (empilhar) item D
- retirar (desempilhar) item



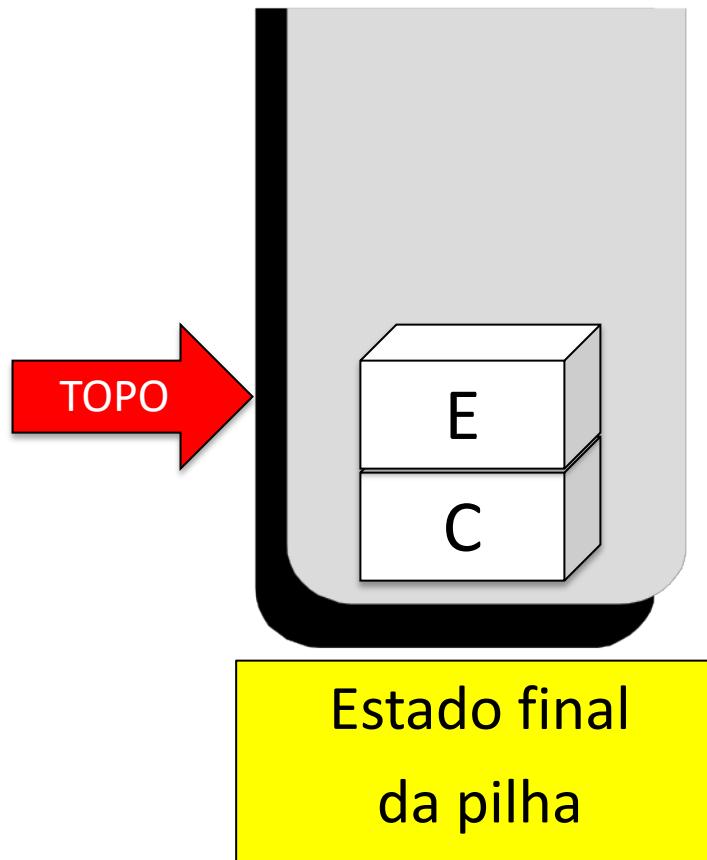
Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item
- retirar (desempilhar) item
- inserir (empilhar) item C
- inserir (empilhar) item D
- retirar (desempilhar) item
- inserir (empilhar) item E



Exemplo

- pilha inicialmente vazia
- inserir (empilhar) item A
- inserir (empilhar) item B
- retirar (desempilhar) item
- retirar (desempilhar) item
- inserir (empilhar) item C
- inserir (empilhar) item D
- retirar (desempilhar) item
- inserir (empilhar) itemE



- É ideal para estruturas aninhadas de profundidade imprevisível, por exemplo:
 - o controle de sequências de chamadas de subprogramas
 - a sintaxe de expressões aritméticas
 - ...
- As pilhas ocorrem em estruturas de natureza recursiva
 - Elas são utilizadas para implementar a **recursividade**

■ Conjunto de Operações

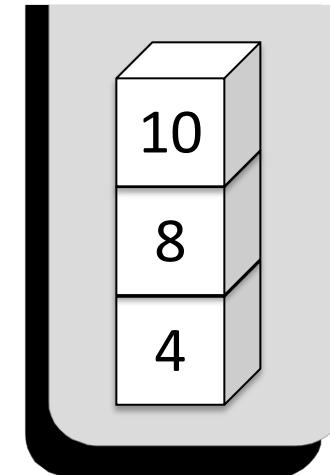
- **TPilha_Inicia(Pilha)**: Inicia uma pilha vazia
- **TPilha_EhVazia(Pilha)**: Retorna *true* se a pilha estiver vazia; caso contrário, retorna *false*
- **TPilha_Empilha(Pilha, x)**: Empilha o item *x* no topo da pilha
- **TPilha_Desempilha(Pilha, x)**: Desempilha o item *x* do topo da pilha, retirando-o da pilha
- **TPilha_Tamanho(Pilha)**: Retorna o número de itens da pilha

- Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas
- As duas representações mais utilizadas são:
 - Implementação por meio de **arranjos**
 - Implementação por meio de **apontadores**

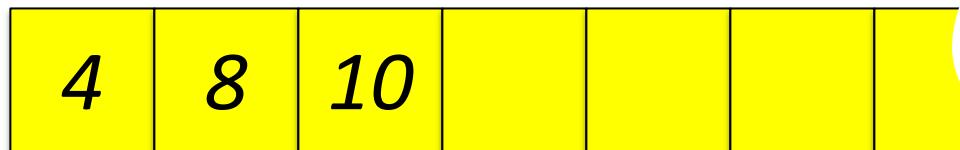
IMPLEMENTAÇÃO POR ARRANJOS

Implementação por Arranjos

- Os itens da pilha são armazenados no **início de um arranjo**, como mostra este exemplo



[1] [2] [3] [4] [5] [6] ...

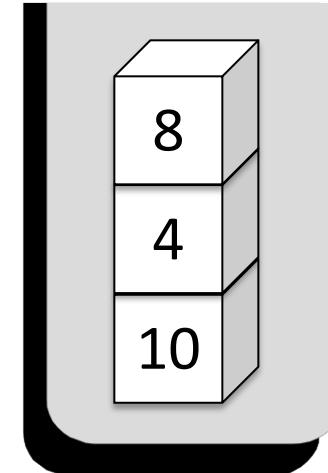


Um arranjo de inteiros

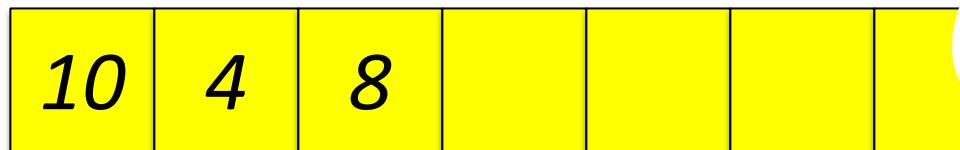
Não nos interessamos para o que está armazenado nesta parte do arranjo

Implementação por Arranjos

- Os itens possuem uma ordem.
A figura abaixo **não** representa a mesma pilha que a anterior ...



[1] [2] [3] [4] [5] [6] ...

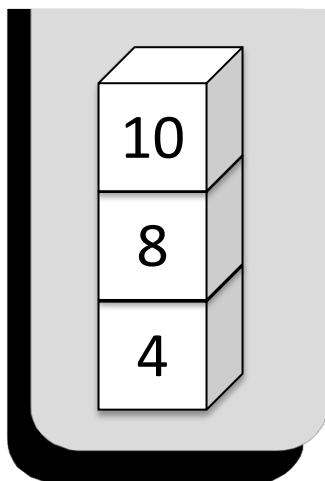


Um arranjo de inteiros

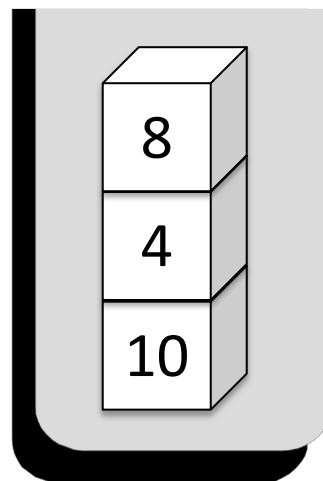
Não nos interessamos para o que está armazenado nesta parte do arranjo

Implementação por Arranjos

- Assim, ...



\neq



Implementação por Arranjos

- Nós precisamos também armazenar o **Topo** da pilha

3

Topo

[1] [2] [3] [4] [5] [6] ...

4

8

10

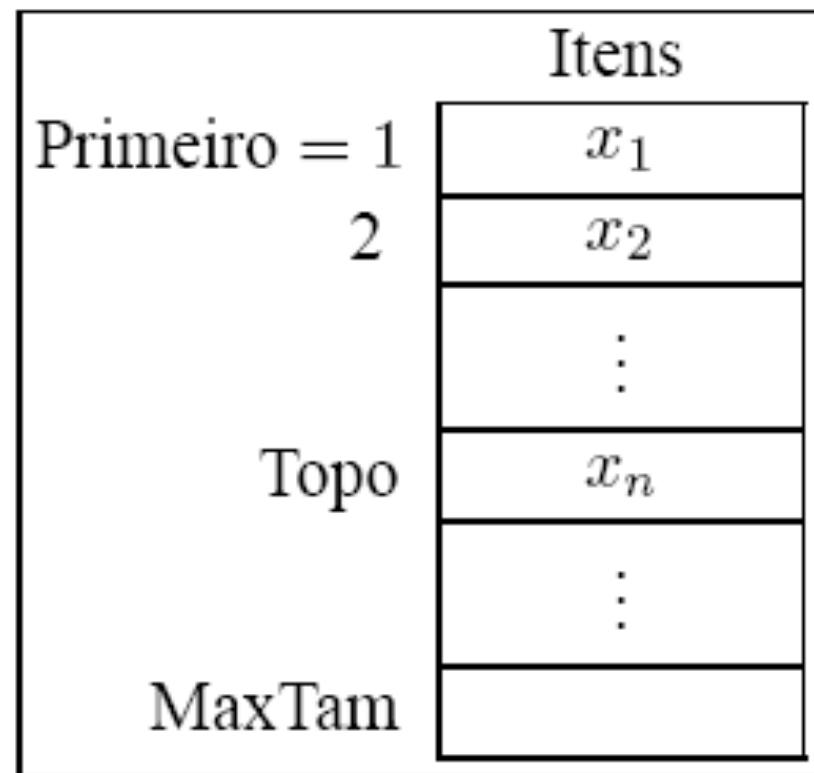
Um arranjo de inteiros



Não nos interessamos para o que está armazenado nesta parte do arranjo

Estrutura da Pilha por Arranjos

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



Estrutura da Pilha por Arranjos

- Os itens são armazenados em um **arranjo** de tamanho suficiente para armazenar a pilha
- O campo **Topo** mantém um apontador para o item no topo da pilha
- A constante **MaxTam** define o tamanho máximo permitido para a pilha

Estrutura da Pilha por Arranjos

```
#define MAXTAM 1000

typedef int TChave;

typedef int TA pontador;

typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

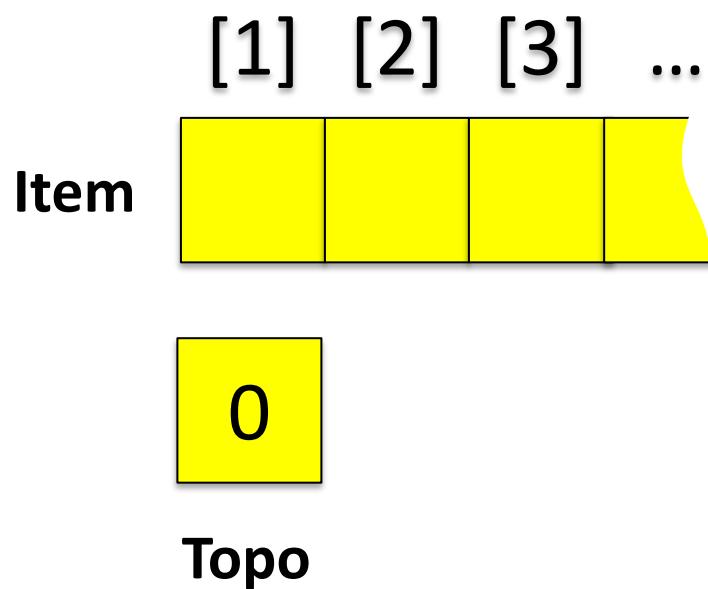
typedef struct {
    TItem Item[MAXTAM];
    TA pontador Topo;
} TPilha;
```

Estrutura da Pilha por Arranjos

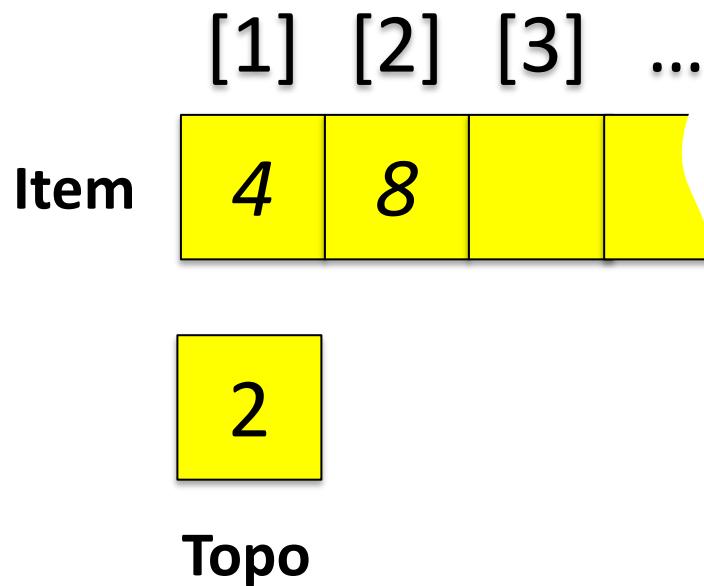
```
/* procedimentos e funcoes do TAD */

void TPilha_Inicia(TPilha *pPilha);
int TPilha_EhVazia(TPilha *pPilha);
int TPilha_Empilha(TPilha *pPilha, TItem x);
int TPilha_Desempilha(TPilha *pPilha, TItem *pX);
int TPilha_Tamanho(TPilha *pPilha);
```

