

RAFAEL VIEIRA COELHO



PILHAS

SUMÁRIO

- Estruturas de Dados
- Listas Simplesmente Encadeada
- Listas Duplamente Encadeada
- Filas
- **Pilhas**
- Grafos
- Árvores

ANDRÉ BACKES

Estrutura de dados descomplicada em linguagem

C

DEFINIÇÃO DE PILHA

O conceito de pilha é algo bastante comum para as pessoas. Trata-se de um conjunto finito de itens sobre um mesmo tema.

Mas, diferente das listas, os itens de uma pilha se encontram dispostos uns sobre os outros.

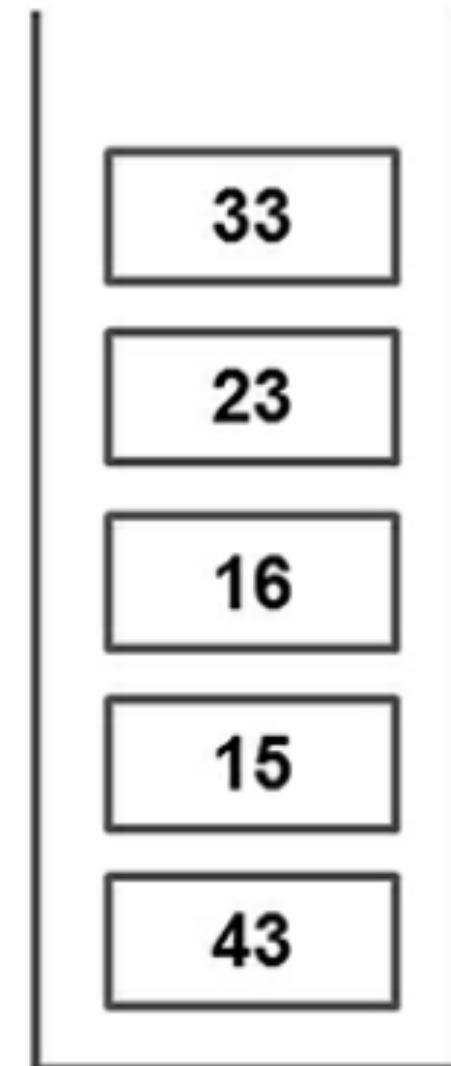
Assim, somente podemos inserir um novo item na pilha se o colocarmos acima dos demais e apenas removeremos o item que está no topo da pilha.

Por esse motivo, as pilhas são conhecidas como estruturas do tipo último a entrar, primeiro a sair ou **LIFO (Last In First Out)**: os elementos são removidos da pilha na ordem inversa daquela em que foram inseridos.

OPERAÇÕES BÁSICAS DE UMA PILHA



Pilha



- Criação da pilha.
- Inserção de um elemento no topo da pilha.
- Remoção de um elemento do topo da pilha.
- Acesso ao elemento do topo da pilha.
- Destruuição da pilha.
- Além de informações com tamanho, se a pilha está cheia ou vazia.

I) PILHA SEQUENCIAL ESTÁTICA



Além do array, essa pilha utiliza um campo adicional (**qtd**) que serve para indicar o quanto do array já está ocupado pelos elementos (**dados**) inseridos na pilha.

