

Pilhas e Filas

Nádia Félix e Hebert Coelho

*Vários slides foram adaptados de Nina Edelwais e Renata Galante
Estrutura de Dados – Série de Livros Didáticos - Informática - UFRGS*

Listas lineares especiais mais usuais

LIFO *Last In First Out*

o último componente inserido
é o primeiro a ser retirado

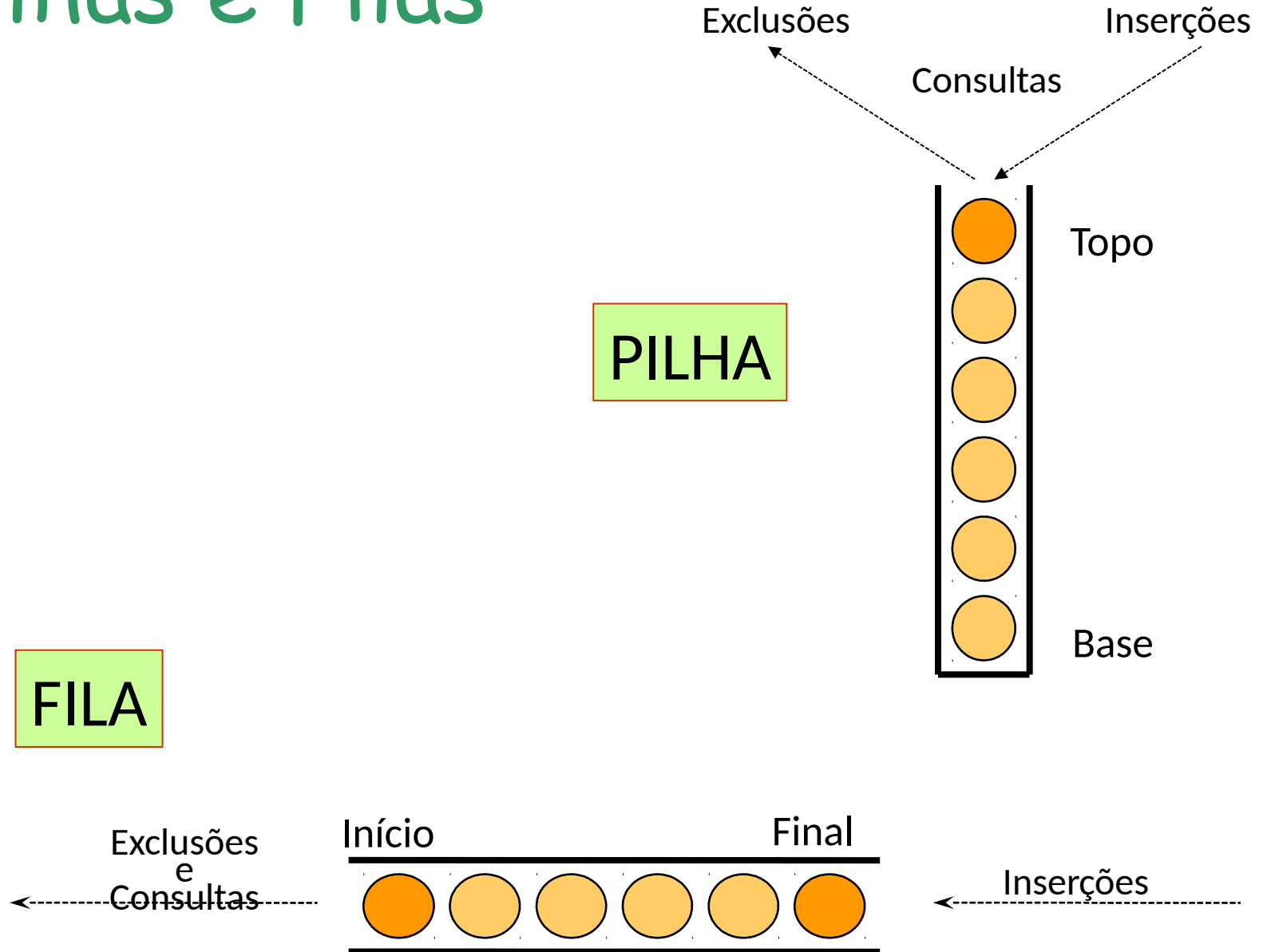
Pilha

FIFO *First In First Out*

o primeiro componente inserido
é também o primeiro a ser retirado

Fila

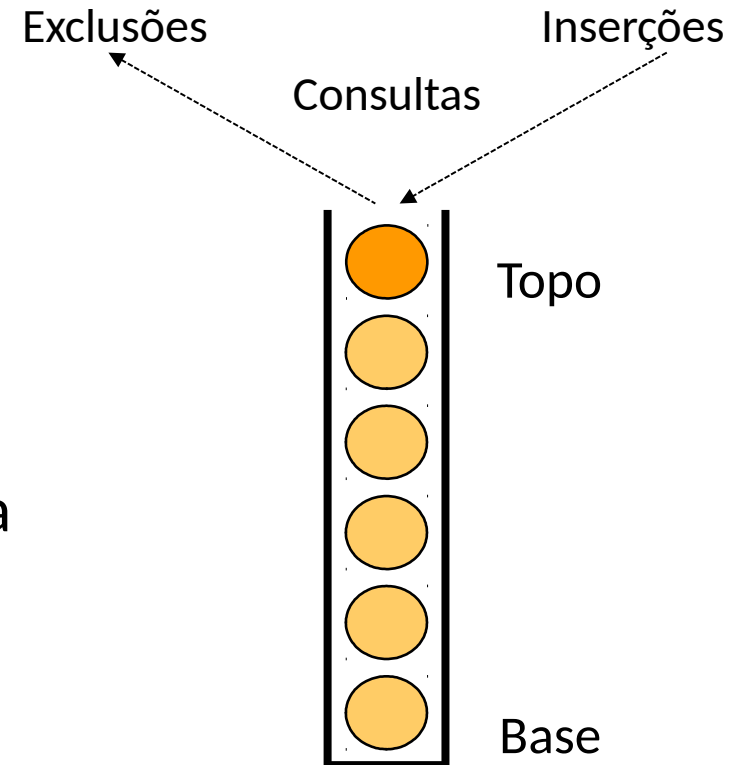
Pilhas e Filas



Pilhas

Operações sobre Pilhas

- Criar uma pilha vazia
- Inserir um elemento no topo da pilha
- Remover um elemento do topo de pilha
- Consultar o topo da pilha
- Destruir a pilha
- Verificar se é cheia
- Verificar se é vazia



