PILHA ESTÁTICA

Tipos de Dados Abstratos (ADT)

- ADT é um tipo de dados que implementa objetos cujo comportamento é definido por um conjunto de valores e operações.
- O conceito da estrutura é separado de sua implementação subjacente.
- O que importa é ***o que*** ela faz, e não ***como*** ela faz.
- Empregadas para simplificar diversas operações em programação.

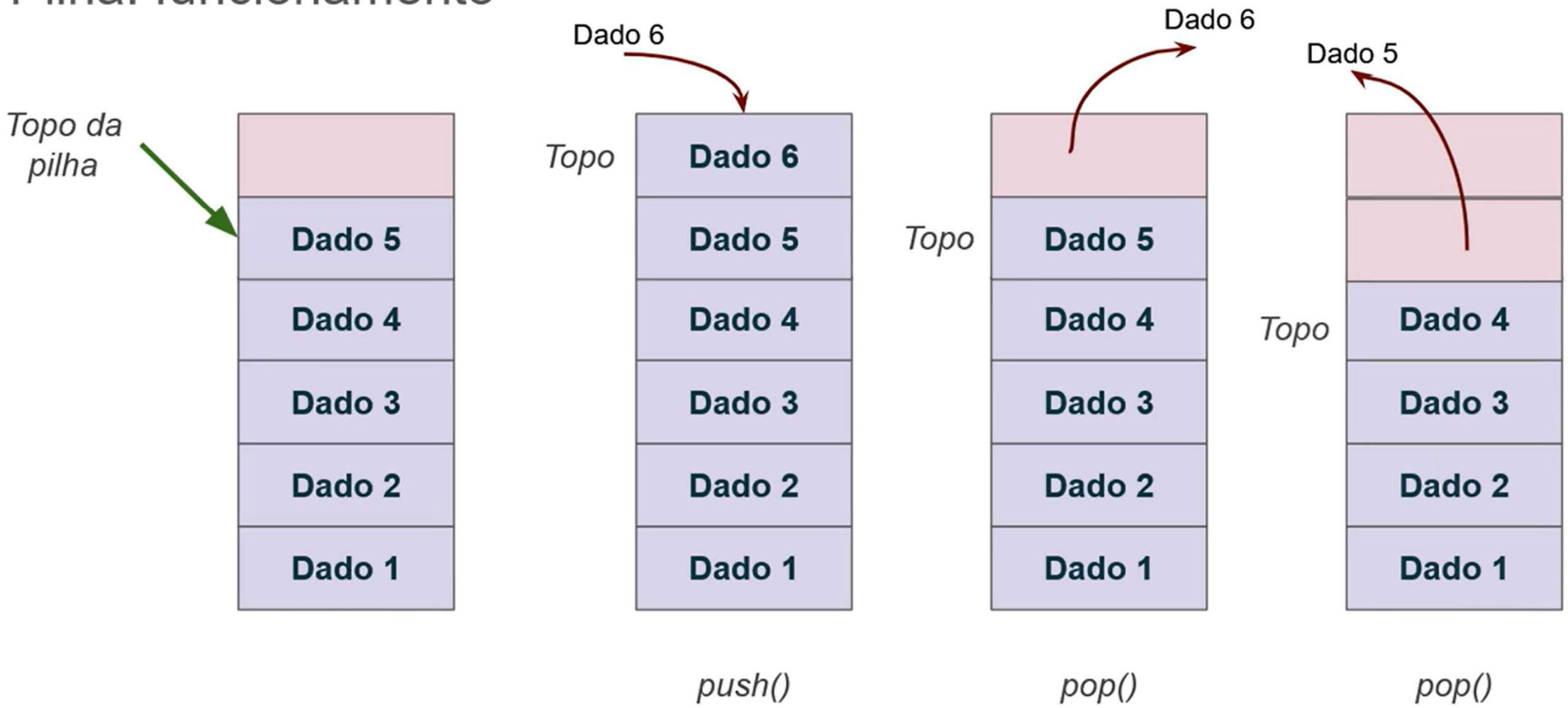
Os principais tipos de ADTs são a lista, pilha e fila.