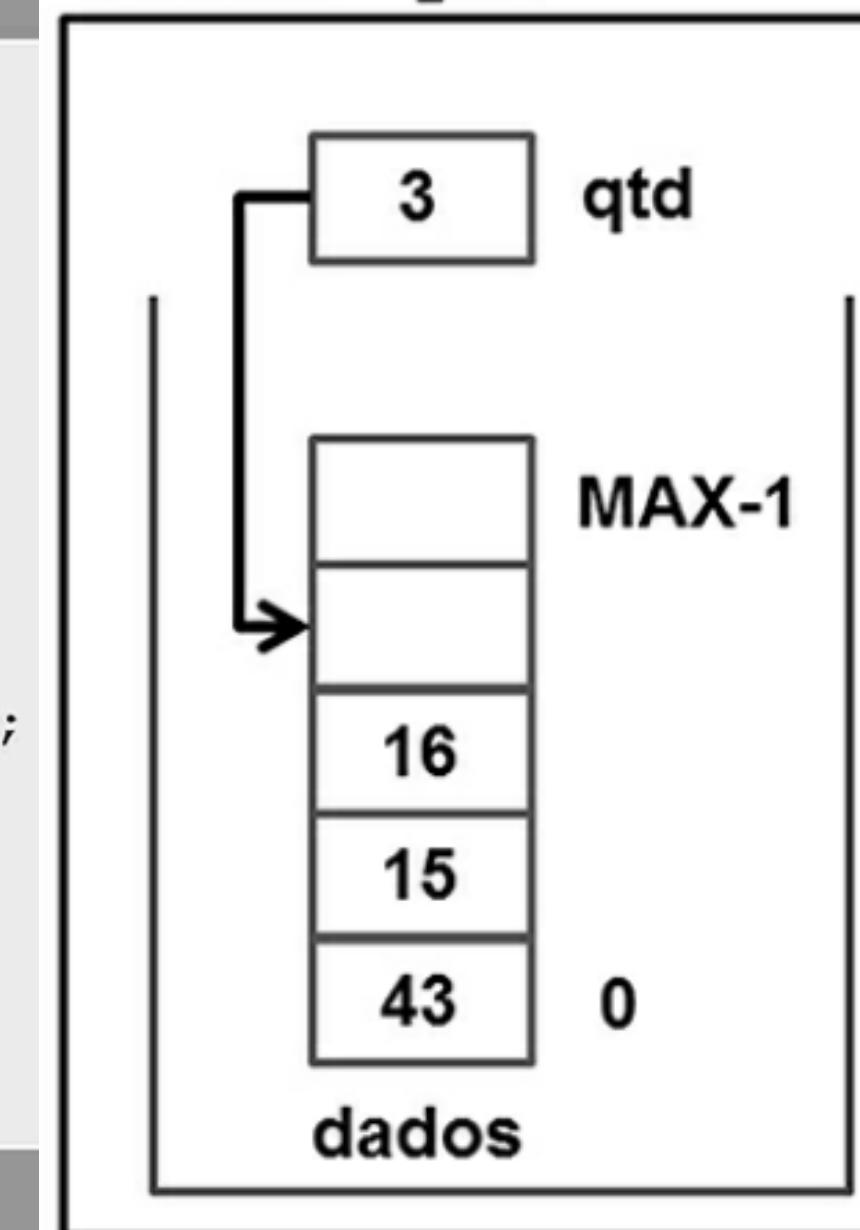
Arquivo PilhaSequencial.h

```
01 #define MAX 100
02 struct aluno{
03     int matricula;
04     char nome[30];
05     float n1,n2,n3;
06 };
07 typedef struct pilha Pilha;
08
09 Pilha* cria_Pilha();
10 void libera_Pilha(Pilha* pi);
11 int acessa_topo_Pilha(Pilha* pi, struct aluno *al);
12 int insere_Pilha(Pilha* pi, struct aluno al);
13 int remove_Pilha(Pilha* pi);
14 int tamanho_Pilha(Pilha* pi);
15 int Pilha_vazia(Pilha* pi);
16 int Pilha_cheia(Pilha* pi);
```

Arquivo PilhaSequencial.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include "PilhaSequencial.h" //inclui os protótipos
04 //Definição do tipo Pilha
05 struct pilha{
06     int qtd;
07     struct aluno dados[MAX];
08 };
```

Pilha *pi;



Criando uma pilha