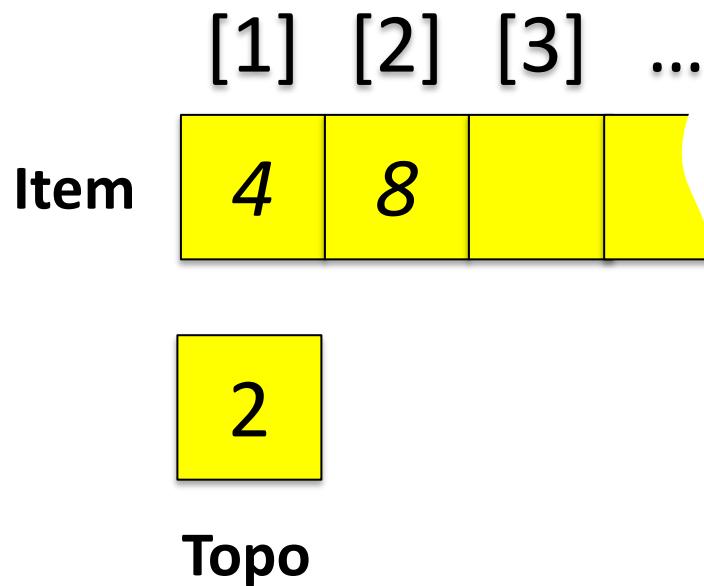
Criar Pilha Vazia



Inserção de Elementos na Pilha

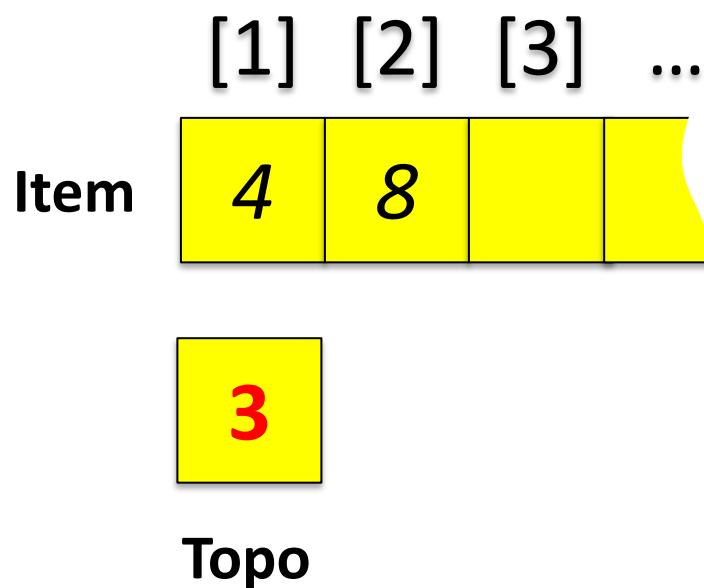


Inserção de Elementos na Pilha

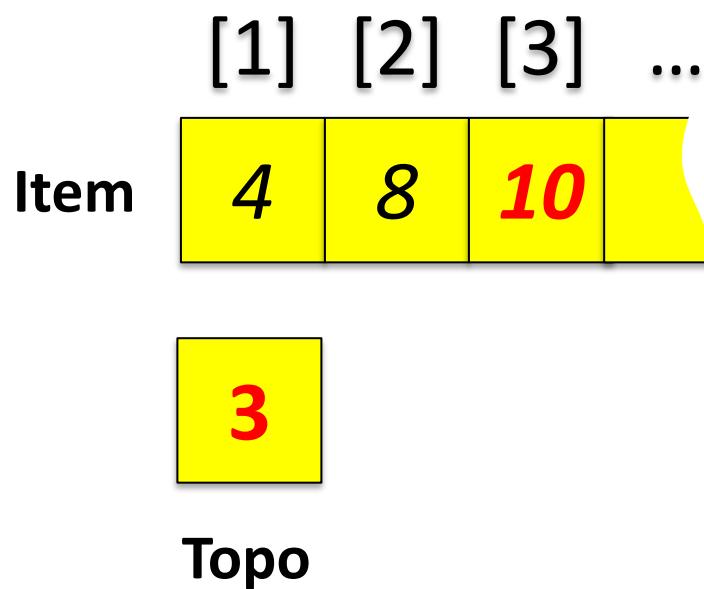


A pilha está cheia?

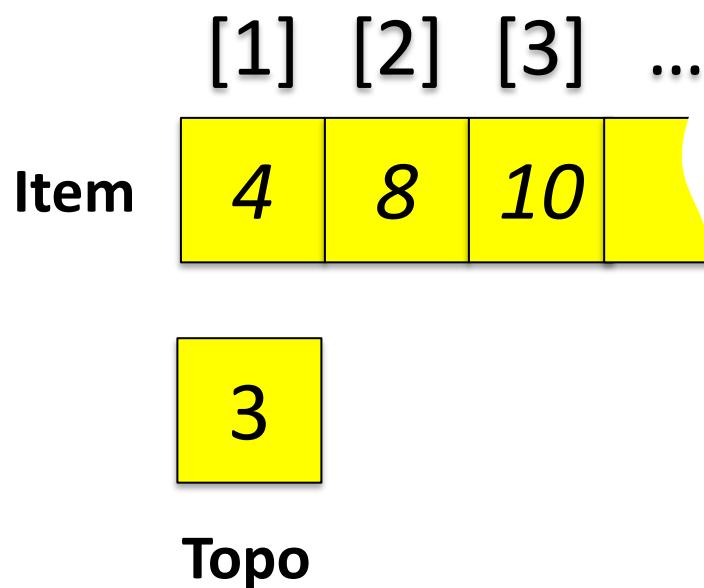
Inserção de Elementos na Pilha



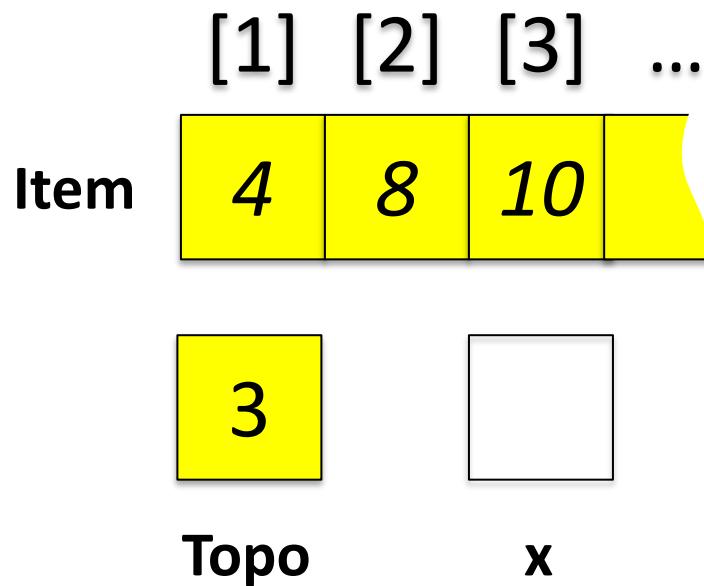
Inserção de Elementos na Pilha



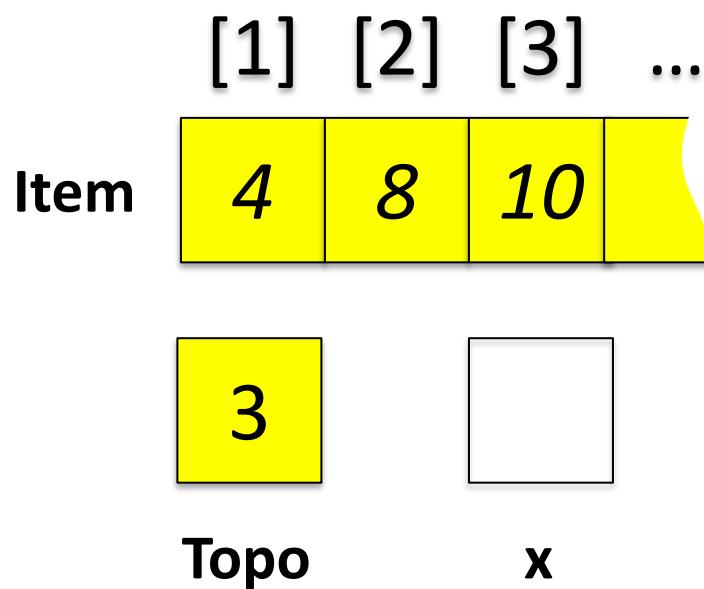
Inserção de Elementos na Pilha



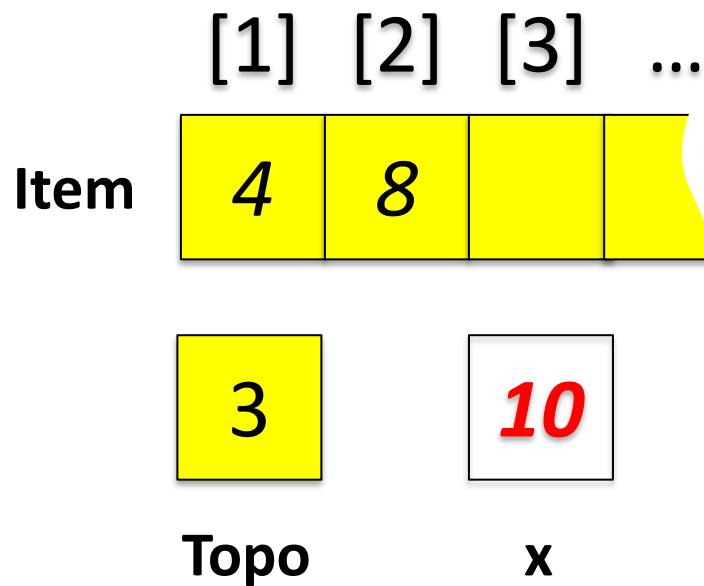
Retirada de Elementos da Pilha



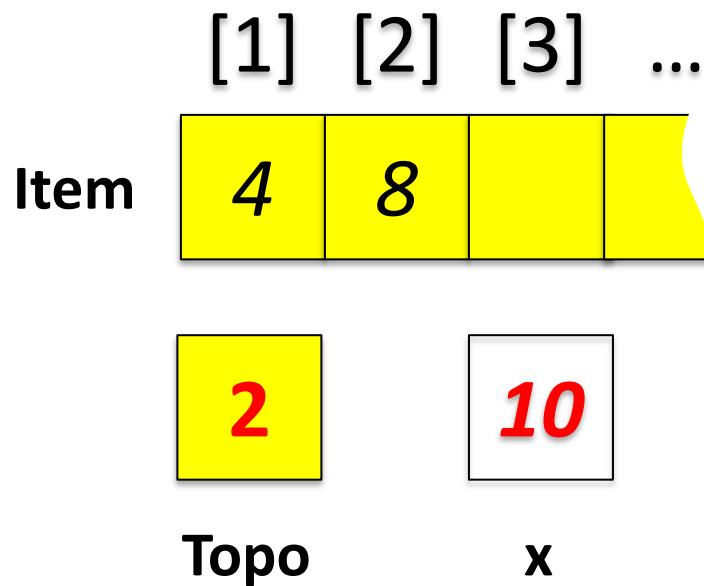
Retirada de Elementos da Pilha



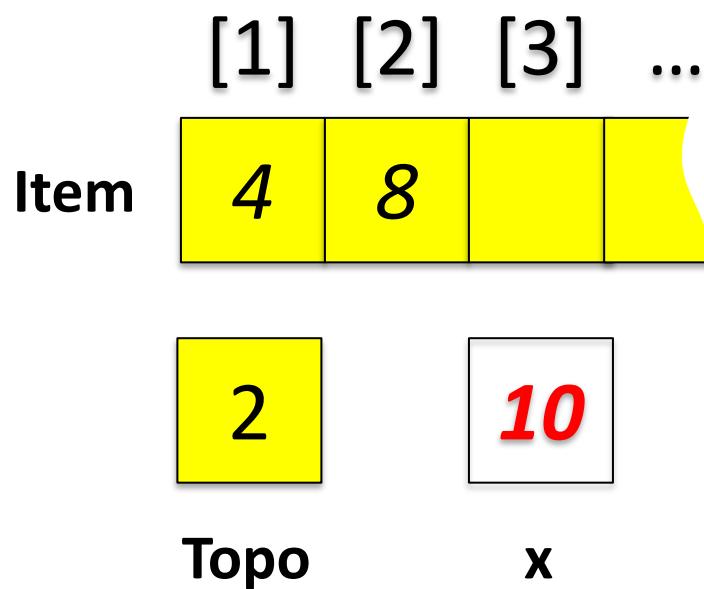
Retirada de Elementos da Pilha



Retirada de Elementos da Pilha



Retirada de Elementos da Pilha



Operações na Pilha por Arranjos

```
void TPilha_Inicia(TPilha *pPilha)
{
    pPilha->Topo = 0;
} /* TPilha_Inicia */

int TPilha_EhVazia(TPilha *pPilha)
{
    return (pPilha->Topo == 0);
} /* TPilha_EhVazia */

int TPilha_Empilha(TPilha *pPilha, TItem x)
{
    if (pPilha->Topo == MAXTAM)
        return 0; /* pilha cheia */

    pPilha->Item[pPilha->Topo] = x;
    pPilha->Topo++;
    return 1;
} /* TPilha_Empilha */
```