Pilha

- Estrutura linear para armazenamento de itens que são inseridos e removidos de acordo com o princípio LIFO (last-in-first-out)
- Os objetos podem ser inseridos a qualquer momento, mas apenas o objeto que foi inserido por último pode ser removido.
- Os elementos podem ser inseridos e excluídos apenas em um lado da lista, chamado de topo.

Dado 5
Dado 4
Dado 3
Dado 2
Dado 1

Pilha: funcionamento



Operações em Pilhas

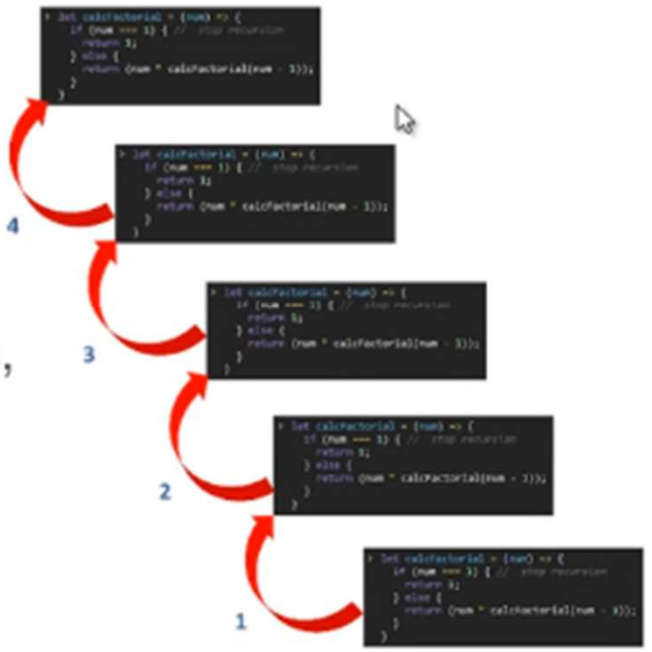
Diversas operações podem ser realizadas em pilhas, tais como:

- push(): Inserir um item o topo da pilha
- pop(): Remover e retornar o elemento do topo da pilha, se não estiver vazia.
- peek(): Retorna o elemento no topo da pilha, sem removê-lo.
- size(): retorna o tamanho da pilha (n° de itens)
- isEmpty(): retorna um booleano informando se a pilha está vazia
- isFull(): retorna um booleano informando se a pilha está cheia

Aplicações das Pilhas

As pilhas encontram inúmeras aplicações em desenvolvimento de algoritmos e software em geral:

- Implementação de funções recursivas
- Mecanismos de desfazer /refazer em aplicações gerais
- Verificação sintática em compiladores
- Avaliação de expressões aritméticas em certas calculadoras
- Operações de backtracking (em rotas de mapas e jogos, por exemplo)
- Avaliação de expressões em geral
- NLP (Natural Language Processing)