```
01 Pilha* cria_Pilha() {  
02     Pilha *pi;  
03     pi = (Pilha*) malloc(sizeof(struct pilha));  
04     if(pi != NULL)  
05         pi->qtd = 0;  
06     return pi;  
07 }
```

Destruindo uma pilha

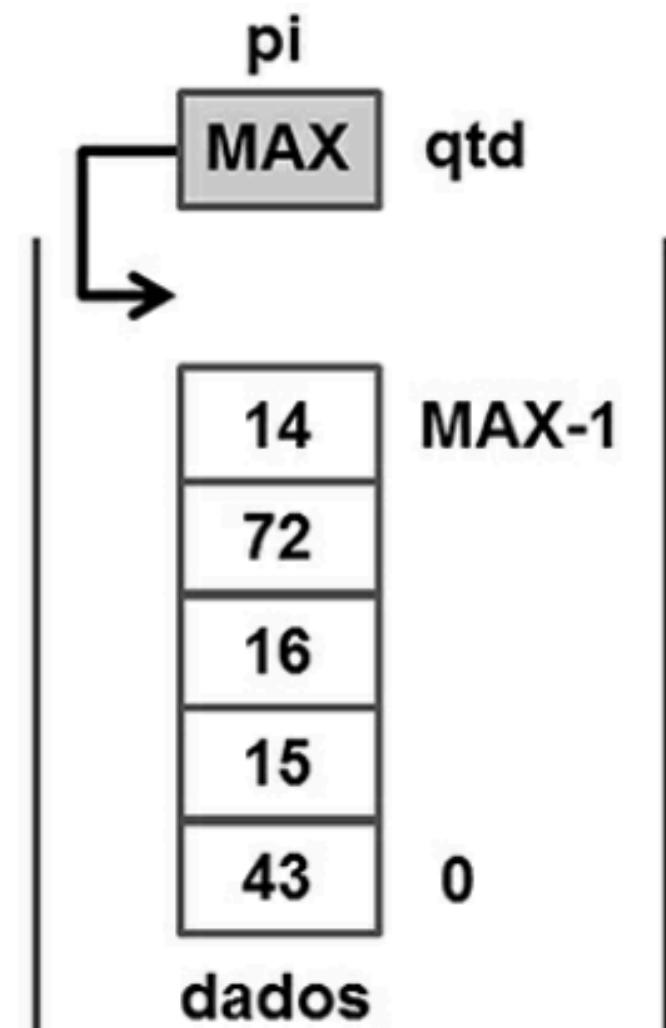