Operações na Pilha por Arranjos

```
int TPilha_Desempilha(TPilha *pPilha, TItem *pX)
{
    if (TPilha_EhVazia(pPilha))
        return 0;

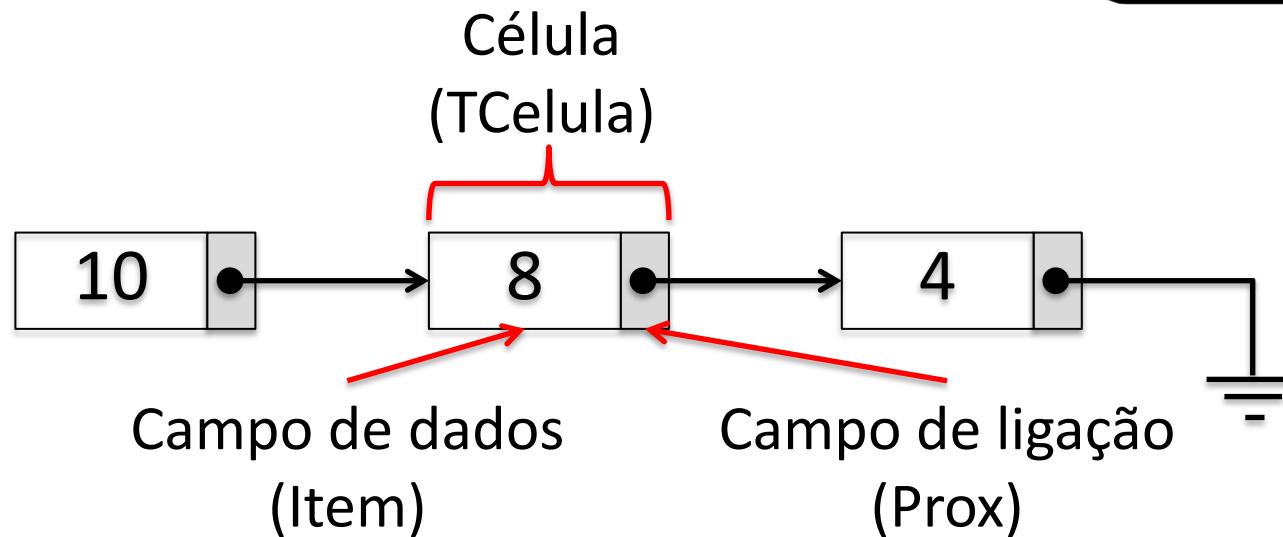
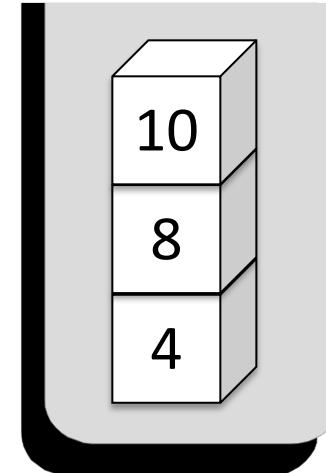
    pPilha->Topo--;
    *pX = pPilha->Item[pPilha->Topo];
    return 1;
} /* TPilha_Desempilha */

int TPilha_Tamanho(TPilha *pPilha)
{
    return (pPilha->Topo);
} /* TPilha_Tamanho */
```

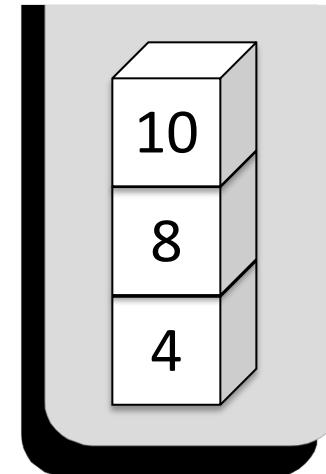
IMPLEMENTAÇÃO POR APONTADORES

Implementação por Apontadores

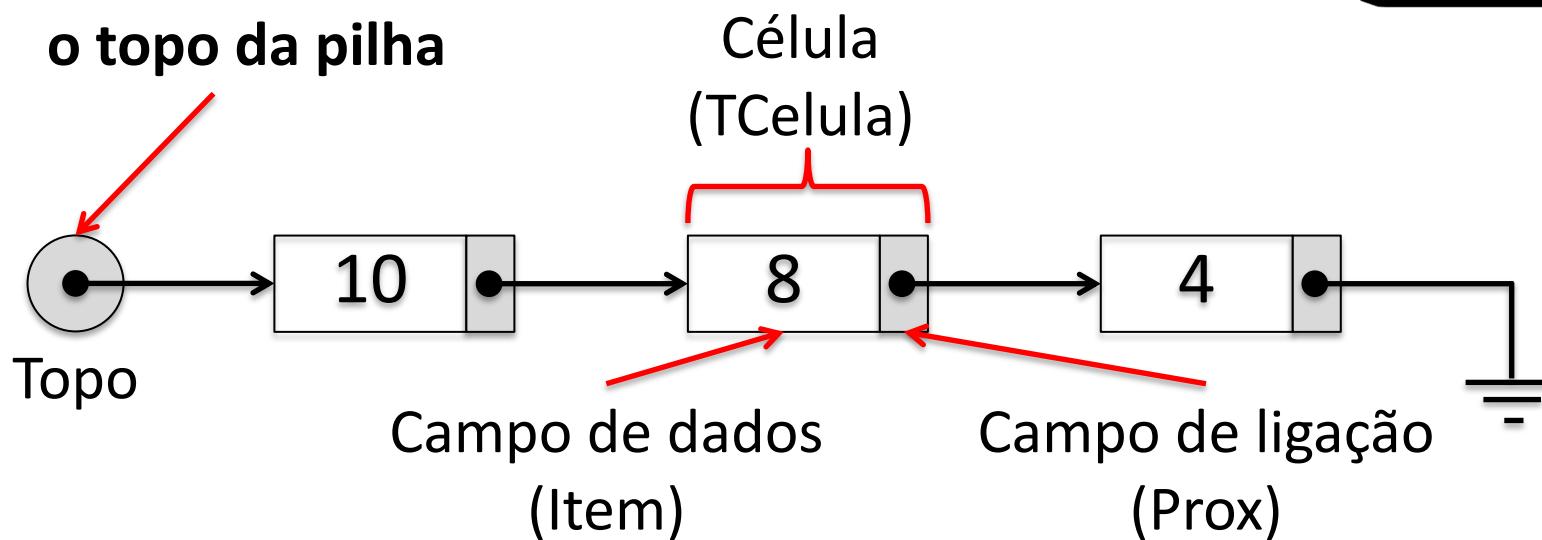
- Os itens são colocados em uma estrutura (**TCelula**) que contém um campo com um item (**Item**) e outro campo com um apontador para a próxima célula (**Prox**)



- Nós precisamos também armazenar o **Topo** da pilha

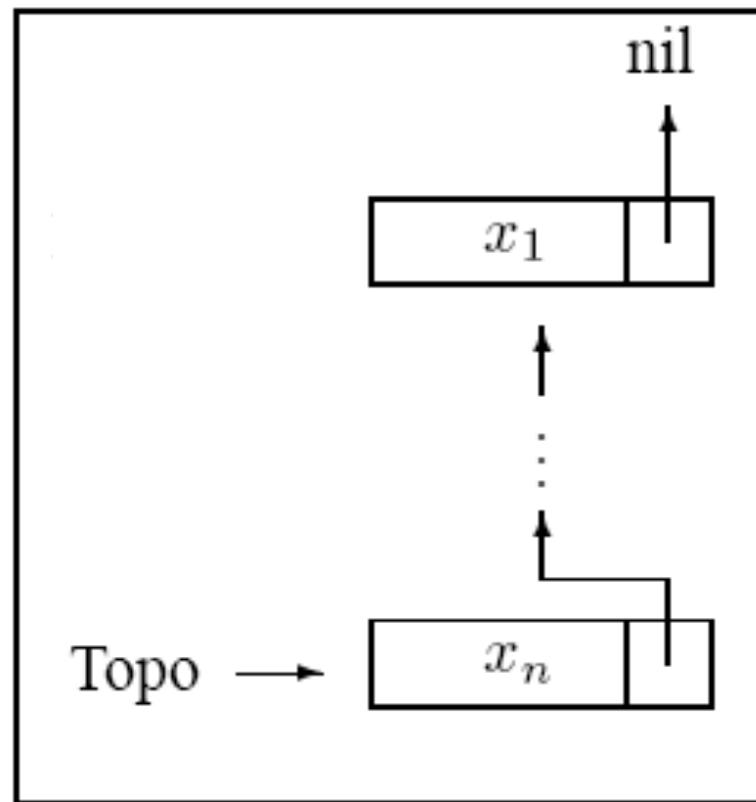


Um apontador armazena
o topo da pilha



Estrutura da Pilha por Apontador

- Cada item é encadeado com o seguinte por meio de um apontador
- Permite utilizar posições não contíguas de memória



- O campo **Tamanho** evita a contagem do número de itens na função **TPilha_Tamanho**
- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula
- O registro **TPilha** contém um apontador para o **Topo** da pilha

Estrutura da Pilha por Apontador

```
typedef int TChave;

typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

typedef struct SCelula *TApontador;

typedef struct SCelula {
    TItem Item;
    TApontador Prox;
} TCelula;

typedef struct {
    TApontador Topo;
    int Tamanho;
} TPilha;
```

Cria Pilha Vazia

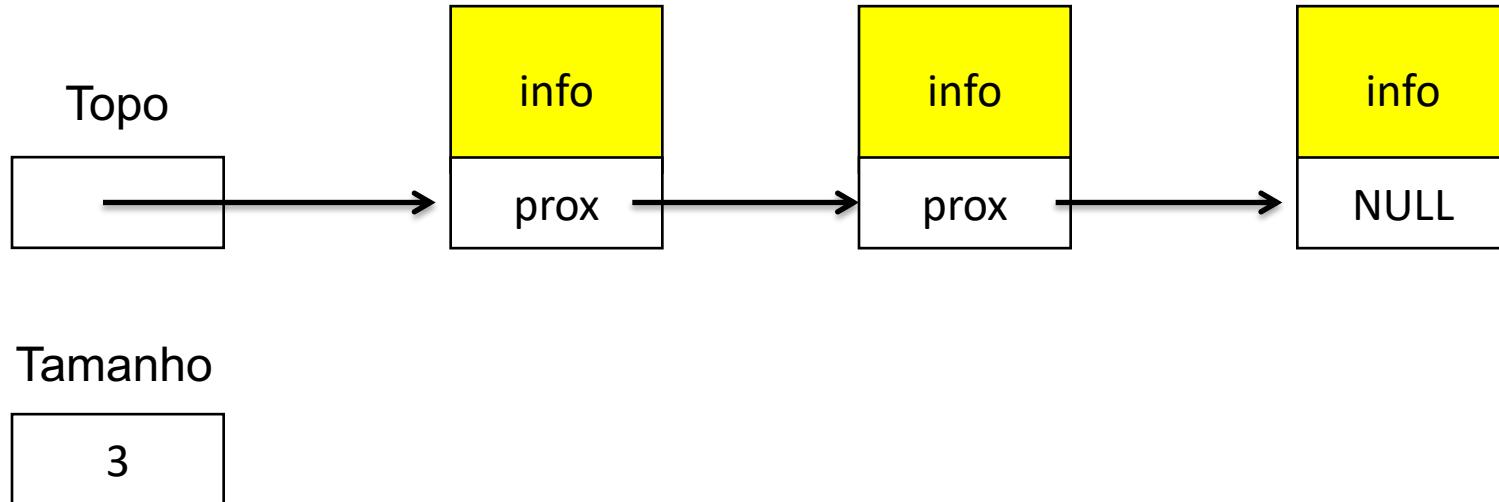
Topo

NULL

0

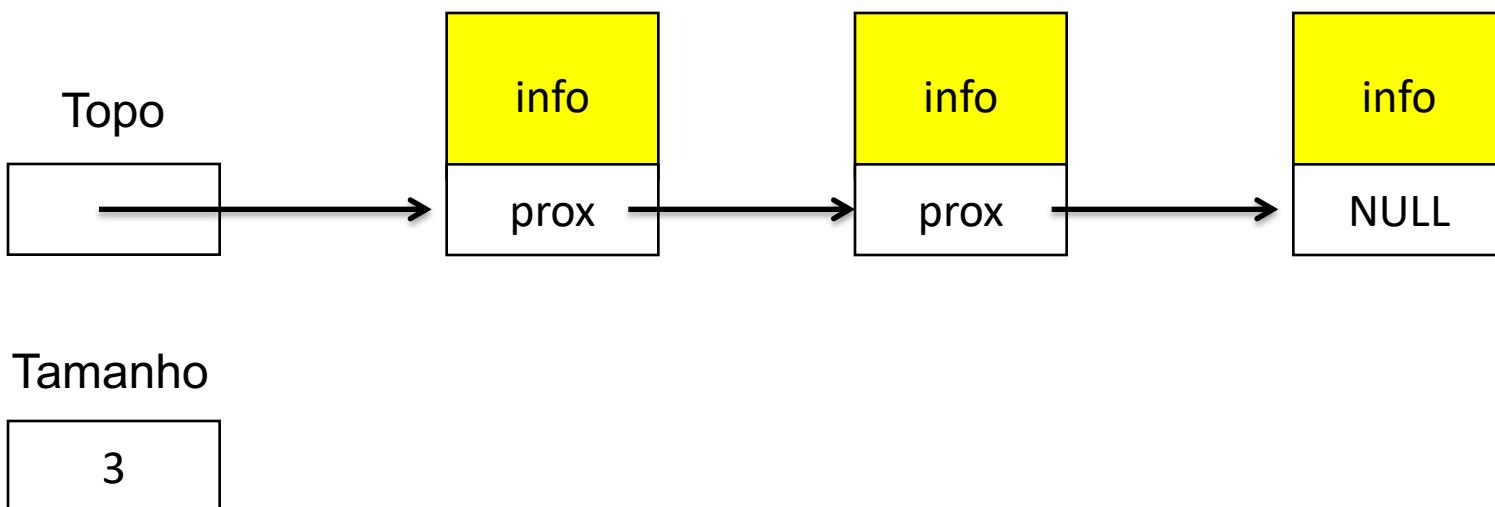
Tamanho

Inserção de Elementos na Pilha



- Opção única de posição onde pode inserir:
 - Topo da pilha, ou seja, 1^a posição