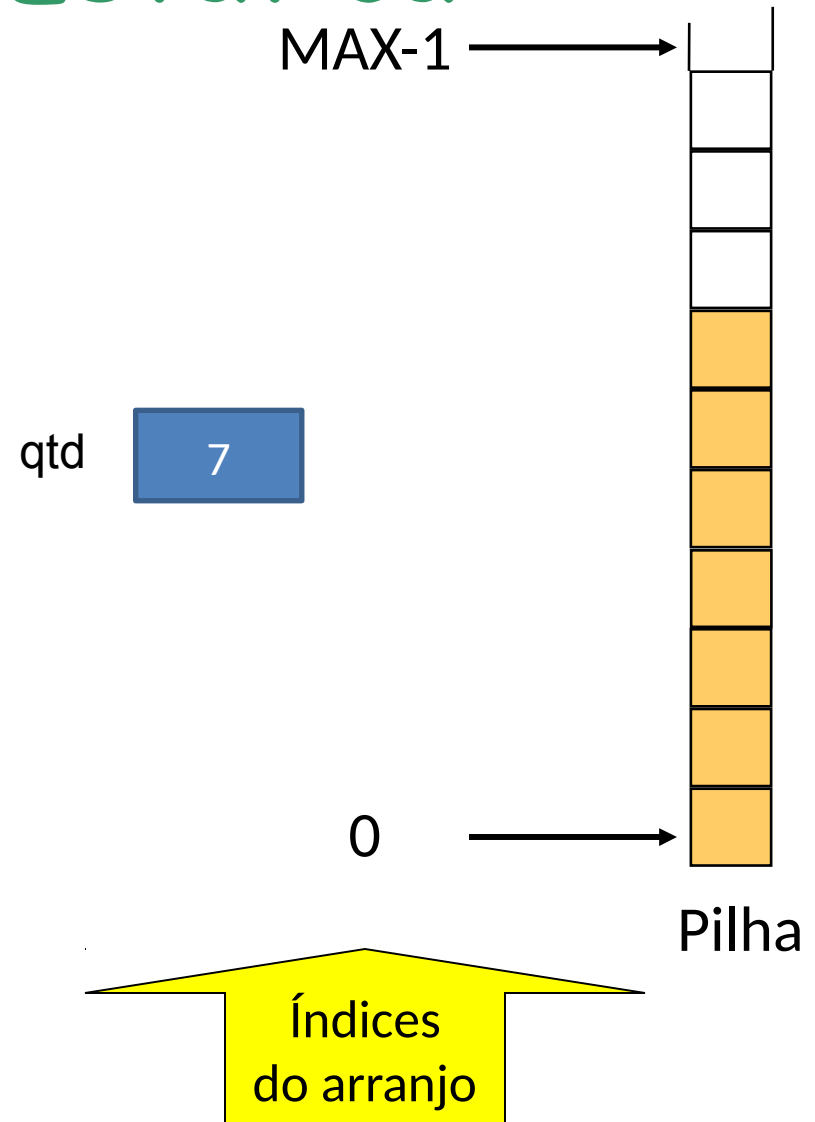
Tipos de pilhas

- Alocação estática com acesso sequencial – uso de um array ou vetor
- Alocação dinâmica com acesso encadeado

PilhaSequencial
estática

Pilha - Sequencial Estática

- Implementada usando um arranjo



PilhaSequencial.h

- O tamanho MAX do array, representada pela constante MAX
- O tipo de dado que será armazenado na pilha, struct aluno
- As funções disponíveis para se trabalhar com essa pilha

PilhaSequencial.c

- As chamadas às bibliotecas necessárias à implementação da pilha
- A definição do tipo que descreve o funcionamento da pilha, struct pilha
- As implementações das funções definidas no arquivo PilhaSequencial.h

PilhaSequencial.h

```
//Arquivo PilhaSequencial.h
#define MAX 100
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct pilha Pilha;

Pilha* cria_Pilha();
void libera_Pilha(Pilha* pi);
int consulta_topo_Pilha(Pilha* pi, struct aluno *al);
int insere_Pilha(Pilha* pi, struct aluno al);
int remove_Pilha(Pilha* pi);
int tamanho_Pilha(Pilha* pi);
int Pilha_vazia(Pilha* pi);
int Pilha_cheia(Pilha* pi);
void imprime_Pilha(Pilha* pi);
```

PilhaSequencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include "PilhaSequencial.h" //inclui os Protótipos

//Definição do tipo Pilha
= struct pilha{
    int qtd;
    struct aluno dados[MAX];
};
```

Criação da pilha

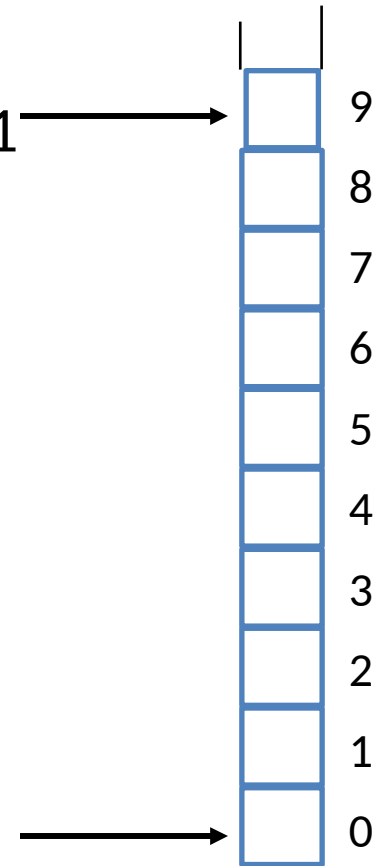
```
Pilha* cria_Pilha(){  
    Pilha *pi;  
    pi = (Pilha*) malloc(sizeof(struct pilha));  
    if(pi != NULL)  
        pi->qtd = 0;  
    return pi;  
}
```

1. Alocar área para a pilha
2. Indicar que a pilha está vazia

qtd


0

MAX - 1




Pilha

Destruir a pilha

A small icon of a code editor window with a white background and a gray border, containing a single line of text.