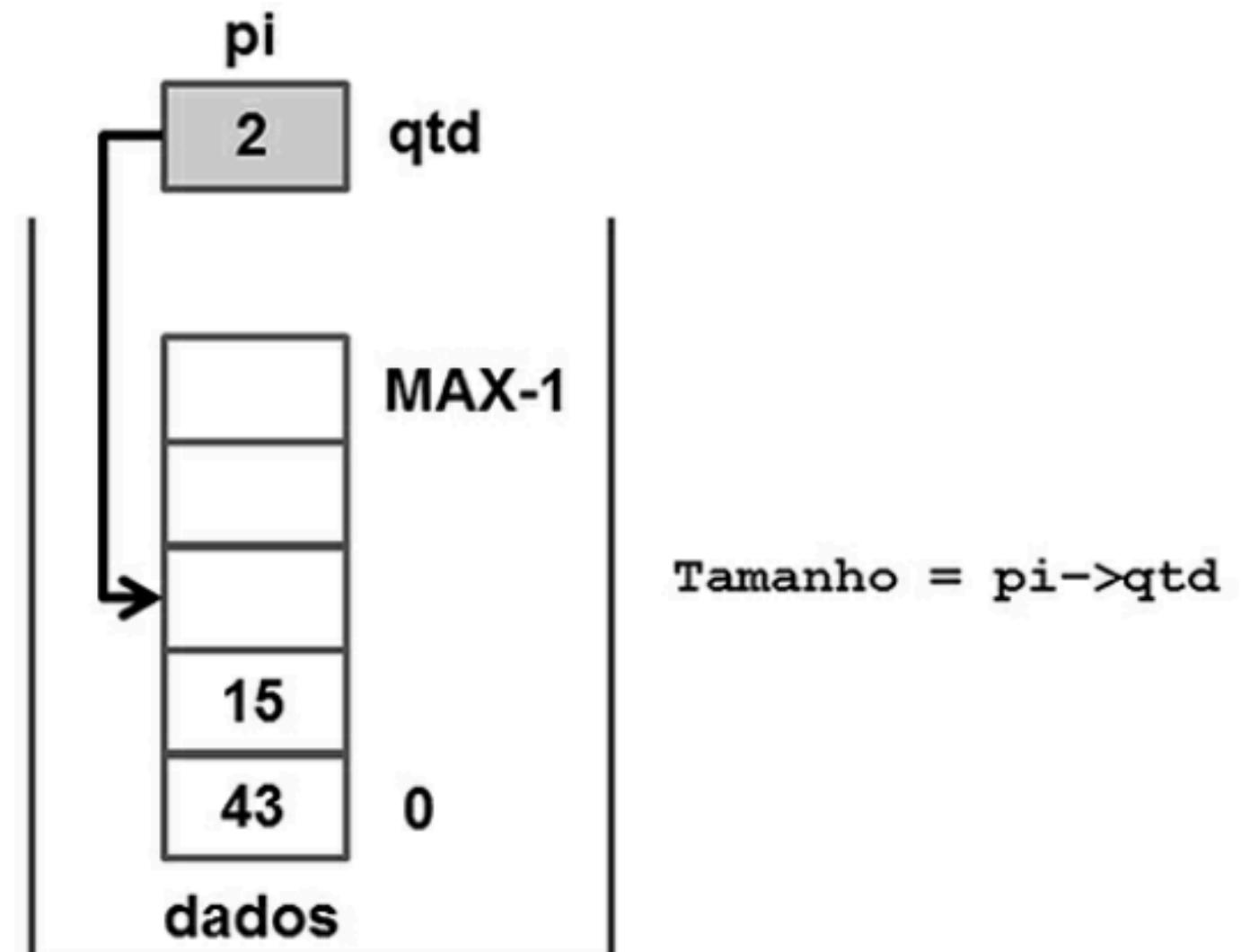
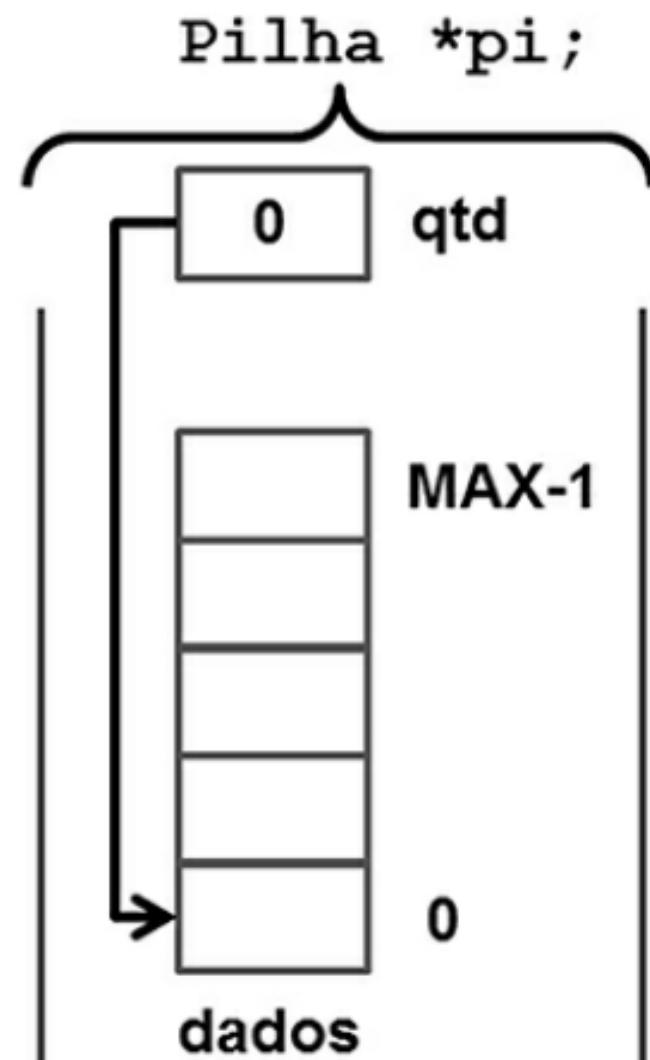
```
01 void libera_Pilha(Pilha* pi) {  
02     free(pi);  
03 }
```

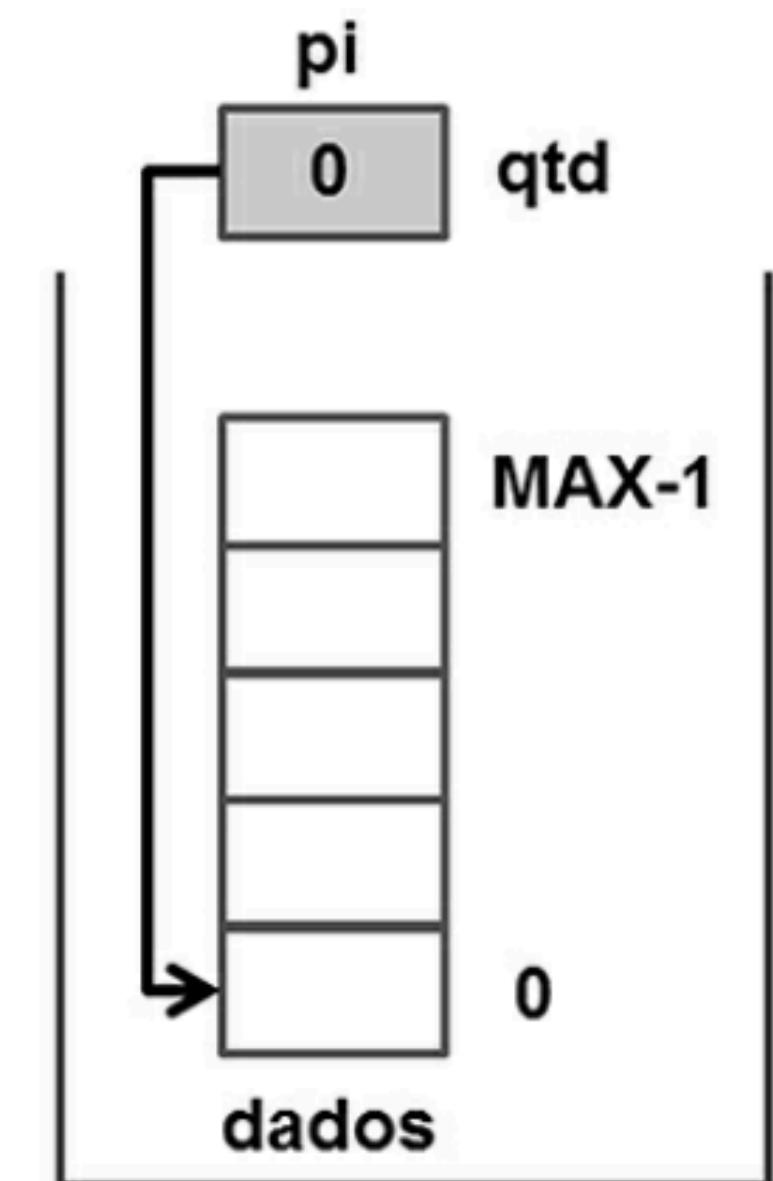
Tamanho da pilha