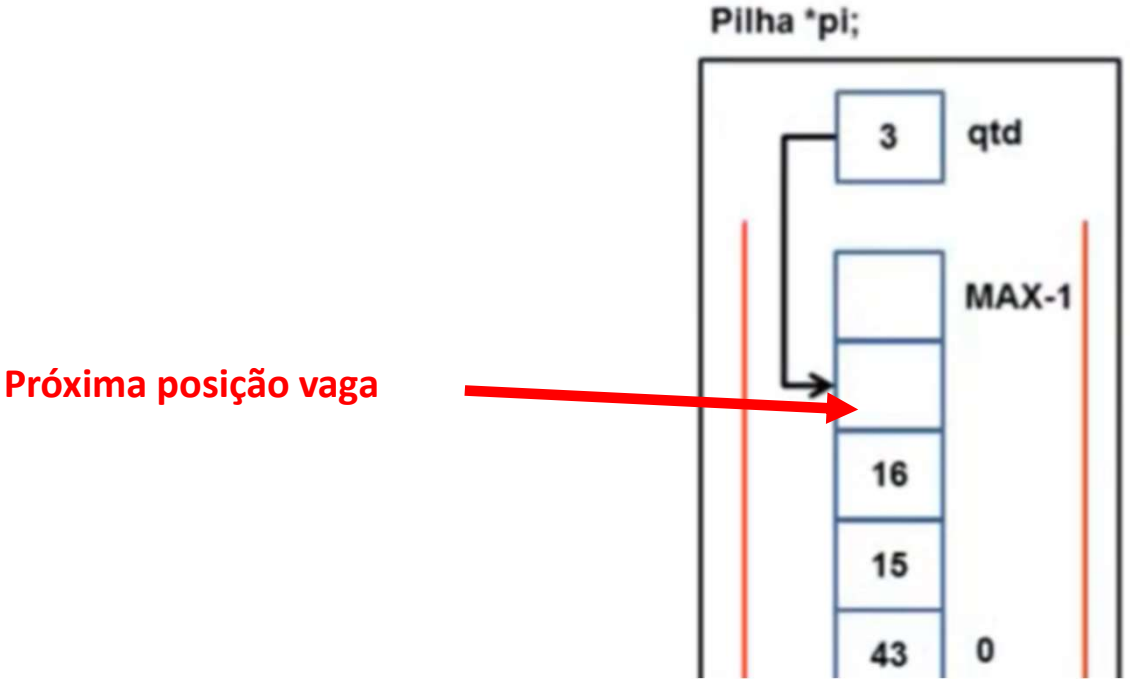
Pilha

- cria uma Pilha vazia;
- insere um elemento no topo da Pilha;
- remove o elemento no topo da Pilha;
- verifica se a Pilha esta vazia;
- Verifica se a pilha está cheia;
- Verifica o tamanho da Pilha;
- Consulta elemento da Pilha;
- libera a Pilha.

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

int main()
{
```

Pilha estática: Tipo de Pilha onde o sucessor de um elemento ocupa a posição física do seguinte do mesmo (uso de array)

```
Pilha *pi;

}
```



Pilha

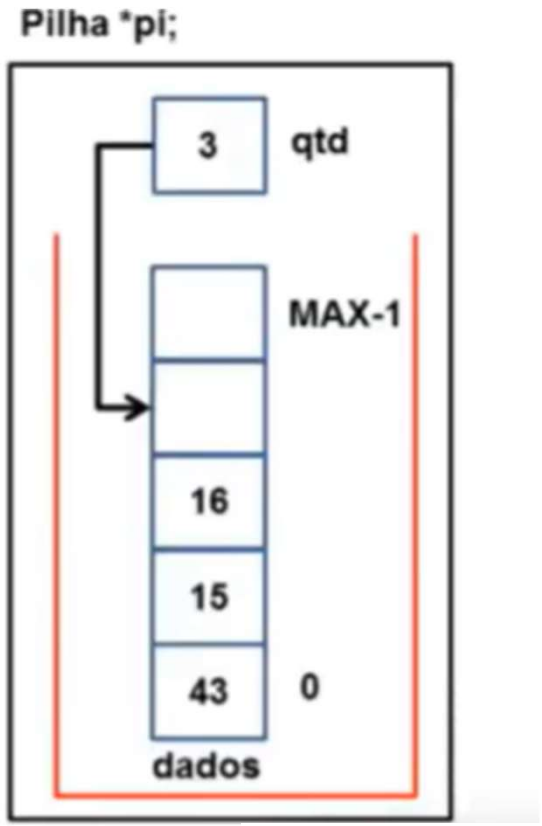
Implementando uma "Pilha Estática"

"PilhaSequencial.h": definir
- os protótipos das funções
- o tipo de dado armazenado na pilha
- o ponteiro "pilha"
- tamanho do vetor usado na pilha

"PilhaSequencial.c": definir
- o tipo de dados "pilha"
- implementar as suas funções.

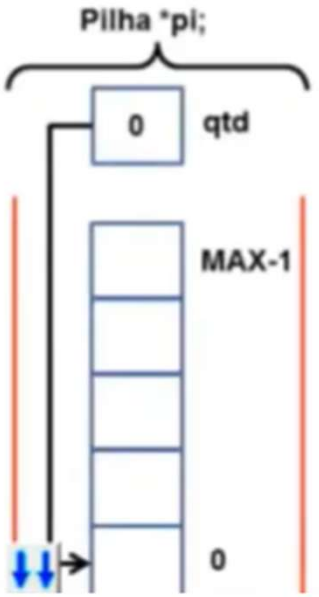
IMPLEMENTAÇÃO

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

//Arquivo PilhaSequencial.c
struct pilha{
    int qtd;
    struct aluno dados[MAX];
};

//programa principal
Pilha *pi;
```