```
void libera_Pilha(Pilha* pi){  
    free(pi);  
}
```

Tamanho da pilha




```
int tamanho_Pilha(Pilha* pi){  
    if(pi == NULL)  
        return -1;  
    else  
        return pi->qtd;  
}
```

Retornando se a pilha está cheia



```
int Pilha_cheia(Pilha* pi){  
    if(pi == NULL)  
        return -1;  
    return (pi->qtd == MAX);  
}
```


Retornando se a pilha está vazia

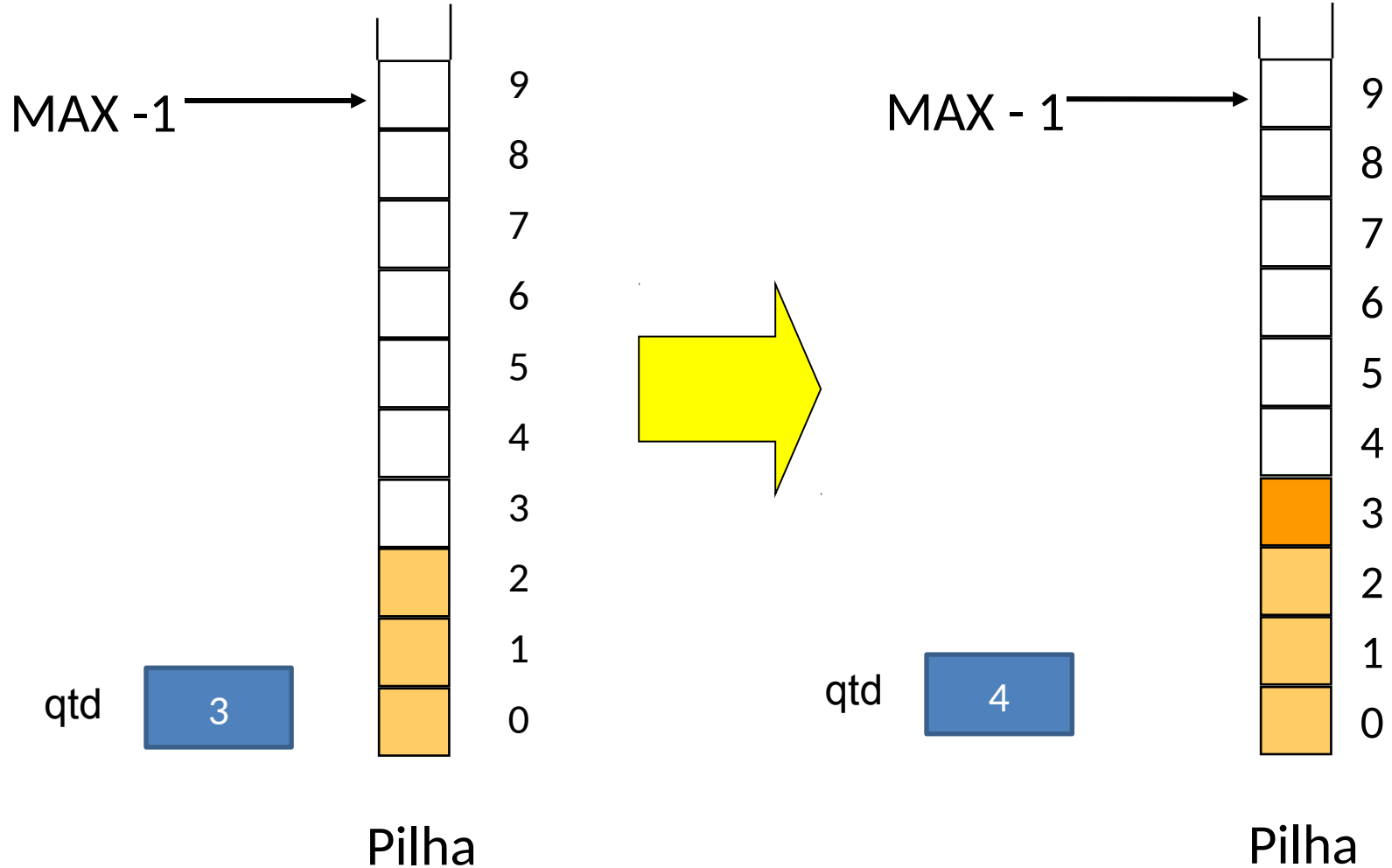