```
01 int tamanho_Pilha(Pilha* pi) {  
02     if(pi == NULL)  
03         return -1;  
04     else  
05         return pi->qtd;  
06 }
```

Retornando se a pilha está cheia

```
01 int Pilha_cheia(Pilha* pi) {  
02     if(pi == NULL)  
03         return -1;  
04     return (pi->qtd == MAX);  
05 }
```





Pilha Vazia:
 $\text{pi} \rightarrow \text{qtd} == 0$

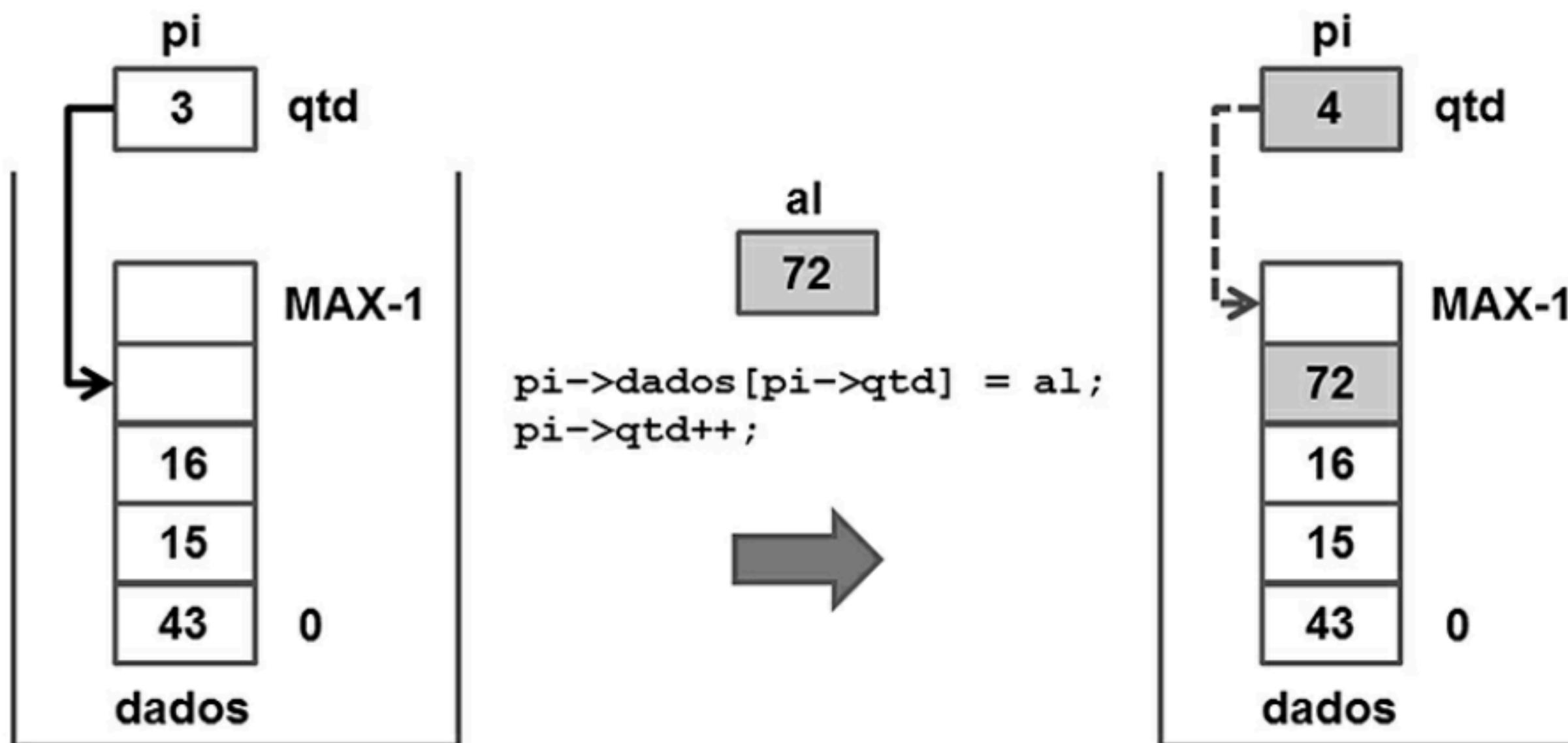
Retornando se a pilha está vazia

