# Pilhas Dinâmicas em C

## INF0286 | INF0447 – Algoritmos e Estruturas de Dados I

**Prof. Me. Raphael Guedes**

raphaelguedes@ufg.br

**2024**

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

# Ponteiro para ponteiro

- Qualquer alteração (seja uma inserção ou remoção) em uma estrutura sequencial estática altera apenas o conteúdo da ***struct*** que define a estrutura de dados, nunca o endereço onde ela se encontra na memória.

- Em uma estrutura dinâmica encadeada, todos os seus elementos são ponteiros alocados dinamicamente e de forma independente. É dentro desse ponteiro que fica armazenada a informação daquele elemento da estrutura.

# Ponteiro para ponteiro

- Numa lista, fila ou pilha dinâmica encadeada não temos mais uma estrutura que define a lista, apenas a estrutura que define os seus elementos. Assim, quando inserimos ou removemos um elemento, estamos mudando o endereço no qual a estrutura se inicia.

- Isso ocorre, pois, quando passamos um ponteiro para uma função (passagem por referência), podemos alterar somente o conteúdo apontado por aquele ponteiro, nunca o endereço guardado dentro dele.

# Ponteiro para ponteiro

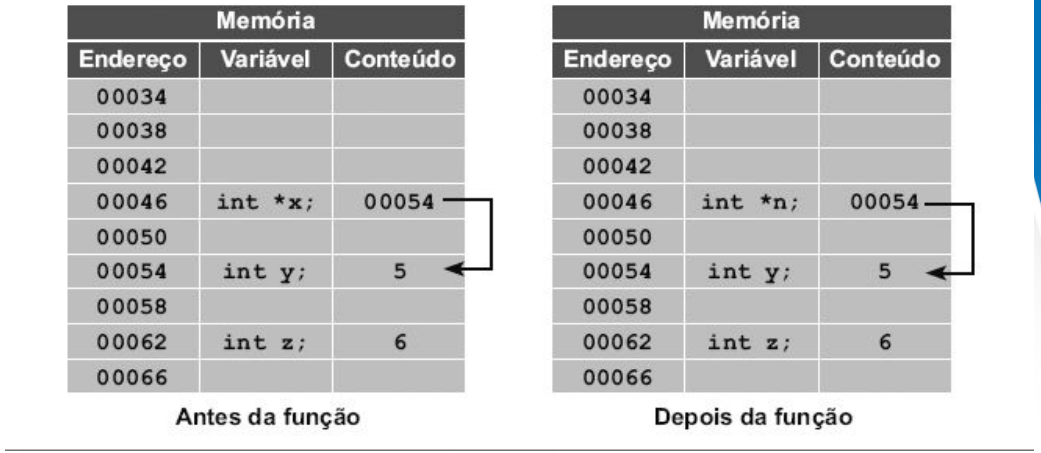
**Trabalhando com ponteiro**

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  void troca_ender(int *a, int *b){
04      a = b;
05      printf("x (Dentro): %#p\n",a);
06  }
07  int main(){
08      int *x, y = 5, z = 6;
09      x = &y;
10      printf("Endereco y: %#p\n",&y);
11      printf("Endereco z: %#p\n",&z);
12      printf("x (Antes): %#p\n",x);
13      troca_ender(x,&z);
14      printf("x (Depois): %#p\n",x);
15      system("pause");
16      return 0;
17  }
```

**Saída**

```
01  Endereco y: 0x03E
02  Endereco z: 0x042
03  x (Antes) : 0x03E
04  x (Dentro): 0x042
05  x (Depois): 0x03E
```

**Memória**

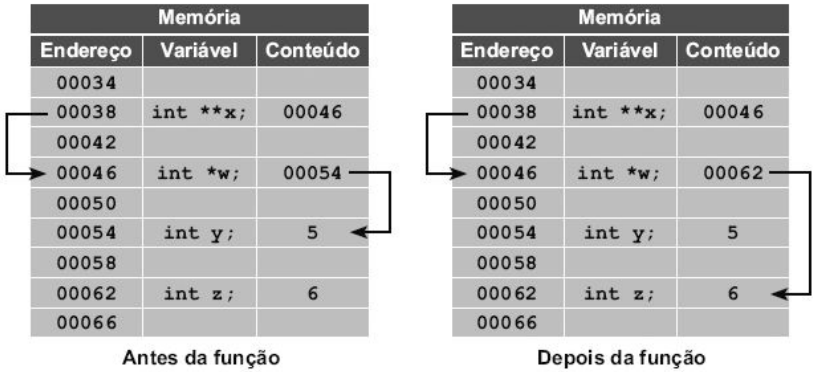


# Ponteiro para ponteiro

**Trabalhando com ponteiro para ponteiro**

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  void troca_ender(int **a, int *b){
04      *a = b;
05      printf("x (Dentro): %#p\n",*a);
06  }
07  int main(){
08      int **x, *w, y = 5, z = 6;
09      x = &w;
10      *x = &y;
11      printf("Endereco y: %#p\n",&y);
12      printf("Endereco z: %#p\n",&z);
13      printf("x (Antes): %#p\n",*x);
14      troca_ender(x,&z);
15      printf("x (Depois): %#p\n",*x);
16      system("pause");
17      return 0;
18  }
```