```
int Pilha_vazia(Pilha* pi){  
    if(pi == NULL)  
        return -1;  
    return (pi->qtd == 0);  
}
```

The diagram shows a vertical stack structure with a small square box at the top containing a horizontal line, representing the top of the stack. A vertical line extends downwards from this box, ending in a curly brace that corresponds to the closing brace of the function in the code.

Inserção de um elemento na pilha

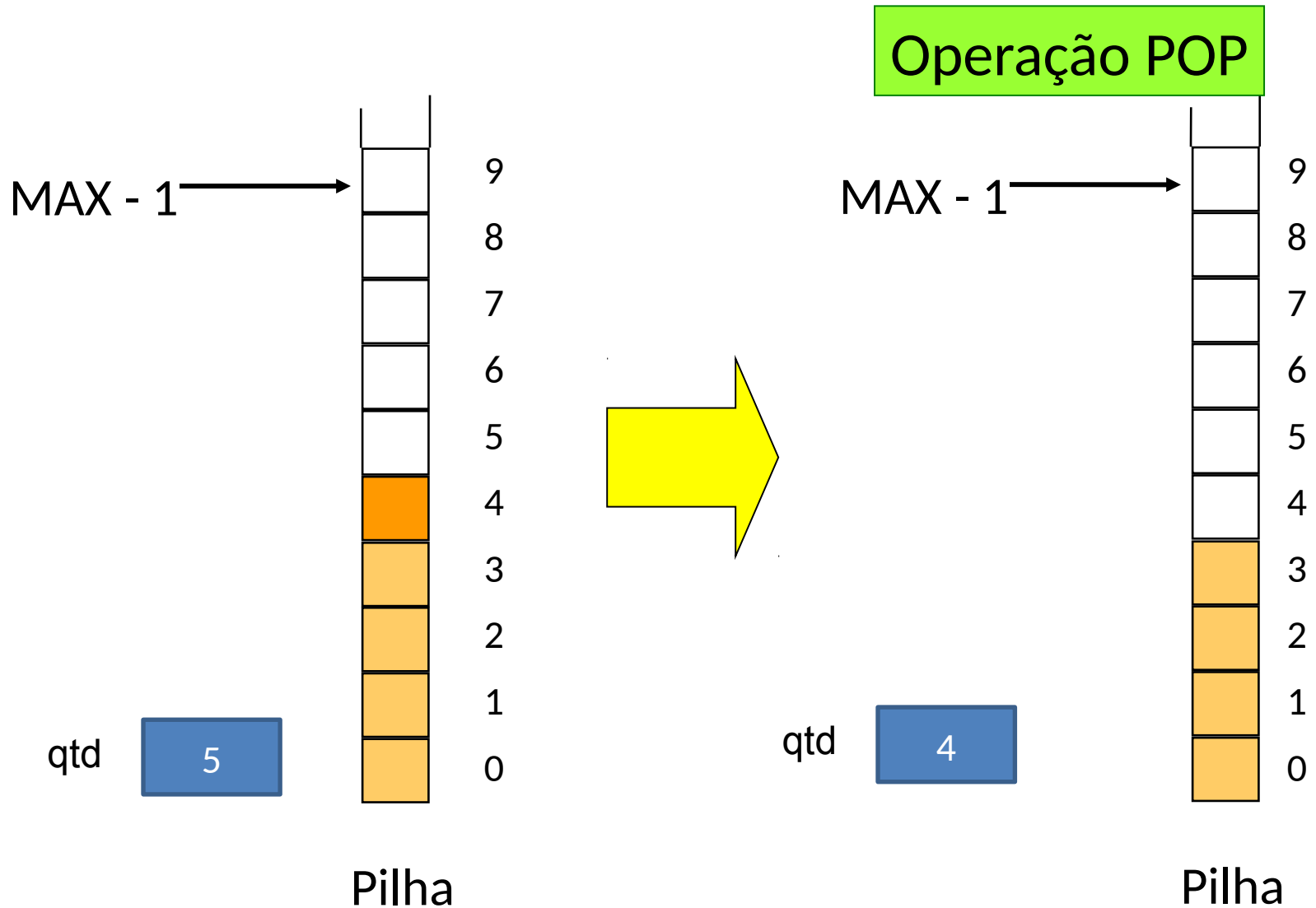
Operação PUSH




Inserindo um elemento na pilha

```
int insere_Pilha(Pilha* pi, struct aluno al){  
    if(pi == NULL)  
        return 0;  
    if(pi->qtd == MAX)//pilha cheia  
        return 0;  
    pi->dados[pi->qtd] = al;  
    pi->qtd++;  
    return 1;  
}
```

Remoção de um elemento da pilha

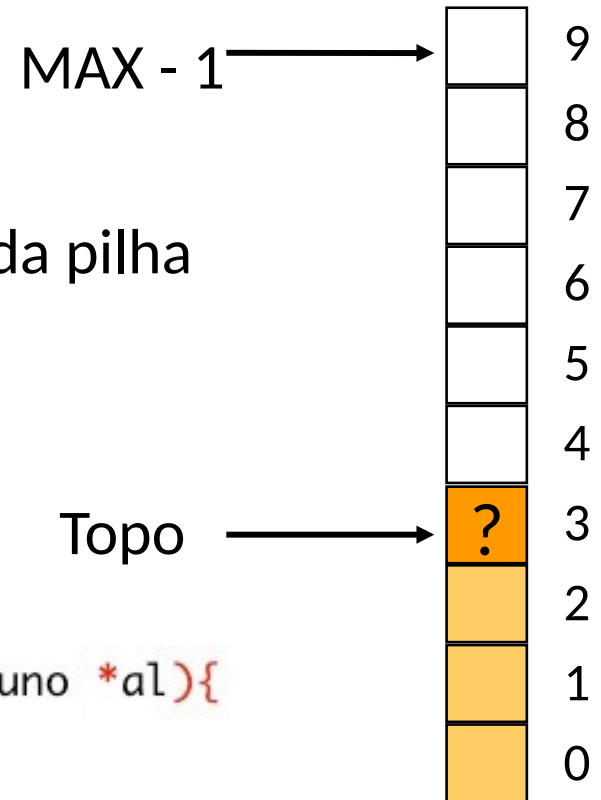


Removendo um elemento da pilha

```
int remove_Pilha(Pilha* pi){  
    if(pi == NULL || pi->qtd == 0)  
        return 0;  
    pi->qtd--;  
    return 1;  
}
```

Consulta o topo da pilha

- Acesso somente ao elemento do topo da pilha