```

01 int Pilha_vazia(Pilha* pi) {
02     if(pi == NULL)
03         return -1;
04     return (pi->qtd == 0);
05 }
```

Inserindo um elemento na pilha

```
01 int insere_Pilha(Pilha* pi, struct aluno al) {  
02     if(pi == NULL)  
03         return 0;  
04     if(pi->qtd == MAX) //pilha cheia  
05         return 0;  
06     pi->dados[pi->qtd] = al;  
07     pi->qtd++;  
08     return 1;  
09 }
```

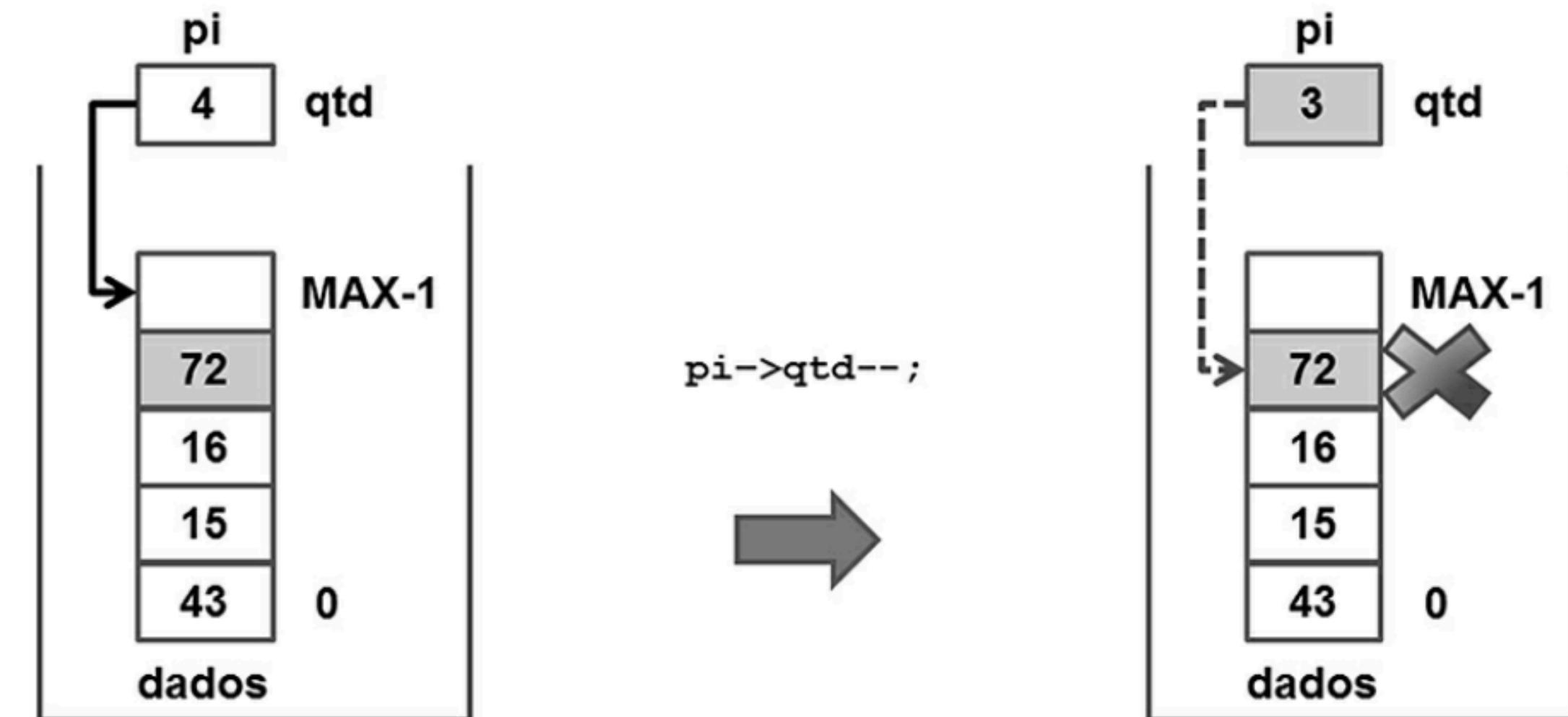


Removendo um elemento da pilha