**Memória**

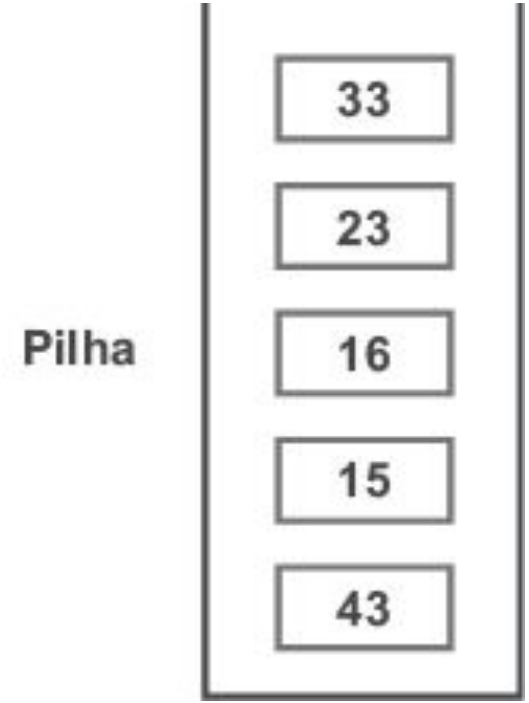



**Saída**

```
01  Endereco y: 0x03E
02  Endereco z: 0x042
03  x (Antes) : 0x03E
04  x (Dentro): 0x042
05  x (Depois): 0x042
```

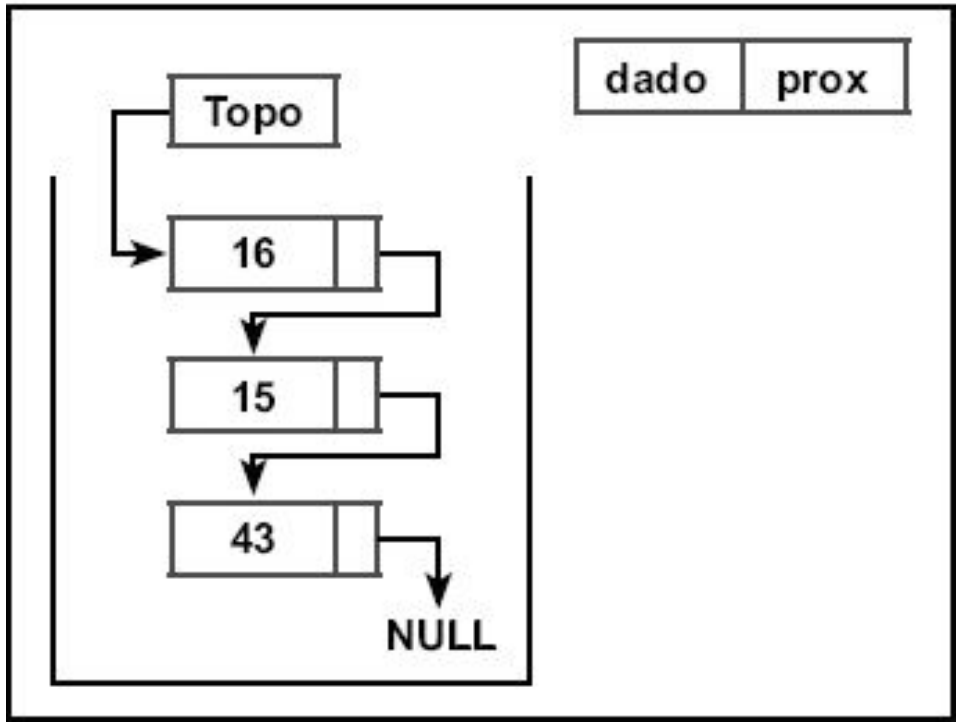
# Pilhas Dinâmicas (LIFO)

- Último a entrar é o primeiro a sair (*Last In, First Out*)
- Primeiro a entrar é o último a sair.