```
int consulta_topo_Pilha(Pilha* pi, struct aluno *al){  
    if(pi == NULL || pi->qtd == 0)  
        return 0;  
    *al = pi->dados[pi->qtd-1];  
    return 1;  
}
```

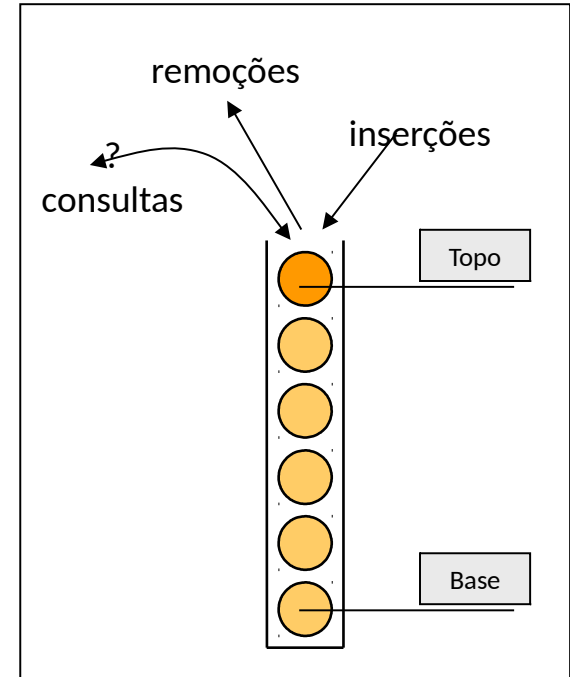
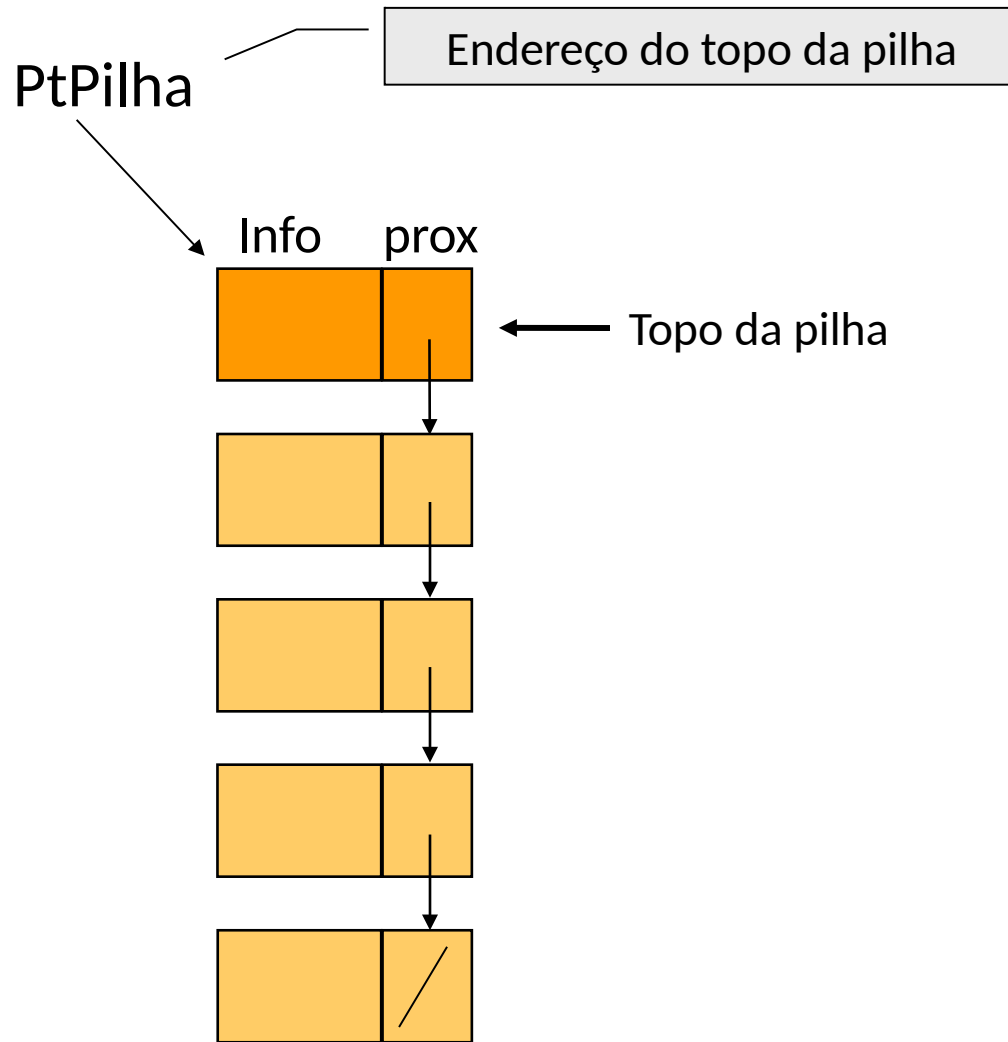
Pilha

Complexidade

- Inserção, remoção e consulta: $O(1)$

Pilhas implementadas
por encadeamento

Pilha implementada por encadeamento



PilhaDin.h

//Arquivo PilhaDin.h

```
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct elemento* Pilha;

Pilha* cria_Pilha();
void libera_Pilha(Pilha* pi);
int consulta_topo_Pilha(Pilha* pi, struct aluno *al);
int insere_Pilha(Pilha* pi, struct aluno al);
int remove_Pilha(Pilha* pi);
int tamanho_Pilha(Pilha* pi);
int Pilha_vazia(Pilha* pi);
int Pilha_cheia(Pilha* pi);
void imprime_Pilha(Pilha* pi);
```

PilhaDin.c

```
#include <stdio.h>
#include <stdlib.h>
#include "PilhaDin.h" //inclui os Protótipos