```
01 int remove_Pilha(Pilha* pi){  
02     if(pi == NULL || pi->qtd == 0)  
03         return 0;  
04     pi->qtd--;  
05     return 1;  
06 }
```

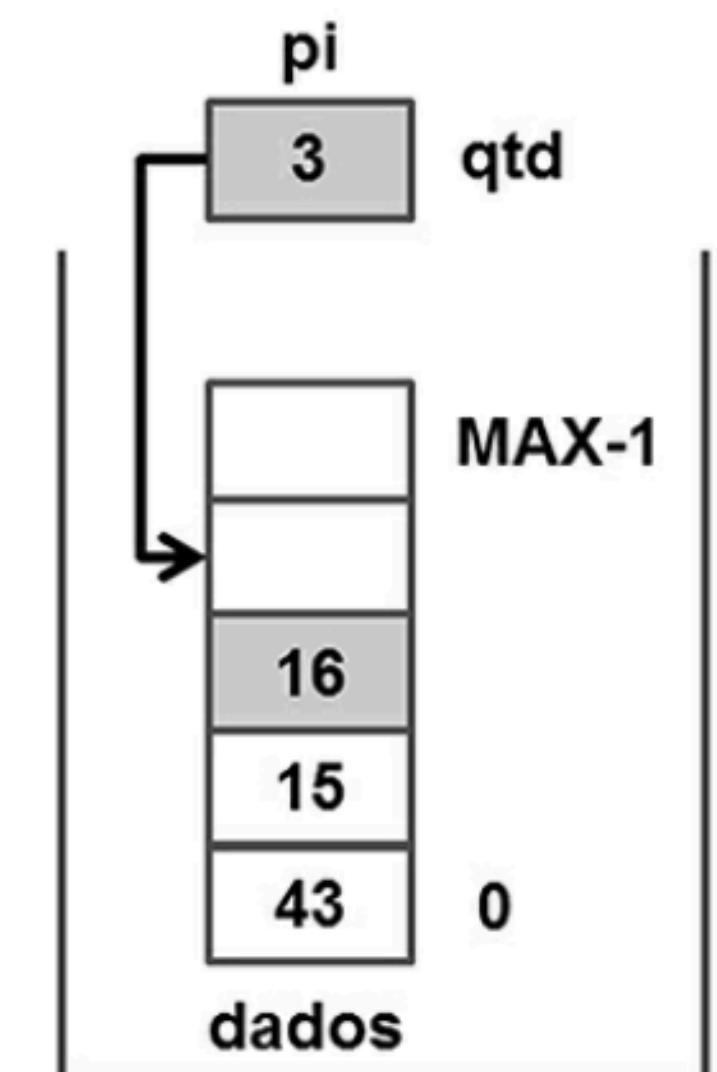


Em uma lista, pode-se acessar e recuperar as informações contidas em qualquer um dos seus elementos. Já em uma pilha, podemos acessar as informações apenas do elemento no **topo** da pilha.



Acessando o topo da pilha