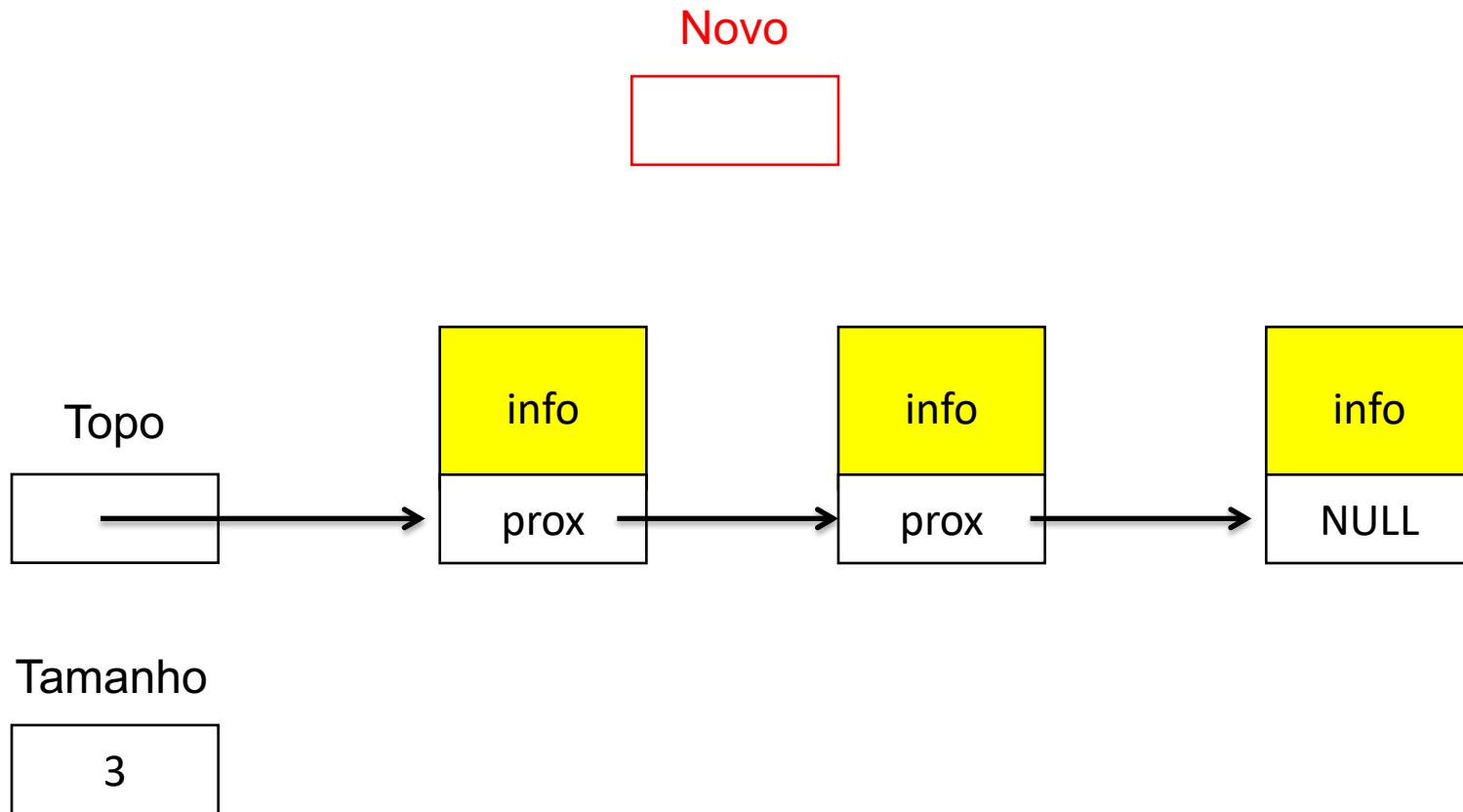
Inserção de Elementos na Pilha



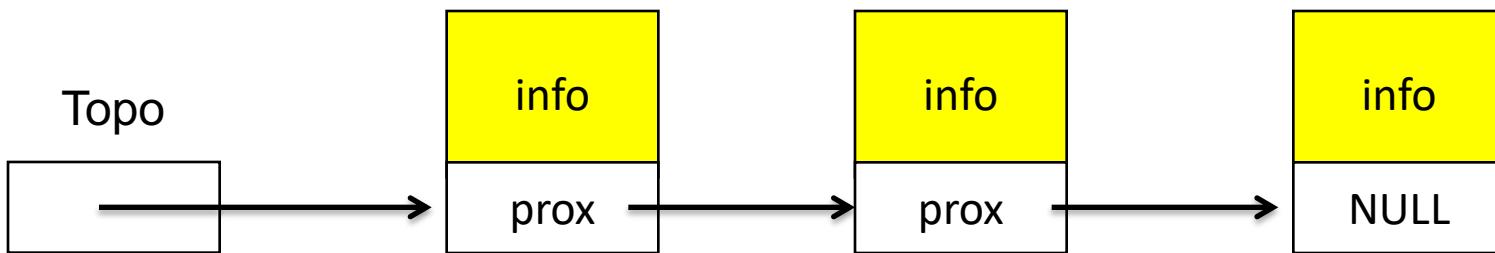
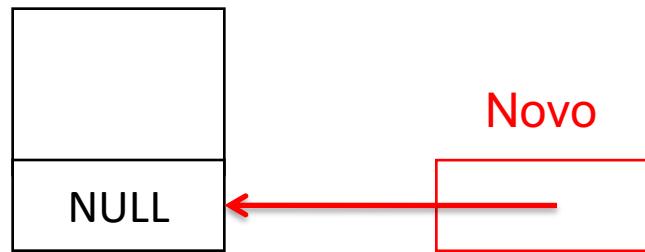
Tamanho

3

Inserção de Elementos na Pilha



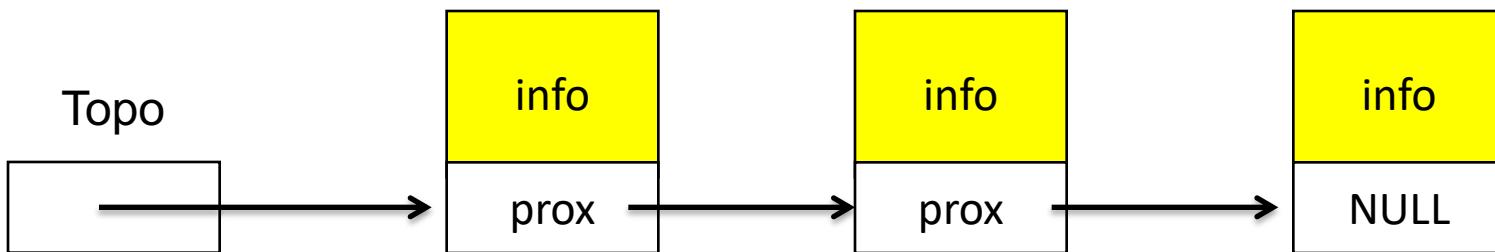
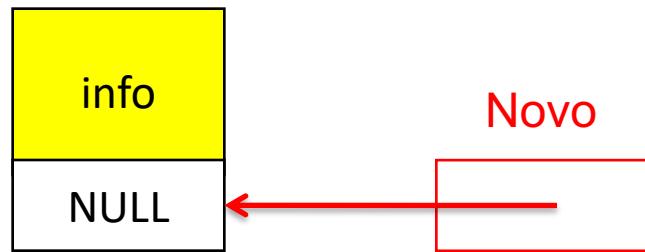
Inserção de Elementos na Pilha