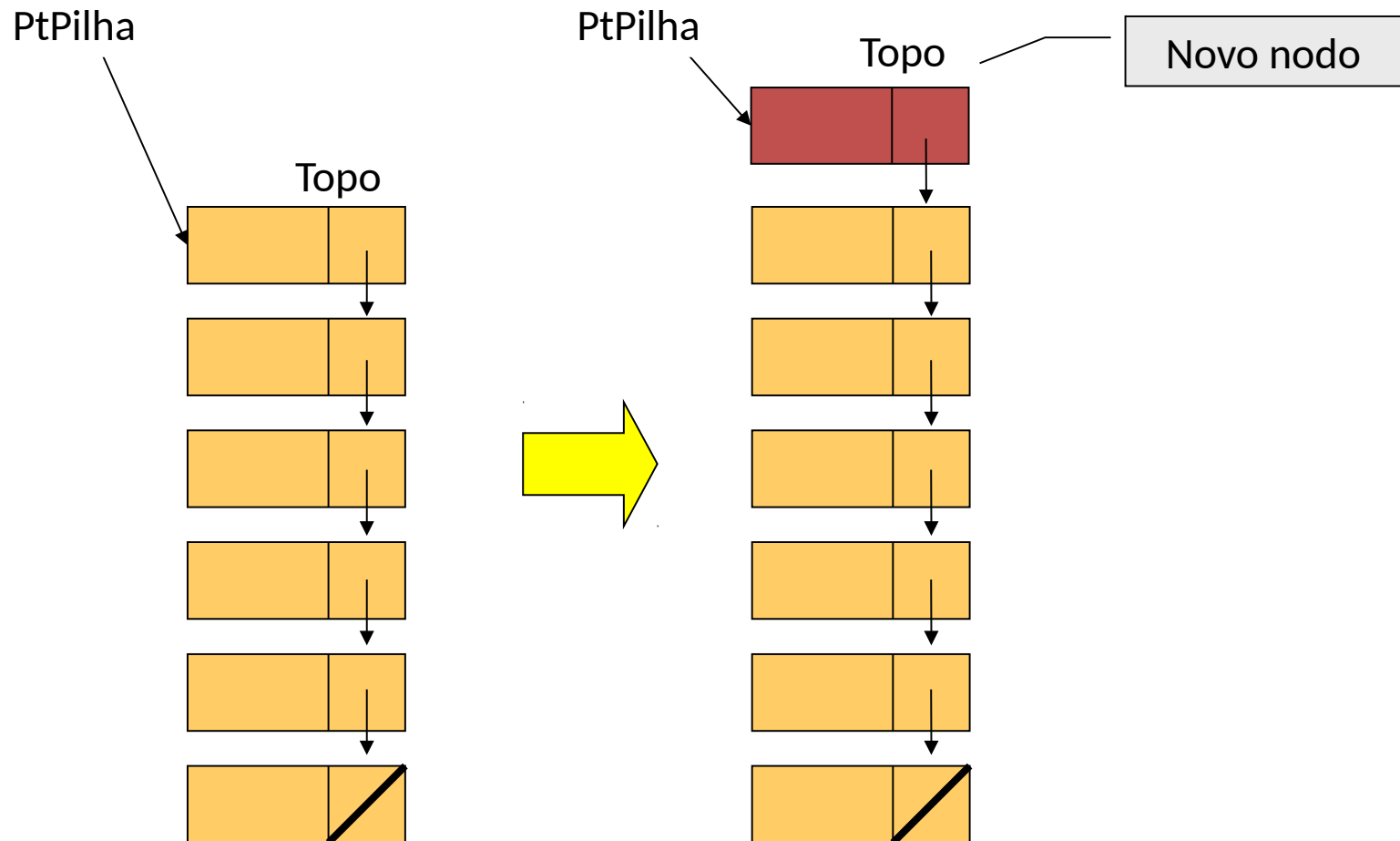
//Definição do tipo Pilha
struct elemento{
    struct aluno dados;
    struct elemento *prox;
};
typedef struct elemento Elem;
```

Cria Pilha vazia

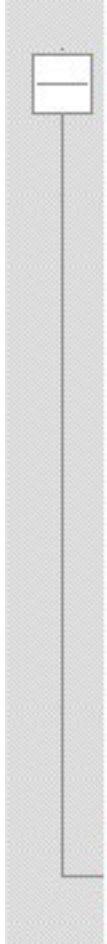
```
Pilha* cria_Pilha(){  
    Pilha* pi = (Pilha*) malloc(sizeof(Pilha));  
    if(pi != NULL)  
        *pi = NULL;  
    return pi;  
}
```

Inserção de um nodo em pilha encadeada

- Novo nodo inserido sempre no topo da pilha



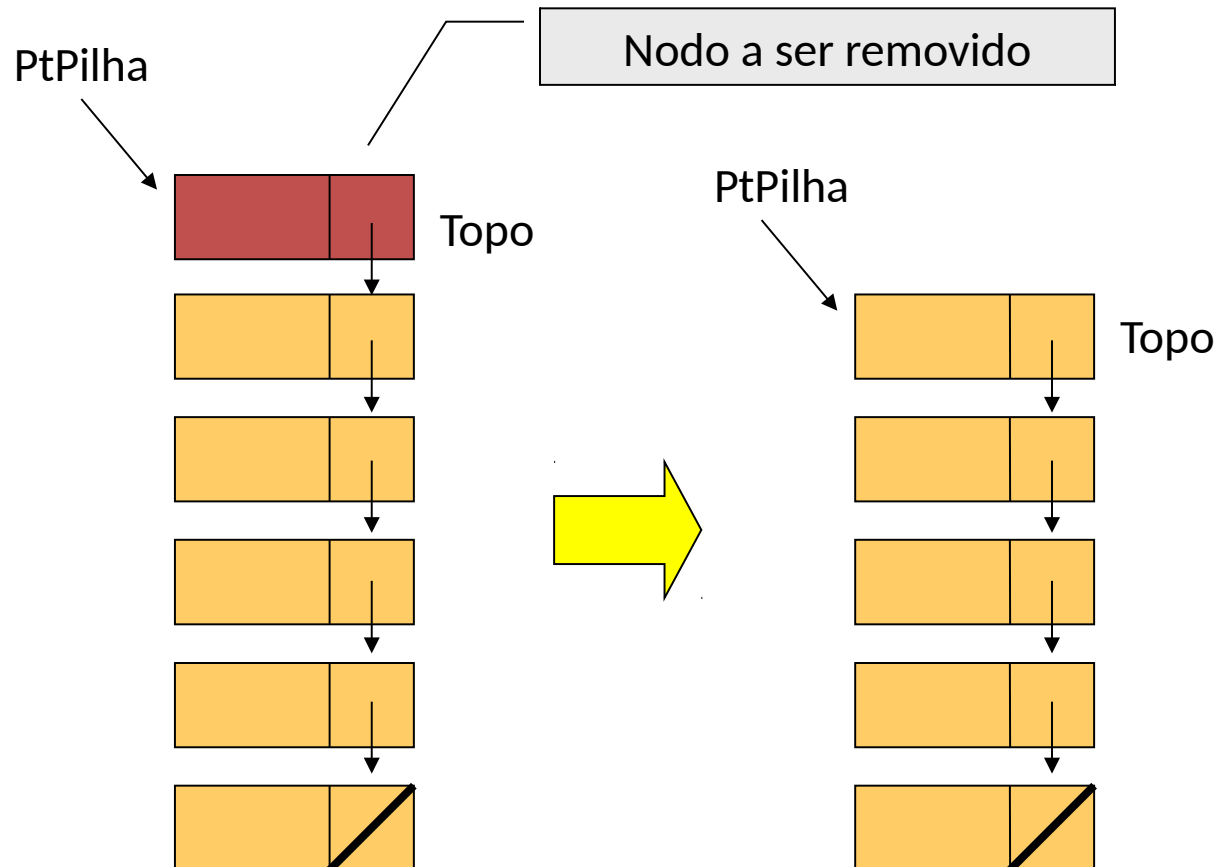
Inserção de um nodo em pilha encadeada

A vertical grey bar on the left side of the code block contains a small white square at the top, representing a pointer. A thin vertical line extends downwards from this square, ending in a curly brace that aligns with the closing brace of the function definition in the code.