Pilha - CRIAR

```
//programa principal
pi = cria_Pilha();

//Arquivo PilhaSequencial.h
Pilha* cria_Pilha();

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

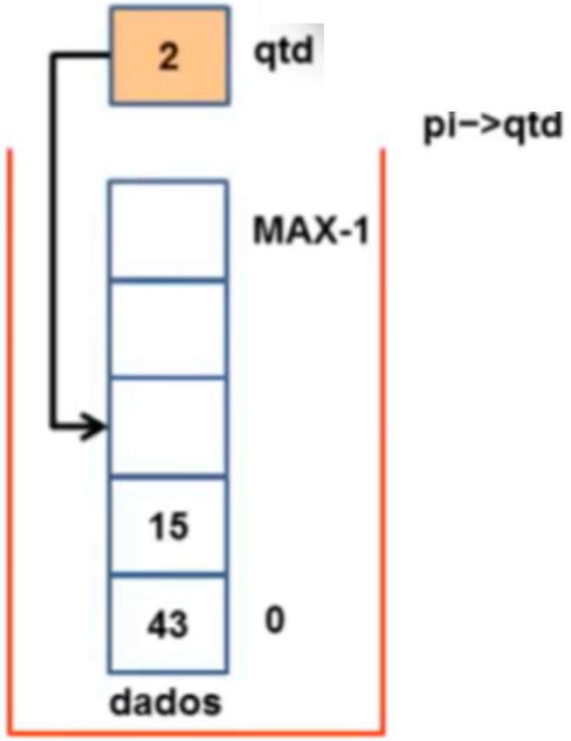


Pilha - Liberar

```
//programa principal
libera_Pilha(pi);

//Arquivo PilhaSequencial.h
void libera_Pilha(Pilha* pi);

//Arquivo PilhaSequencial.c
void libera_Pilha(Pilha* pi){
    free(pi);
}
```

Pilha - TAMANHO

```c
//programa principal
int x = tamanho_Pilha(pi);

//Arquivo PilhaSequencial.h
int tamanho_Pilha(Pilha* pi);