Tamanho

3

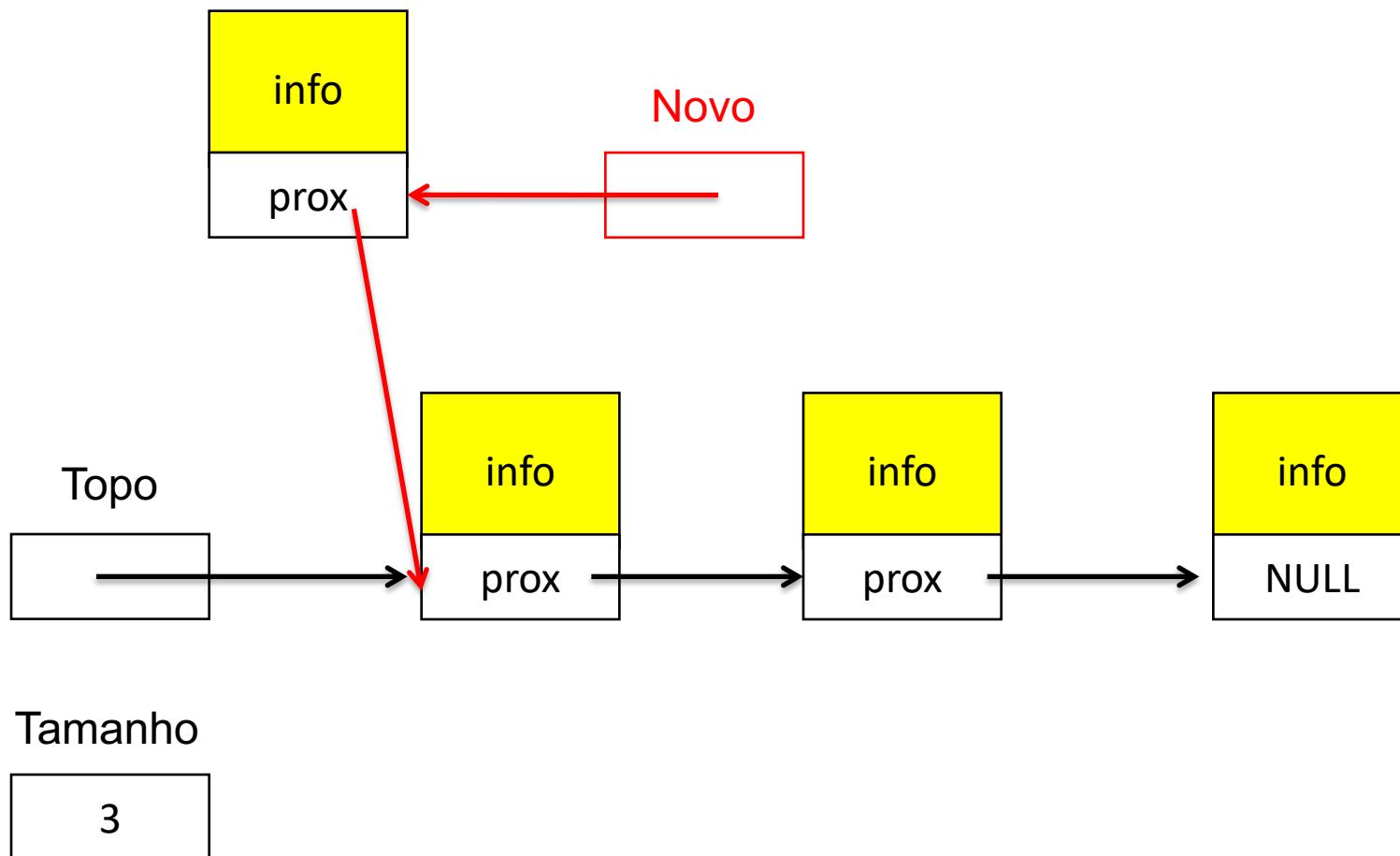
Inserção de Elementos na Pilha



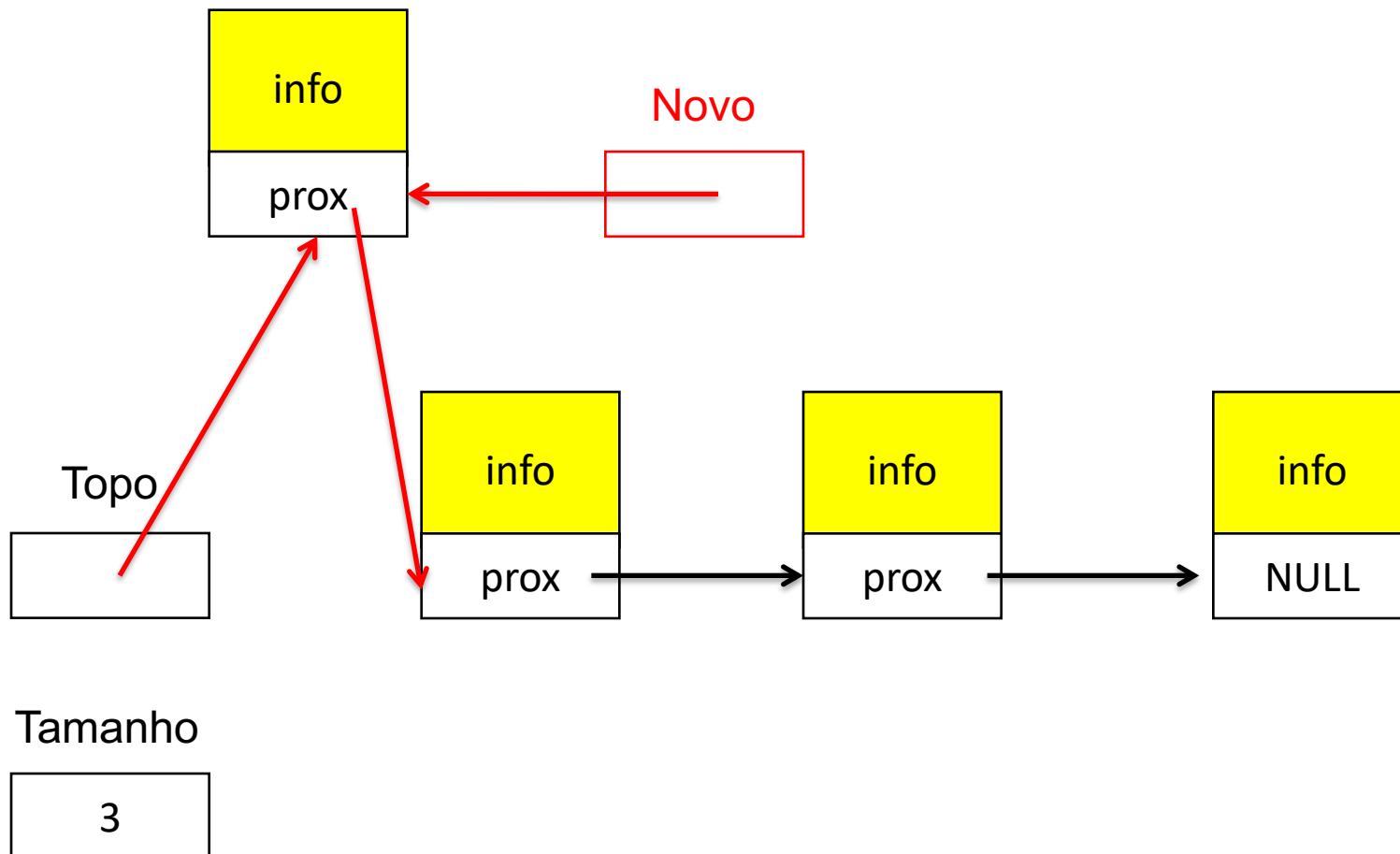
Tamanho

3

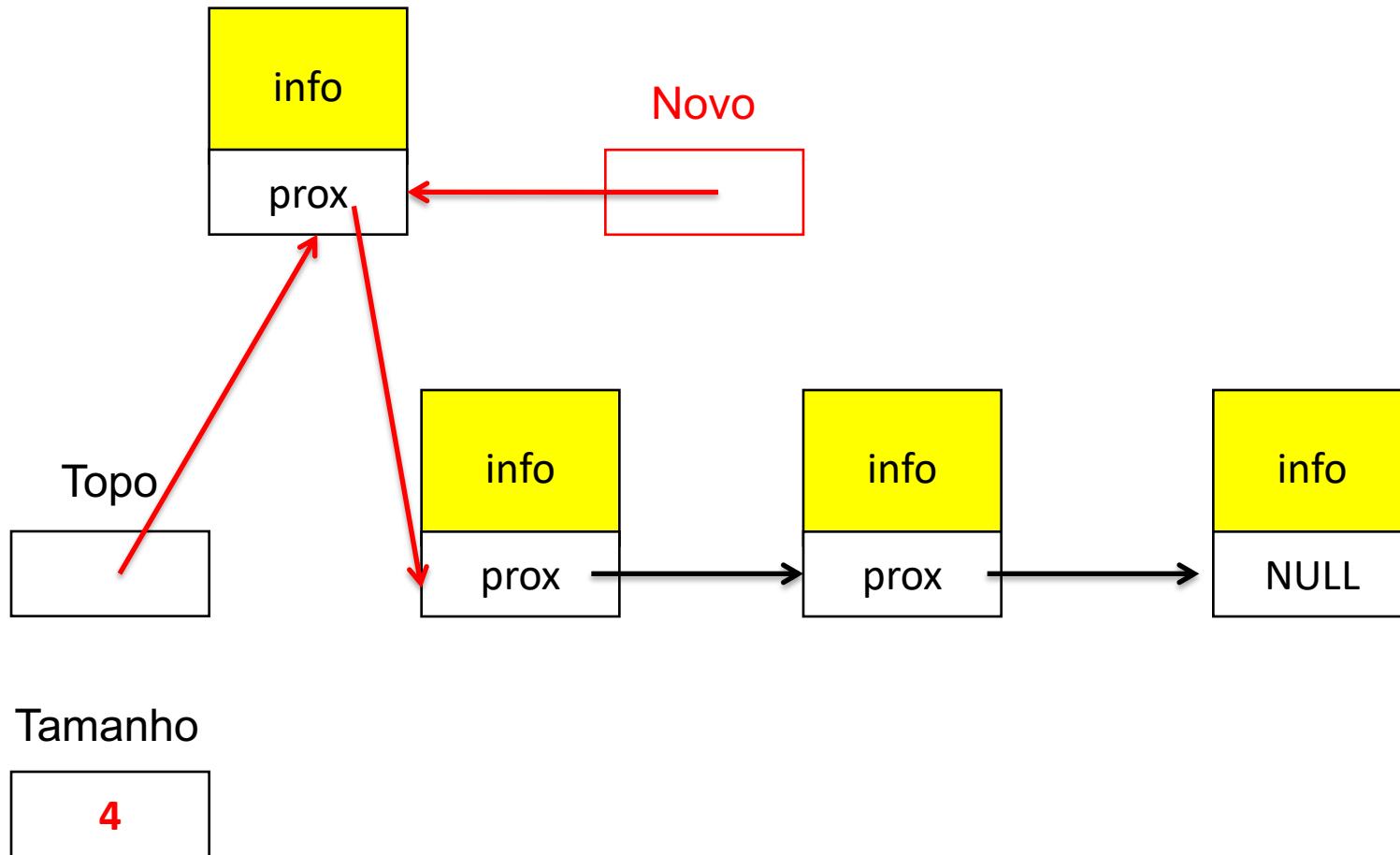
Inserção de Elementos na Pilha



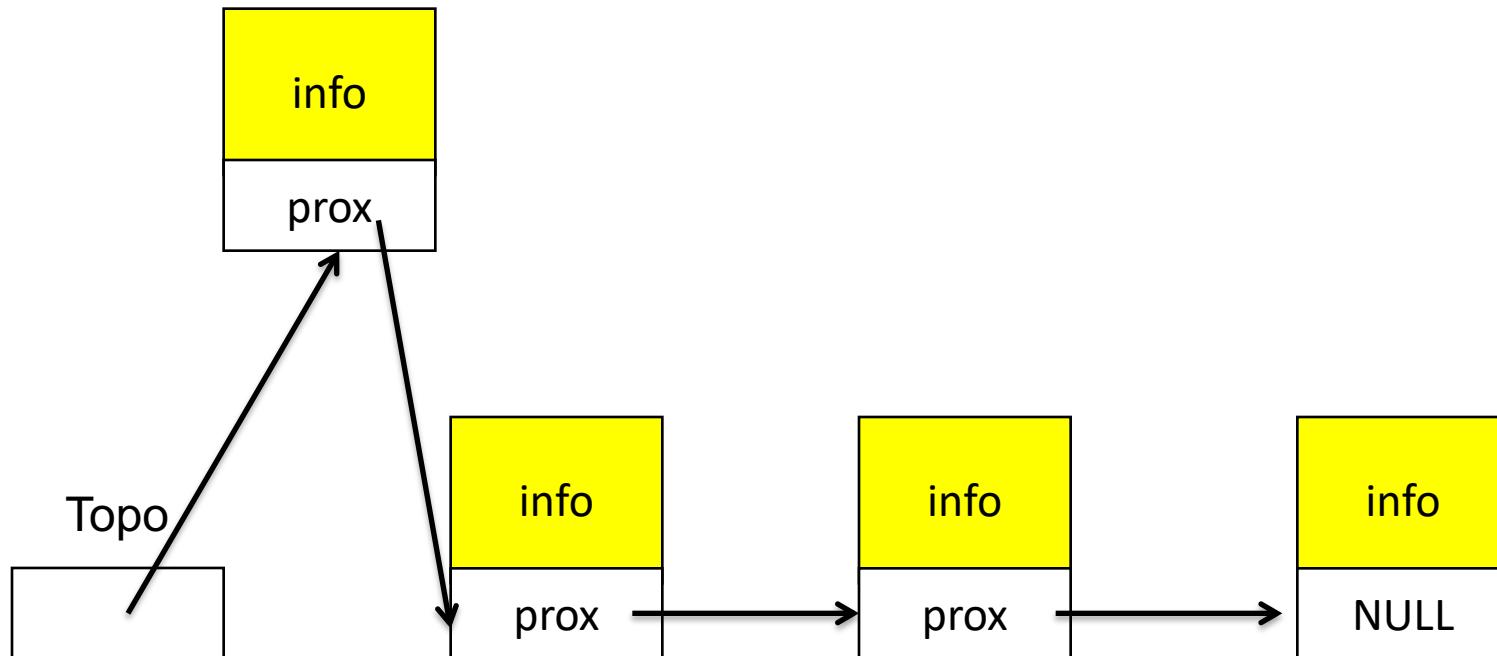
Inserção de Elementos na Pilha



Inserção de Elementos na Pilha



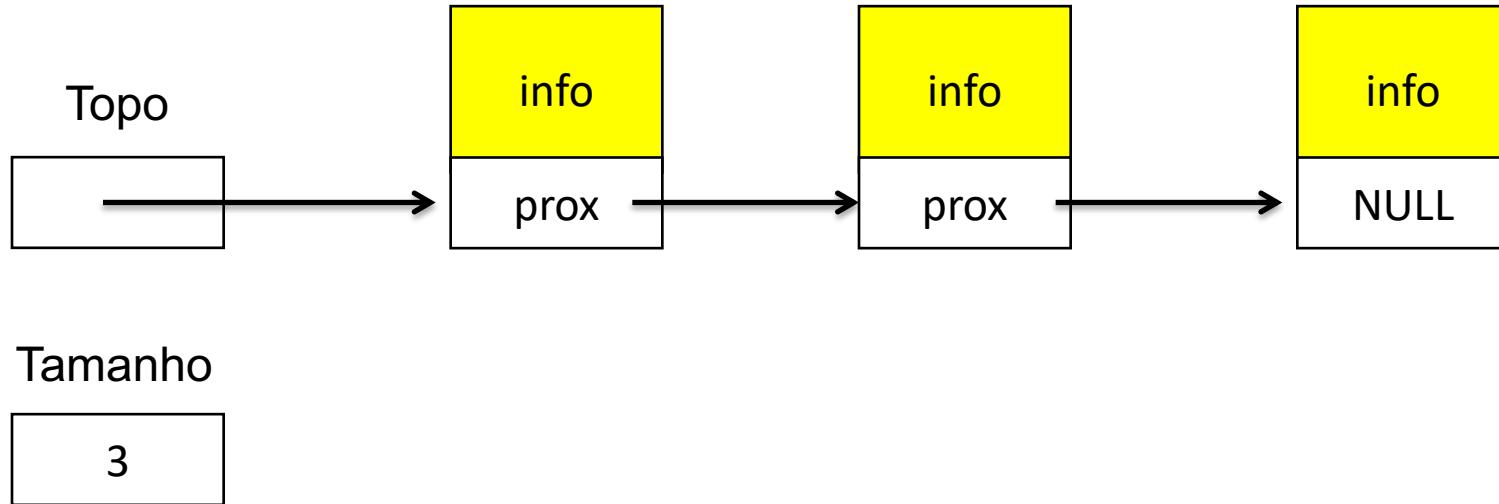