```
int insere_Pilha(Pilha* pi, struct aluno al){  
    if(pi == NULL)  
        return 0;  
    Elem* no;  
    no = (Elem*) malloc(sizeof(Elem));  
    if(no == NULL)  
        return 0;  
    no->dados = al;  
    no->prox = (*pi);  
    *pi = no;  
    return 1;  
}
```

Desempilha um nodo de uma pilha encadeada

- Só pode ser removido o nodo do topo da pilha

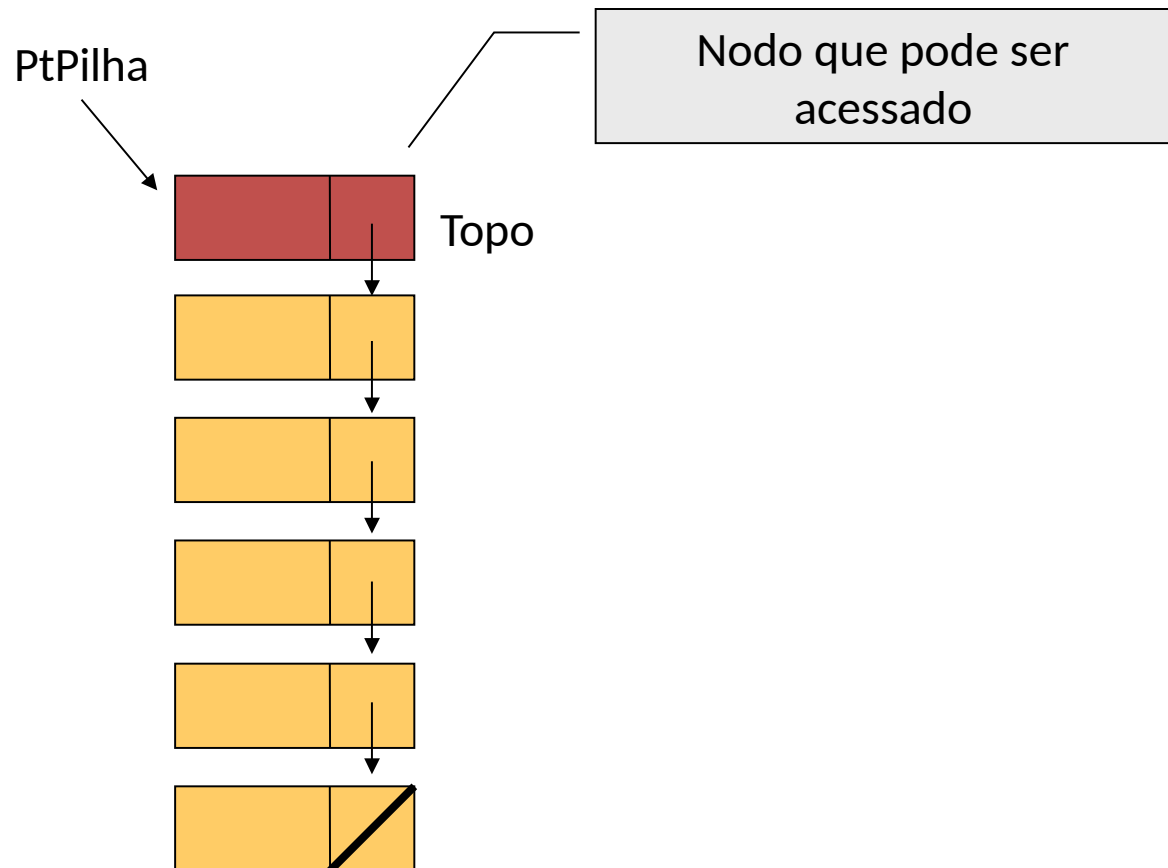


Desempilha um nodo de uma pilha encadeada

```
int remove_Pilha(Pilha* pi){  
    if(pi == NULL)  
        return 0;  
    if((*pi) == NULL)  
        return 0;  
    Elem *no = *pi;  
    *pi = no->prox;  
    free(no);  
    return 1;  
}
```


Consulta à pilha encadeada

- Só pode ser acessado o nodo do topo da pilha

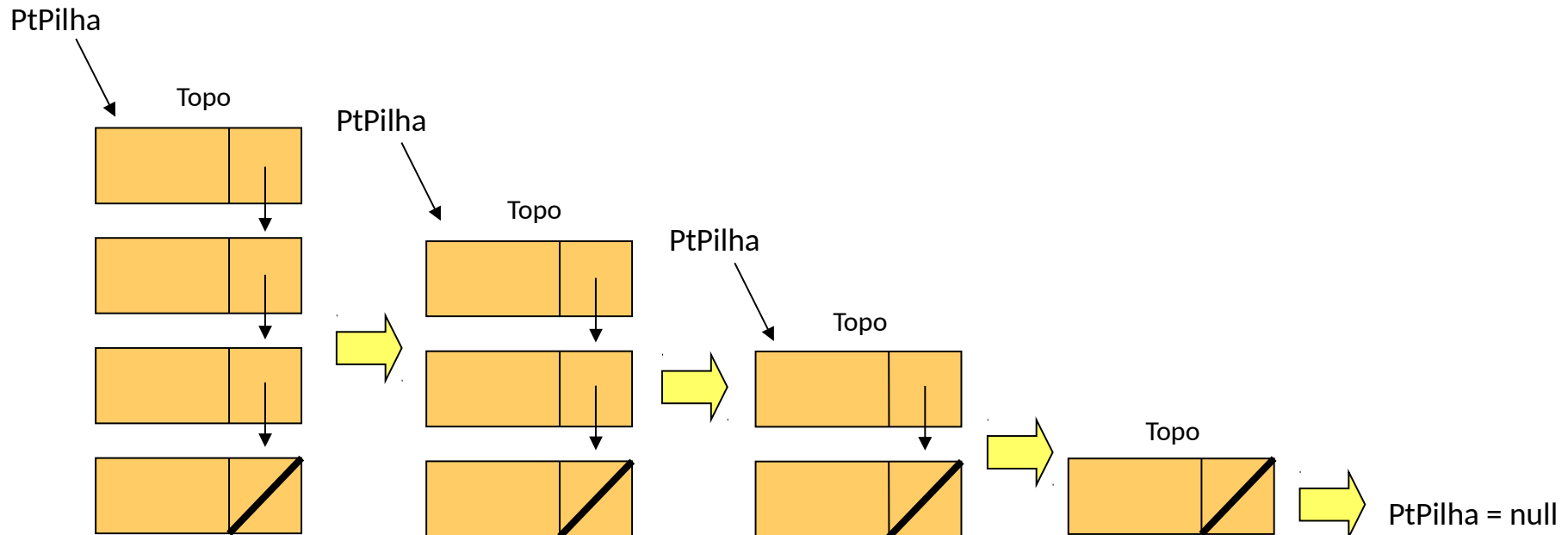


Consulta à pilha


```
int consulta_topo_Pilha(Pilha* pi, struct aluno *al){  
    if(pi == NULL)  
        return 0;  
    if((*pi) == NULL)  
        return 0;  
    *al = (*pi)->dados;  
    return 1;  
}
```

Destruição de uma pilha encadeada

- Liberar espaço ocupado pelos nodos, sempre a partir do topo da pilha
- No final: apontador para o endereço nulo



Destruir pilha

A vertical stack of three rectangular boxes, representing a stack data structure. The top box is empty, the middle box is empty, and the bottom box is empty.