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

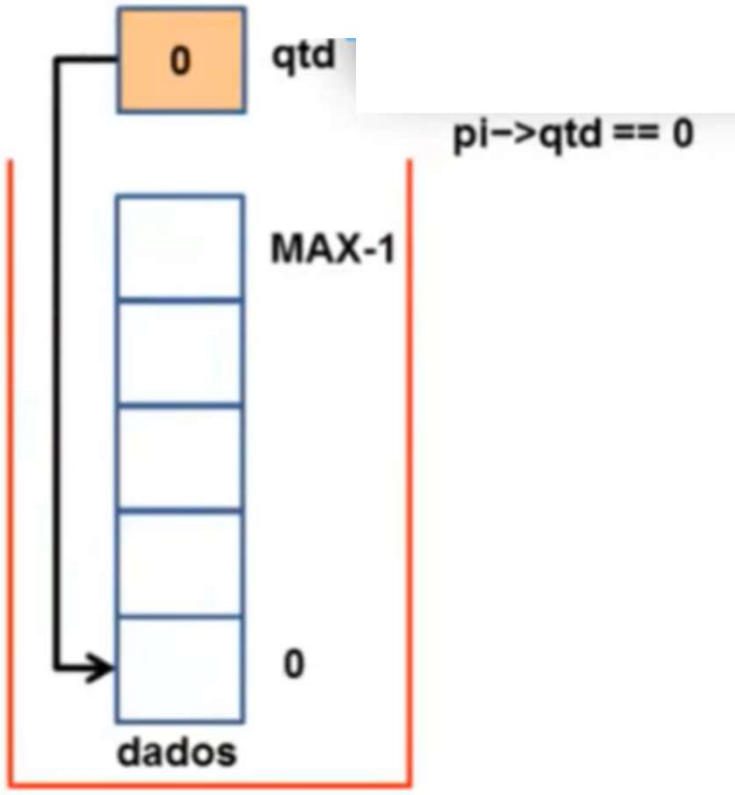


Pilha - CHEIA

```
//programa principal
int x = Pilha_cheia(pi);
if(Pilha_cheia(pi))
//Arquivo PilhaSequencial.h
int Pilha_cheia(Pilha* pi);
//Arquivo PilhaSequencial.c
int Pilha_cheia(Pilha* pi){
    if(pi == NULL)
        return -1;
    return (pi->qtd == MAX);
}
```

Pilha - VAZIA

```c
//programa principal
int x = Pilha_vazia(pi);
if(Pilha_vazia(pi))
//Arquivo PilhaSequencial.h
int Pilha_vazia(Pilha* pi);
//Arquivo PilhaSequencial.c
int Pilha_vazia(Pilha* pi){
    if(pi == NULL)
        return -1;
    return (pi->qtd == 0);
}
```

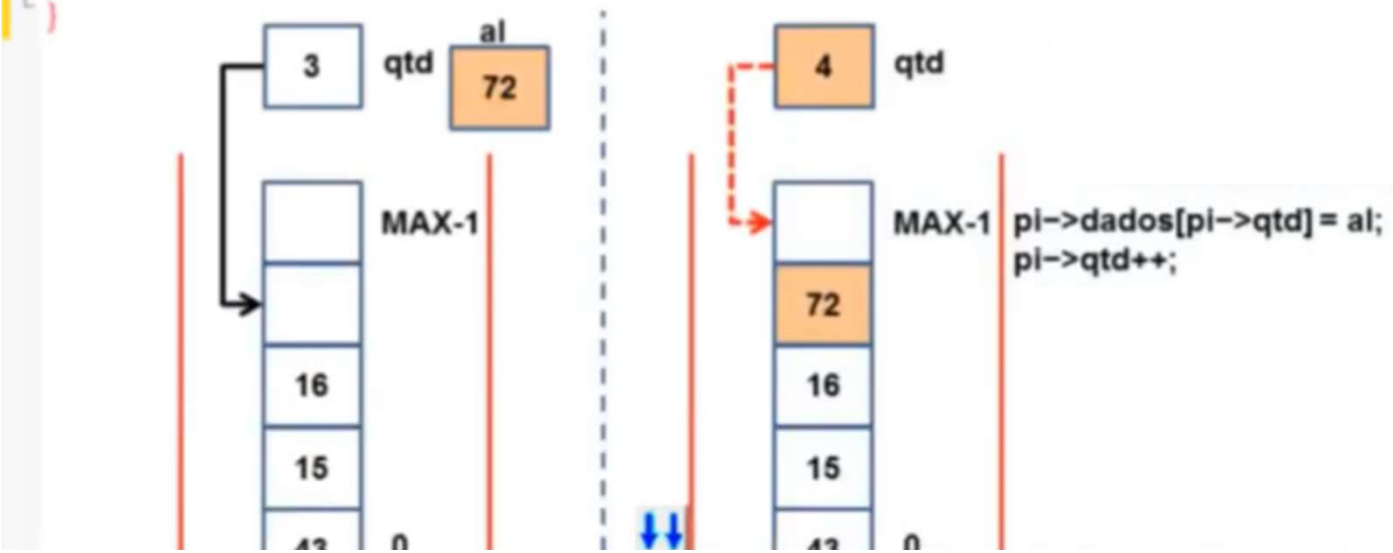


Pilha – INSERÇÃO DE UM ELEMENTO

Sempre no início da Pilha

****** não pode inserir elemento em pilha cheia**

```c
//programa principal
int x = insere_Pilha(pi, dados_aluno);
//Arquivo PilhaSequencial.h
int insere_Pilha(Pilha* pi, struct aluno al);
//Arquivo PilhaSequencial.c
int insere_Pilha(Pilha* pi, struct aluno al){
    if(pi == NULL) return 0;
    if(Pilha_cheia(pi)) return 0;
    pi->dados[pi->qtd] = al;
    pi->qtd++;
    return 1;
}
```