Inserção de Elementos na Pilha



Tamanho

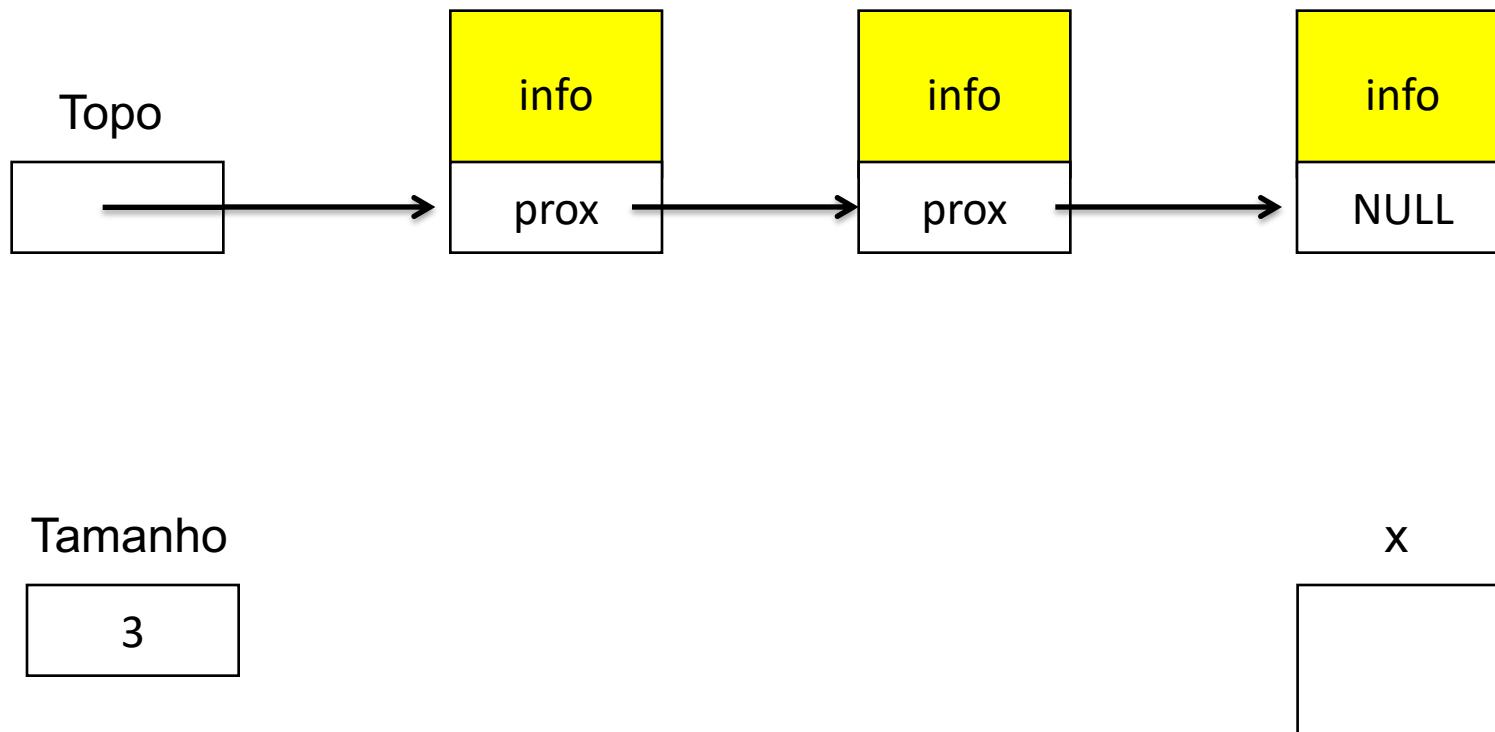
4

Retirada de Elementos da Pilha

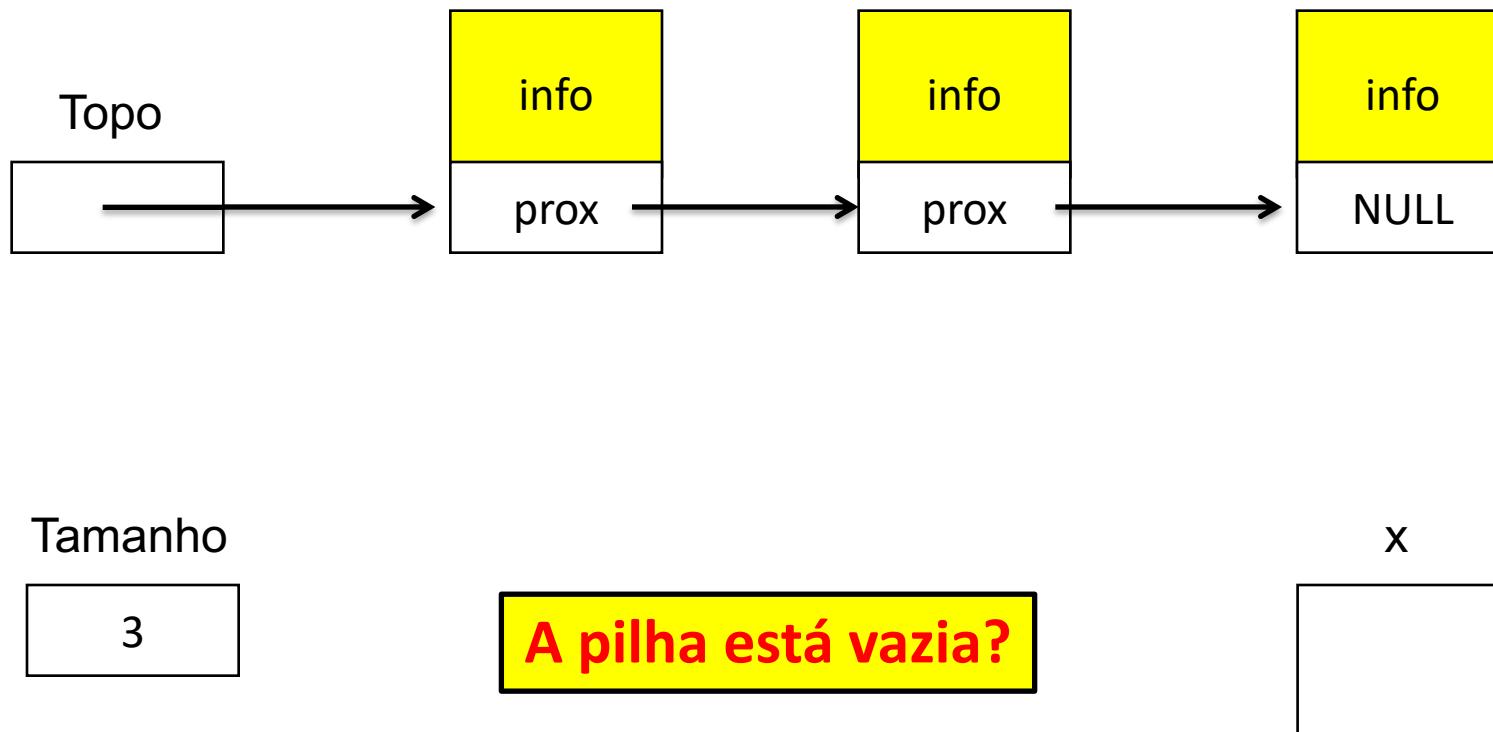


- Opção única de posição onde pode retirar:
 - Topo da pilha, ou seja, 1^a posição

Retirada de Elementos da Pilha



Retirada de Elementos da Pilha

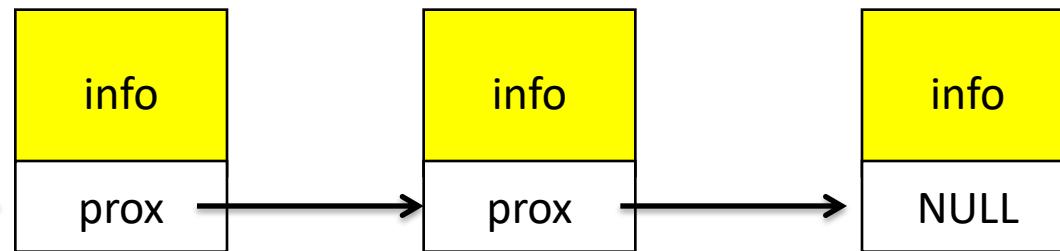
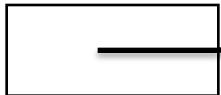