```
void libera_Pilha(Pilha* pi){  
    if(pi != NULL){  
        Elem* no;  
        while((*pi) != NULL){  
            no = *pi;  
            *pi = (*pi)->prox;  
            free(no);  
        }  
        free(pi);  
    }  
}
```

Verifica de pilha esta cheia

```
int Pilha_cheia(Pilha* pi){  
    return 0;  
}
```

Verifica de pilha esta vazia

```
int Pilha_vazia(Pilha* pi){  
    if(pi == NULL)  
        return 1;  
    if(*pi == NULL)  
        return 1;  
    return 0;  
}
```

Exercício

Implemente o TAD Pilha utilizando alocação dinâmica de memória e acesso encadeado.

Teste o programa para reconhecer quais palavras são palíndromos em uma dada frase.

Aplicação da estrutura de dados Pilha em avaliação de expressões aritméticas

Forma das expressões

Considere:

Operandos: $[0, \dots, 9]$

Operadores: $[+, -, /, \times, ^]$

Delimitadores: $[(,)]$

Exemplos:

$$2 + 3 * 5 = 17$$

As operações são efetuadas de acordo com a ordem de precedência dos operadores;

$$(2 + 3) * 5 = 25$$

Os parênteses alteram a ordem natural de avaliação dos operadores.

Proponham um algoritmo para avaliar expressões aritméticas

$$22 - 56 / 2;$$

$$(22 - 56) / 2;$$

$$40 / (2 \times (3 - 1) + 6);$$

$$2^3 \times 4;$$

$$2^{(3 \times 4)};$$



Notação Polonesa e Notação Polonesa Inversa

Notação Polonesa (pré-fixada) - Proposta por Jan lukasiewicz em 1920

- permite escrever uma expressão aritmética em que a precedência é implícita

- operador antes dos operandos:

Notação infixa: $(1-2) * (4+5)$

Notação pré-fixa: $* - 1 2 + 4 5$

Notação Polonesa Inversa (pós-fixada) - proposta por Charles Hamblin em 1950.

- pós-fixa = operador após operandos

Notação pós-fixa: $1 2 - 4 5 + *$

Notação Polonesa e Notação Polonesa Inversa

expressão: -----	forma pós- fixada	forma pré- fixada
$2 + 5 \times 3$	$2\ 5\ 3\ \times\ +$	$+ \ 2\ \times\ 5\ 3$
$2 + 3 - 4$	$2\ 3\ +\ 4\ -$	$- \ +\ 2\ 3\ 4$
$2 + (3 - 4)$	$2\ 3\ 4\ -\ +$	$+ \ 2\ -\ 3\ 4$
$(2 + 4)/(3 - 1)$	$2\ 4\ +\ 3\ 1\ -\ /$	$/ \ +\ 2\ 4\ -3\ 1$
$(2+4)/(3-1)\times 4$	$2\ 4\ +\ 3\ 1\ -\ / \ 4\ \times$	$\times \ / \ +2\ 4-3\ 1\ 4$
$2^2 \times 3 - 4 + 1/2/(1+1)$	$2\ 2\ ^\wedge\ 3\ *\ 4\ -\ 1\ 2\ /\ 1\ 1\ +\ /\ +$	$+ \ -\ *\ ^\wedge\ 2\ 2\ 3\ 4\ /\ /\ 1\ 2\ +\ 1\ 1$

Um possível algoritmo para avaliação de uma expressão pós-fixa

- cada operando é empilhado numa pilha de valores
- quando se encontra um operador
 - desempilha-se o número apropriado de operandos (dois para operadores binários e um para operadores unários)
 - realiza-se a operação devida
 - empilha-se o resultado
- Exemplo:
- avaliação da expressão $1\ 2\ -\ 4\ 5\ +\ *$

empilhe os valores 1 e 2	1 2 - 4 5 + *	<div>2</div> <div>1</div>
quando aparece o operador “-”	1 2 - 4 5 + *	
desempilhe 1 e 2		
empilhe -1, o resultado da operação (1 - 2)		-1
empilhe os valores 4 e 5	1 2 - 4 5 + *	<div>5</div> <div>4</div> <div>-1</div>
quando aparece o operador “+”	1 2 - 4 5 + *	
desempilhe 4 e 5		-1
empilhe 9, o resultado da operação (4+5)		<div>9</div> <div>-1</div>
quando aparece o operador “*”	1 2 - 4 5 + *	
desempilhe -1 e 9		
empilhe -9, o resultado da operação (-1*9)		-9

Como transformar uma expressão na forma infixa para pos-fixa???

- Expressões entre parênteses devem ser convertidos de tal forma que possam ser tratadas como um único operando.
- Somente operadores são empilhados . Um operador é empilhado apenas se possui precedência maior que o operador no topo da pilha.
- Abre parênteses é sempre empilhado
- Fecha parênteses nunca é empilhado (menor precedência). Todos os operadores são desempilhados até encontrar um '('.
- Operandos e operadores desempilhados são colocados na expressão na forma pós-fixa

Exercício usando pilha

- 1) Escreva uma função que transforma uma expressão da forma in-fixa para a forma pós-fixa.
- 2) Escreva uma função que avalie uma expressão na forma pós-fixa
- 3) Escreva um programa que entra com a expressão aritmética pela linha de comando e avalia a expressão.

Exercício usando pilha - cont.

Os operandos são números reais (positivos e negativos);

Os operadores são: $\{+, -, \times, /, ^\}$

Os delimitadores são: $\{ (,) \}$

Ex: $3.4 \times 5 + (2.1 - 12)$

$(-23 + 24.3 / 2.1) ^ 3$