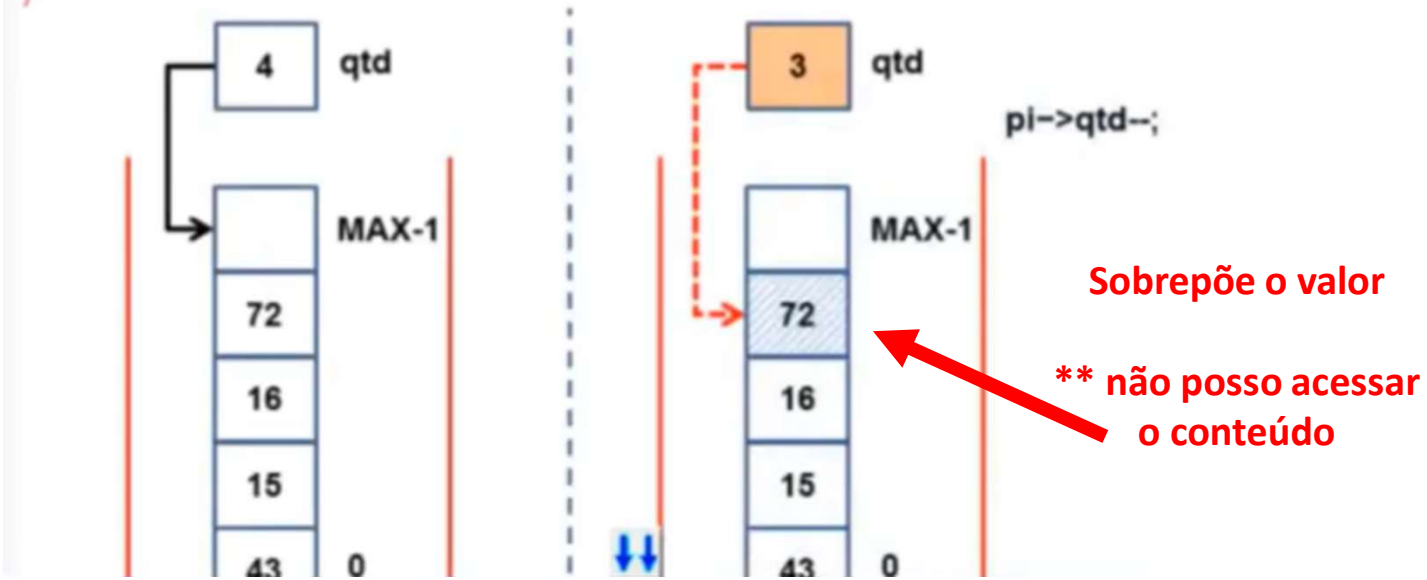


Pilha – REMOÇÃO DE UM ELEMENTO

Sempre no início da Pilha

****** não pode remover elemento em pilha vazia**

```
//programa principal
int x = remove_Pilha(pi);
//Arquivo PilhaSequencial.h
int remove_Pilha(Pilha* pi);
//Arquivo PilhaSequencial.c
int remove_Pilha(Pilha* pi){
    if(pi == NULL || pi->qtd == 0)
        return 0;
    pi->qtd--;
    return 1;
}
```



Pilha - CONSULTA

A consulta se dá apenas pelo elemento que está no Topo da Pilha

```c
//programa principal
int x = consulta_topo_Pilha(pi,&dados_aluno);
//Arquivo PilhaSequencial.h
int consulta_topo_Pilha(Pilha* pi, struct aluno *al);
//Arquivo PilhaSequencial.c
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;
}
```