Retirada de Elementos da Pilha

Aux



Topo



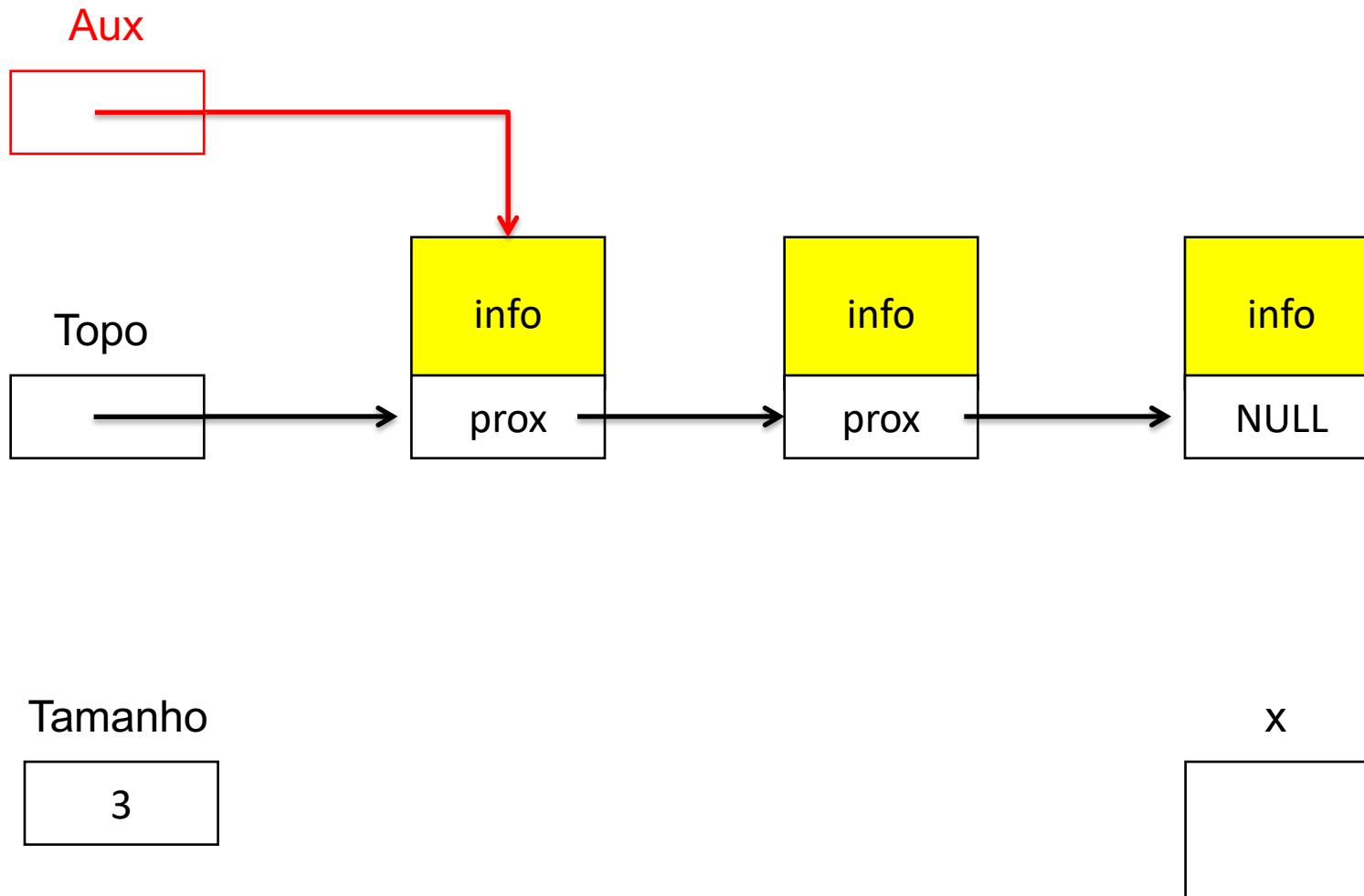
Tamanho



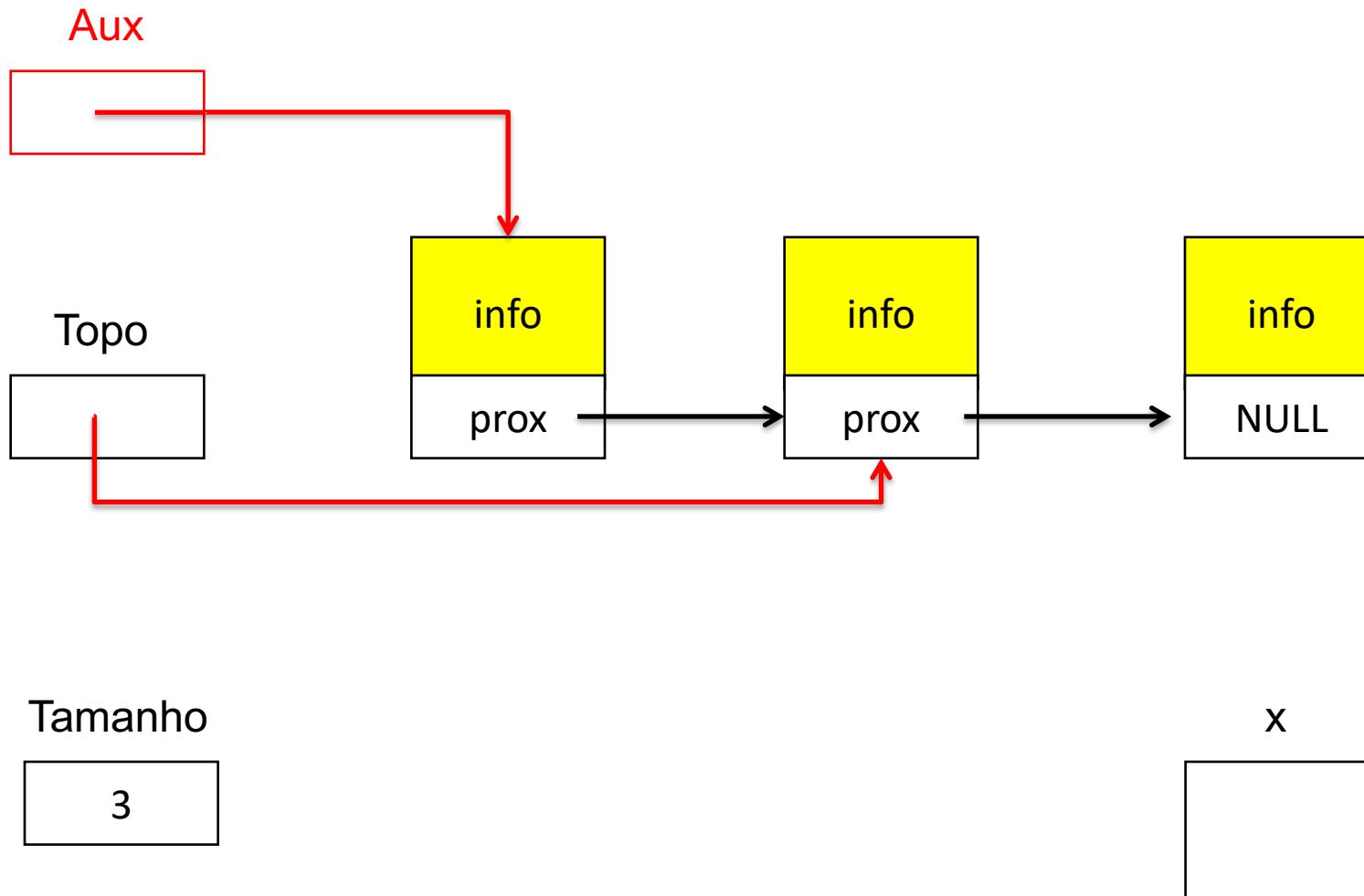
x



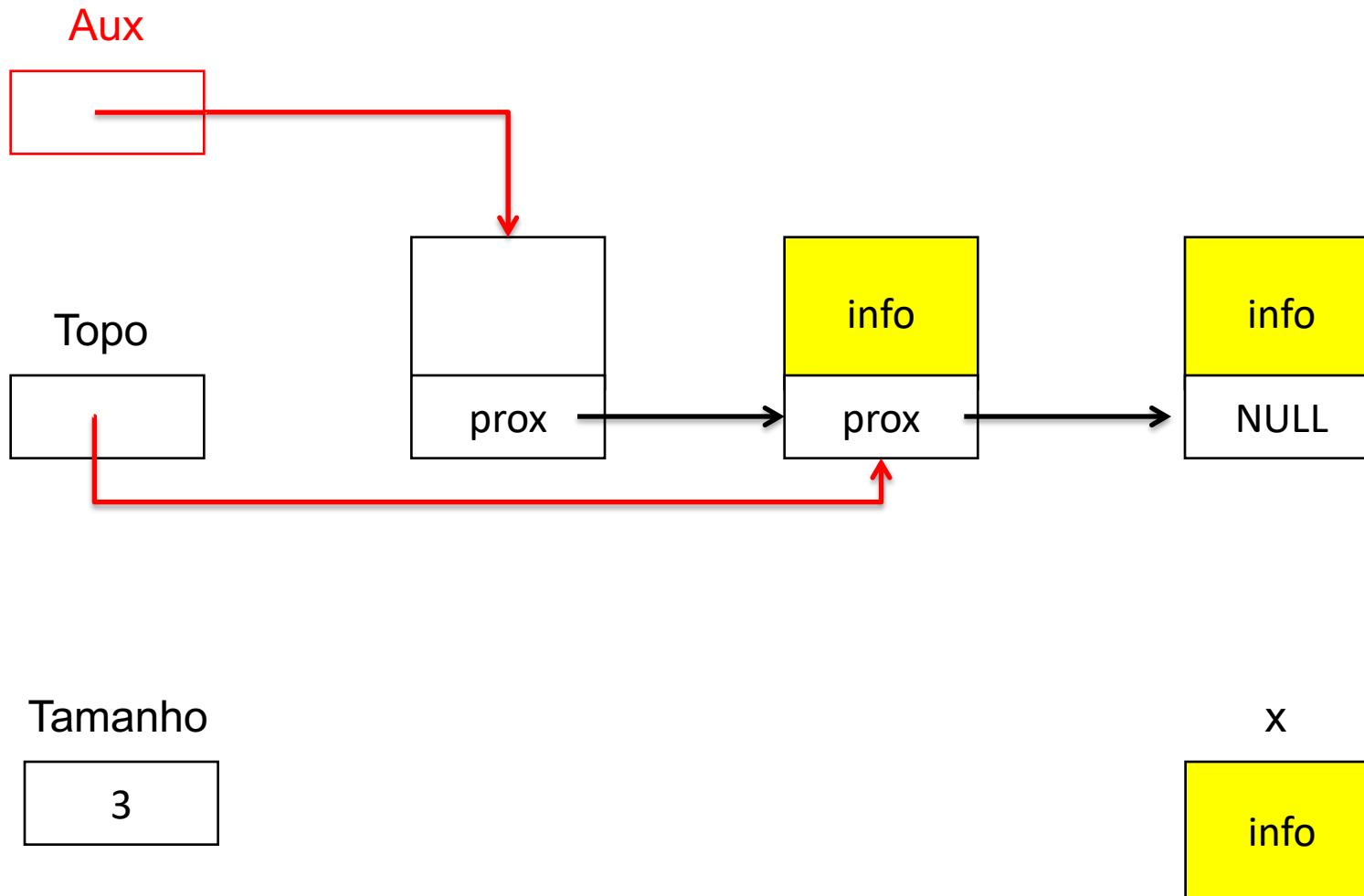
Retirada de Elementos da Pilha



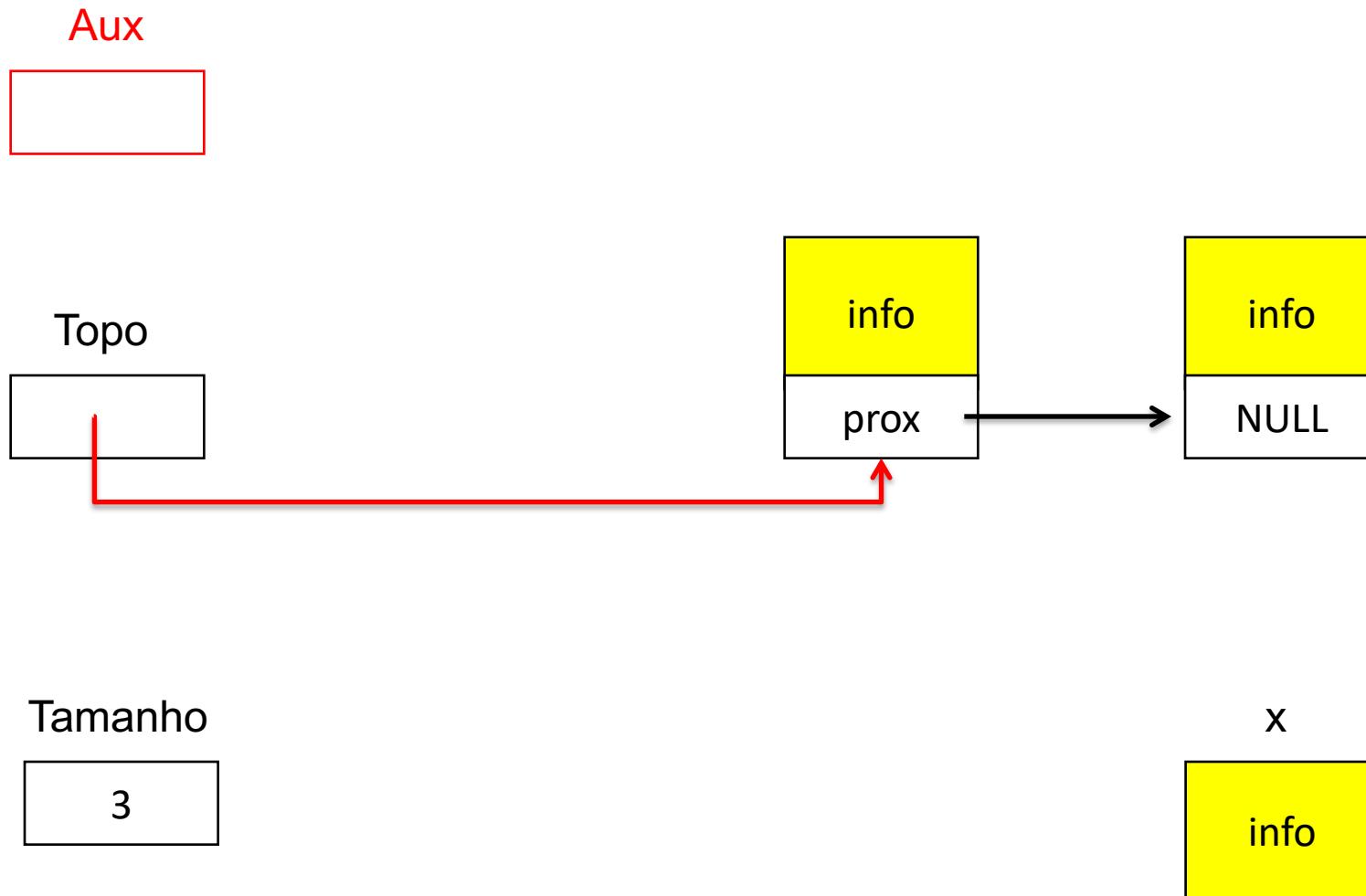
Retirada de Elementos da Pilha



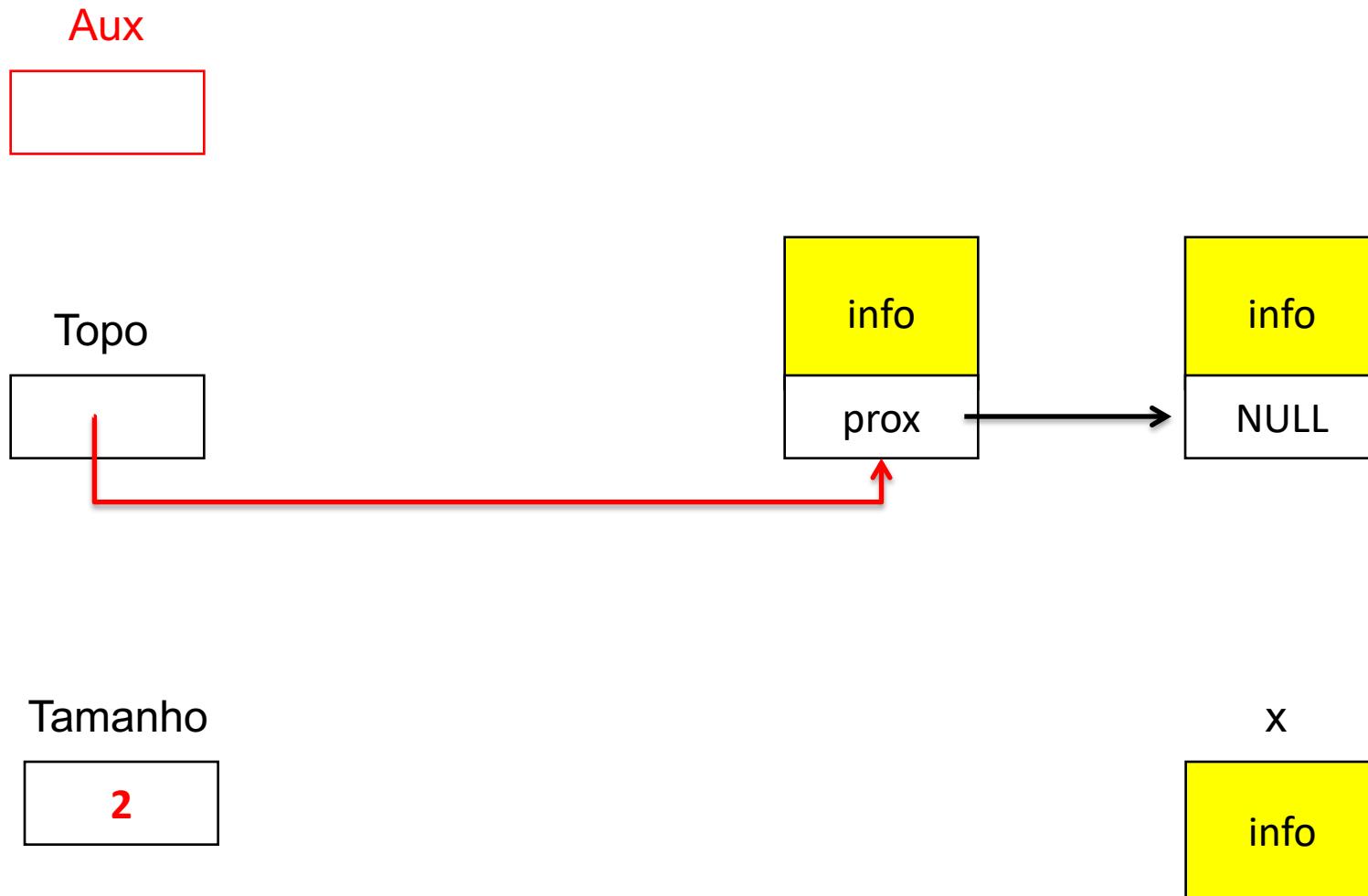
Retirada de Elementos da Pilha



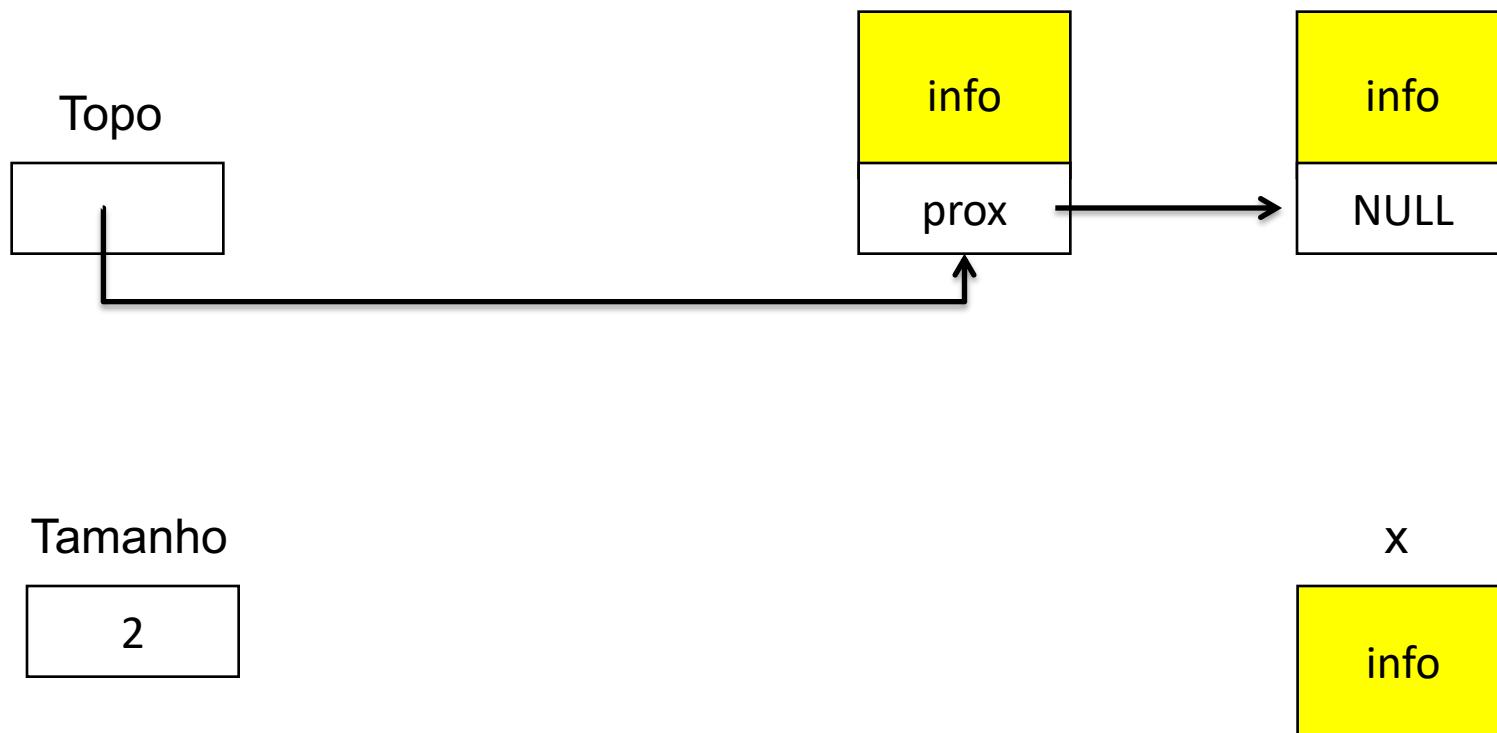
Retirada de Elementos da Pilha



Retirada de Elementos da Pilha



Retirada de Elementos da Pilha



Operações em Pilhas

```
/* Inicia as variaveis da pilha */
void TPilha_Inicia(TPilha *pPilha)
{
    pPilha->Topo = NULL;
    pPilha->Tamanho = 0;
}

/* Retorna se a pilha esta vazia */
int TPilha_EhVazia(TPilha *pPilha)
{
    return (pPilha->Topo == NULL);
}

/* Retorna o tamanho da pilha */
int TPilha_Tamanho(TPilha *pPilha)
{
    return (pPilha->Tamanho);
}
```

Operações em Pilhas

```
/* Empilha um item no topo da pilha */
int TPilha_Empilha(TPilha *pPilha, TItem x)
{
    TApontador pNovo;

    pNovo = (TApontador) malloc(sizeof(TCelula));
    pNovo->Item = x;
    pNovo->Prox = pPilha->Topo;
    pPilha->Topo = pNovo;
    pPilha->Tamanho++;
    return 1;
}
```

Operações em Pilhas

```
/* Desempilha o item do topo da pilha */
int TPilha_Desempilha(TPilha *pPilha, TItem *pX)
{
    TA pontador pAux;

    if (TPilha_EhVazia(pPilha))
        return 0;

    pAux = pPilha->Topo;
    pPilha->Topo = pAux->Prox;
    *pX = pAux->Item;
    free(pAux);
    pPilha->Tamanho--;

    return 1;
}
```

Exemplo de Aplicação

- Problema de balanceamento de parênteses

([] (([[]]))) [()])

A sequência está correta!

Exemplo de Aplicação

■ Problema de balanceamento de parênteses

Entrada	Símbolo	Ação	Pilha	Balanceado
([] [()])	(Empilha	(SIM
[] [()])	[Empilha	[(SIM
] [()])]	Desempilha, verifica compatibilidade	(SIM
[()])	[Empilha	[(SIM
()])	(Empilha	([(SIM
)]))	Desempilha, verifica compatibilidade	[(SIM
])]	Desempilha, verifica compatibilidade	(SIM
))	Desempilha, verifica compatibilidade	Ø	SIM
		Esvazia pilha	Ø	SIM

Exemplo de Aplicação

■ Problema de balanceamento de parênteses

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef char TItem;

int main(void)
{
    char *cadeia = "([ ] ((( [ ] ))) [ () ]) ";
    if ( ParentesesBalanceados(cadeia) )
        printf("Correto\n");
    else
        printf("Incorreto\n");

    return 0;
}
```

Exemplo de Aplicação

■ Problema de balanceamento de parênteses

```
int ParentesesBalanceados(char *cadeia)
{
    int i, balanceada = 1;
    TPilha *pPilha;
    char c;

    pPilha = (TPilha *) malloc(sizeof(TPilha));
    TPilha_Inicia(pPilha);

    for (i = 0; balanceada && i < strlen(cadeia); i++) {
        switch (cadeia[i]) {
            case ')': if (TPilha_Desempilha(pPilha, &c) && (c != '(')) balanceada = 0; break;
            case ']': if (TPilha_Desempilha(pPilha, &c) && (c != '[')) balanceada = 0; break;
            default: TPilha_Empilha(pPilha, cadeia[i]);
        }
    }

    while (! TPilha_EhVazia(pPilha))
        TPilha_Desempilha(pPilha, &c);
    free(pPilha);

    return balanceada;
}
```

Exemplo de Aplicação

- Transformação da notação infixa para pós-fixa

Notação Infixa	Notação Pós-Fixa	Notação Pré-Fixa
$A + B \times C$	$A B C \times +$	$+ \times B C A$