# Pilhas Dinâmicas (LIFO): o tipo

```
Pilha *pi;
```



# Pilhas Dinâmicas (LIFO): estrutura

**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);
```

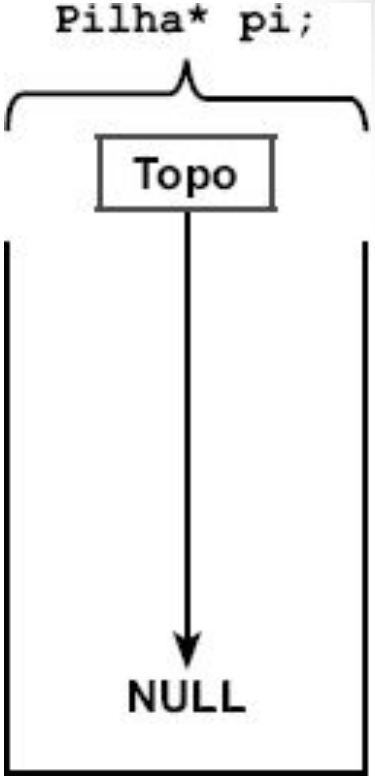
**Arquivo PilhaDin.c**

```
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;
```

# Pilhas Dinâmicas (LIFO): criação

**Criando uma pilha**

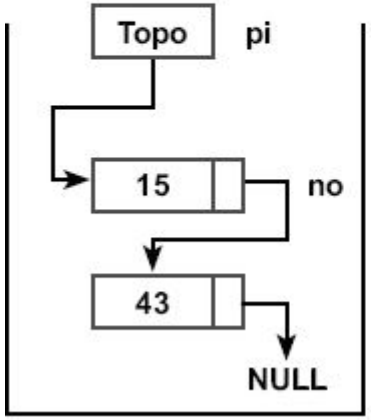
```
01    Pilha* cria_Pilha(){
02        Pilha* pi = (Pilha*) malloc(sizeof(Pilha));
03        if(pi != NULL)
04            *pi = NULL;
05        return pi;
06    }
```



# Pilhas Dinâmicas (LIFO): destruição

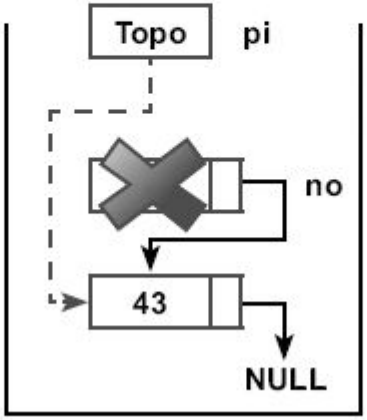


Pilha inicial



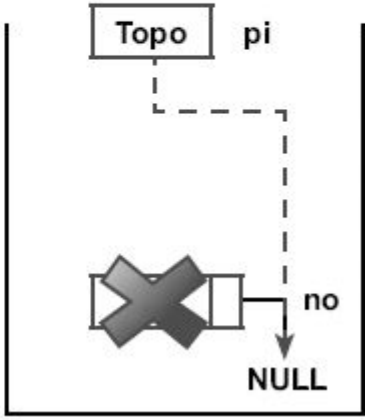

Passo 1:

```
no = *pi;
*pi = (*pi)->prox;
free(no);
```




Passo 2:

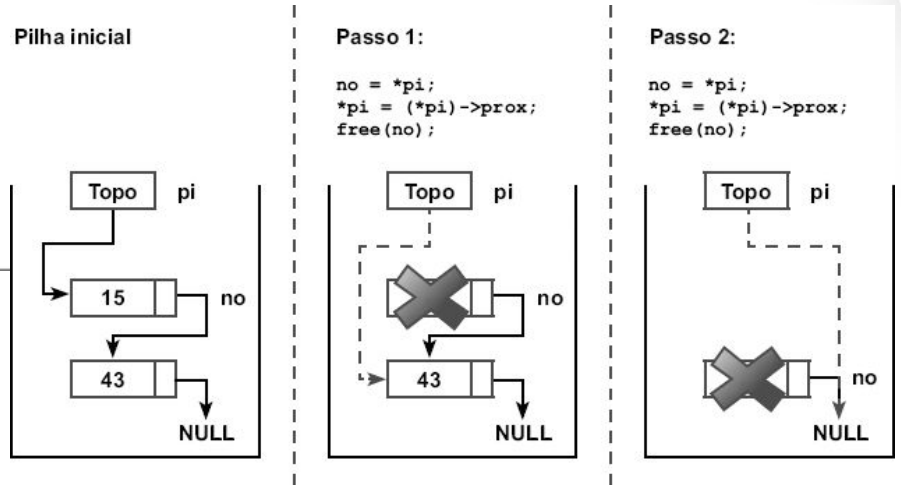
```
no = *pi;
*pi = (*pi)->prox;
free(no);
```



# Pilhas Dinâmicas (LIFO): destruição

**Destruindo uma pilha**

```
01    void libera_Pilha(Pilha* pi){
02        if(pi != NULL){
03            Elem* no;
04            while((*pi) != NULL){
05                no = *pi;
06                *pi = (*pi)->prox;
07                free(no);
08            }
09            free(pi);
10        }
11    }
```

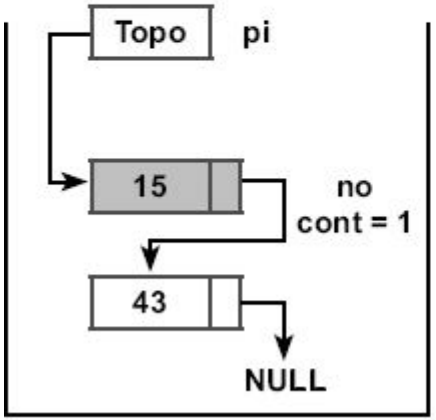


# Pilhas Dinâmicas (LIFO): tamanho



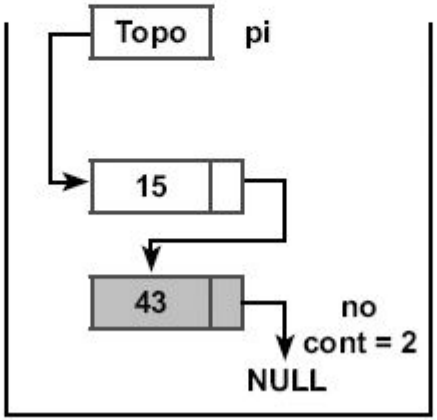
**Passo 1:**

```
cont++;
no = no->prox;
```



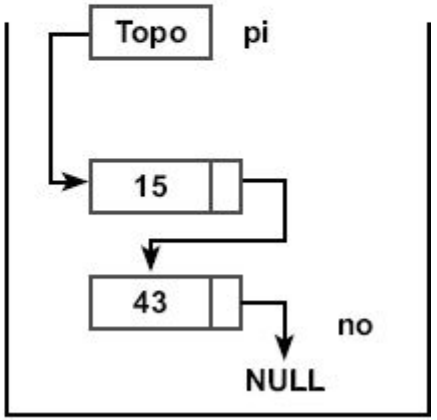

**Passo 2:**

```
cont++;
no = no->prox;
```




**Fim:**

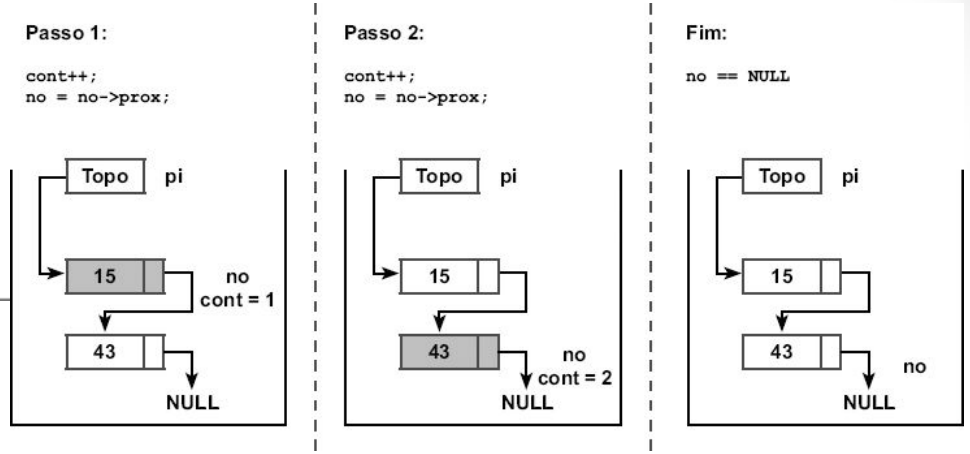
```
no == NULL
```



# Pilhas Dinâmicas (LIFO): tamanho

**Tamanho da pilha**