```
01 int acessa_topo_Pilha(Pilha* pi, struct aluno *al){  
02     if(pi == NULL || pi->qtd == 0)  
03         return 0;  
04     *al = pi->dados[pi->qtd-1];  
05     return 1;  
06 }
```



2) PILHA DINÂMICA ENCADEADA

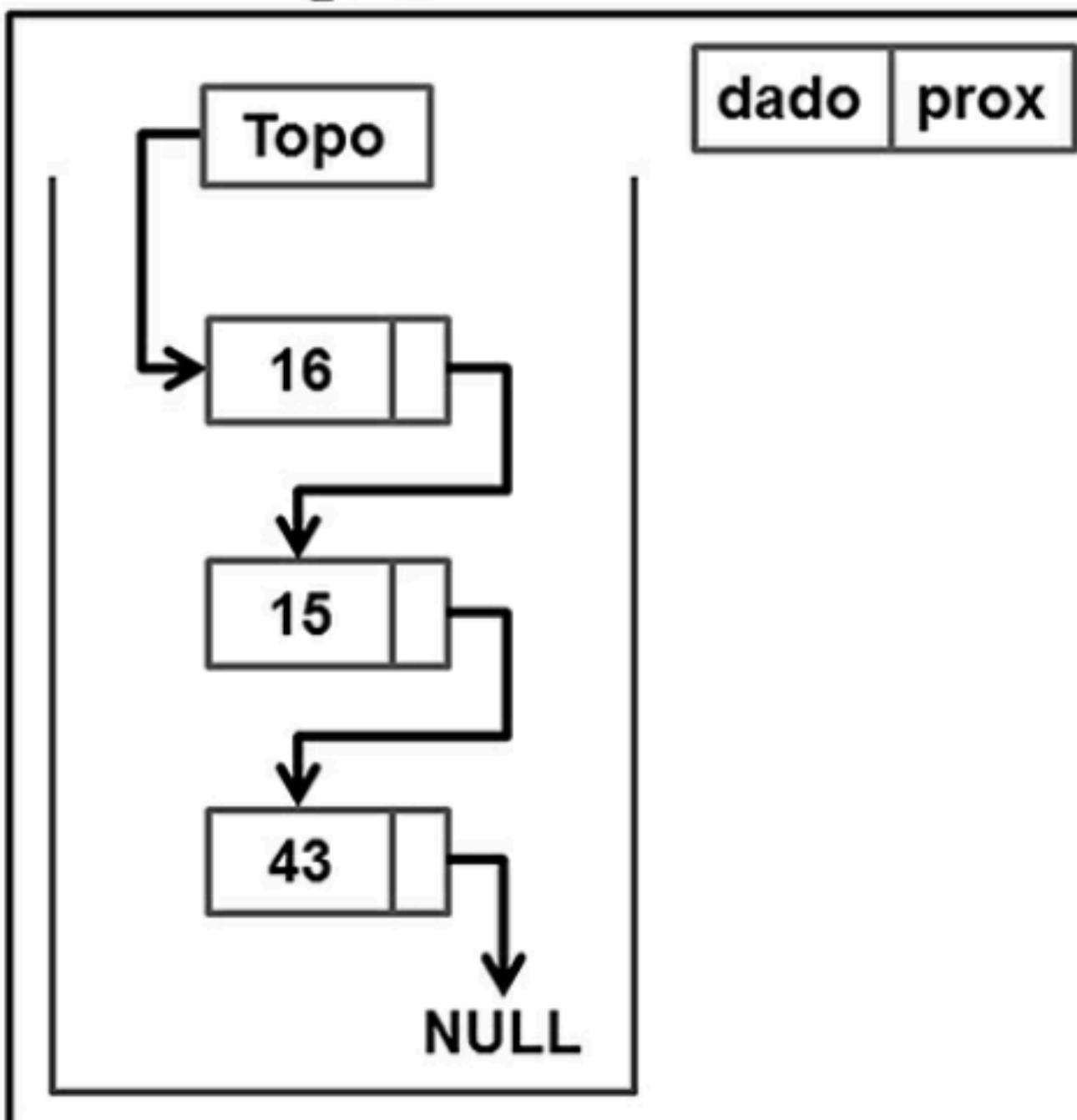
Arquivo PilhaDin.h

```
01 struct aluno{  
02     int matricula;  
03     char nome[30];  
04     float n1,n2,n3;  
05 };  
06 typedef struct elemento* Pilha;  
07  
08 Pilha* cria_Pilha();  
09 void libera_Pilha(Pilha* pi);  
10 int acessa_topo_Pilha(Pilha* pi, struct aluno *al);  
11 int insere_Pilha(Pilha* pi, struct aluno al);  
12 int remove_Pilha(Pilha* pi);  
13 int tamanho_Pilha(Pilha* pi);  
14 int Pilha_vazia(Pilha* pi);  
15 int Pilha_cheia(Pilha* pi);
```

Arquivo PilhaDin.c

```
01 #include <stdio.h>  
02 #include <stdlib.h>  
03 #include "PilhaDin.h" //inclui os protótipos  
04 //Definição do tipo Pilha  
05 struct elemento{  
06     struct aluno dados;  
07     struct elemento *prox;  
08 };  
09 typedef struct elemento Elem;
```

Pilha *pi;



TAREFAS

- 1) Complemente a implementação de fila dinâmica encadeada (arquivo PilhaDin.c).
- 2) Dada uma pilha que armazena caracteres, crie uma função que verifique se uma palavra é um palíndromo.
- 3) Escreva uma função que receba uma string contendo uma expressão aritmética e retorne se ela está com a parentização correta. A função deverá verificar se cada “abre parênteses” tem um “fecha parênteses” correspondente. Exemplos:
 - Correto: (()) -- (() ()) -- () ()
 - Incorreto:) (-- (() (--)) ((



OBRIGADO!

RAFAELVC2@GMAIL.COM