$A \times (B + C)$	$A B C + \times$	$\times + B C A$
$(A + B) / (C - D)$	$A B + C D - /$	$/ + A B - C D$
$(A + B) / (C - D) \times E$	$A B + C D - / E \times$	$\times / + A B - C D E$

Exemplo de Aplicação

- Transformação da notação infixa para pós-fixa
- Considerações a respeito da notação pós-fixa:
 - Não é necessário parênteses
 - A ordem dos operadores na expressão diz a ordem em que eles vão ser executados (da esquerda para a direita)
 - Nota-se que os nomes das variáveis são copiados da entrada infixa para a saída pós-fixa na mesma ordem
 - Os operadores esperam até que apareça na entrada um de prioridade menor ou igual (x ou / precede + ou -)

Exemplo de Aplicação

- Transformação da notação infixa para pós-fixa

Entrada	Símbolo	Ação	Pilha	Saída
$A \times (B + C) / D$	A	Copia	\emptyset	A
$x (B + C) / D$	x	Pilha vazia, empilha	x	A
$(B + C) / D$	(Empilha	(x	A
$B + C) / D$	B	Copia	(x	A B
$+ C) / D$	+	Prioridade menor, empilha	+ (x	A B
$C) / D$	C	Copia	+ (x	A B C
$) / D$)	Desempilha até achar (x	A B C +
$/ D$	/	Prioridade igual, desempilha, empilha	/	A B C + x
D	D	Copia	/	A B C + x D
		Esvazia pilha	\emptyset	A B C

Exemplo de Aplicação

- Transformação da notação infixa para pós-fixa

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef char TItem;

int Prioridade(char op)
{
    switch (op) {
        case '(': return 0;
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
        default: return 3;
    }
}

int main(void)
{
    char expr[] = "a*(b+c)*(d-g)*h";

    printf("%s\n", Infixa2Posfixa(expr));

    return 0;
}
```

- Escreva uma função que recebe como parâmetro uma cadeia de caracteres representando uma expressão aritmética na forma infixa (ex.: A + B x C) e a retorna na forma pós-fixa (ex.: A B C x +)

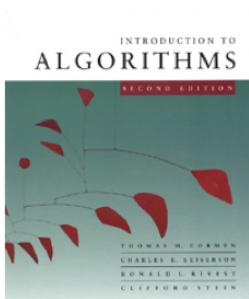
Exercício

```
char *Infixa2Posfixa(char *in)
{
    TPilha *pPilha;
    char *pos;
    int i, j;

    pPilha = (TPilha *) malloc(sizeof(TPilha));
    TPilha_Inicia(pPilha);

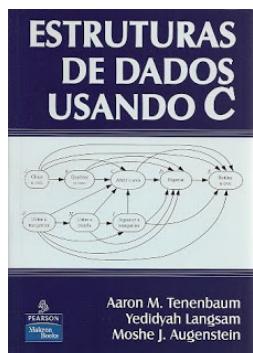
    pos = malloc(strlen(in));
    for (i = 0, j = 0; i < strlen(in); i++) {
        switch (in[i]) {
            case '+':
            case '-':
            case '*':
            case '/':
                while (TPilha_Desempilha(pPilha, &pos[j]) && (Prioridade(pos[j]) >= Prioridade(in[i])))
                    j++;
                if (Prioridade(pos[j]) < Prioridade(in[i]))
                    TPilha_Empilha(pPilha, pos[j]);
                TPilha_Empilha(pPilha, in[i]);
                break;
            case '(': TPilha_Empilha(pPilha, in[i]); break;
            case ')':
                while (TPilha_Desempilha(pPilha, &pos[j]) && (pos[j] != '('))
                    j++;
                break;
            default: pos[j++] = in[i];
        }
    }
    while (! TPilha_EhVazia(pPilha))
        TPilha_Desempilha(pPilha, &pos[j++]);
    free(pPilha);

    return pos;
}
```



CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 3^a Edição. MIT Press, 2009. **Seção 10.1**

ZIVIANI, N. **Projeto de Algoritmos com Implementações em Pascal e C**. 3^a Edição. Cengage Learning, 2010. **Seção 3.2**



TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de Dados usando C**. Pearson Makron Books, 2008.
Capítulo 2