```
01    int tamanho_Pilha(Pilha* pi){
02        if(pi == NULL)
03            return 0;
04        int cont = 0;
05        Elem* no = *pi;
06        while(no != NULL){
07            cont++;
08            no = no->prox;
09        }
10        return cont;
11    }
```



# Pilhas Dinâmicas (LIFO): pilha cheia

Não teríamos uma abordagem melhor?

Vamos relembrar da pilha estática?

E se tentarmos alocar um nó, resolve?

**Retornando se a pilha está cheia**

```
01    int Pilha_cheia(Pilha* pi){
02        return 0;
03    }
```

# Pilhas Dinâmicas (LIFO): pilha vazia

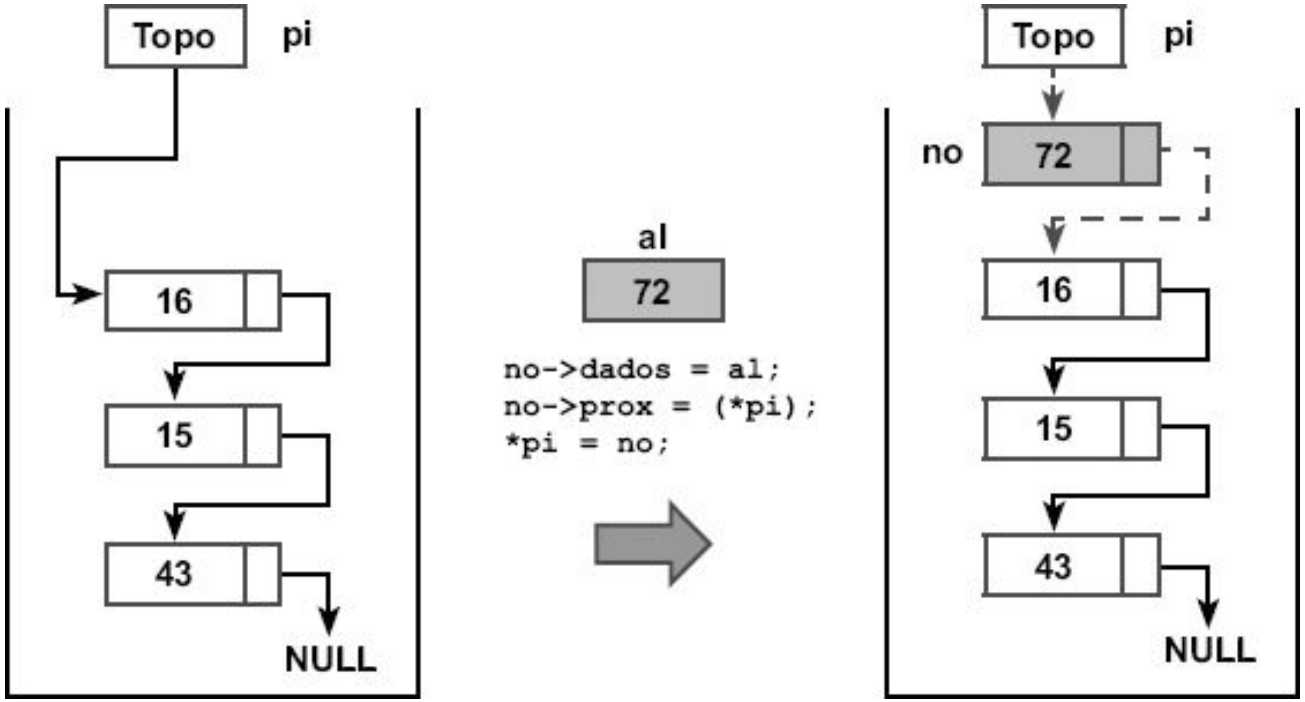
**Retornando se a pilha está vazia**

```
01    int Pilha_vazia(Pilha* pi){
02        if(pi == NULL)
03            return 1;
04        if(*pi == NULL)
05            return 1;
06        return 0;
07    }
```

# Pilhas Dinâmicas (LIFO): inserção

Onde a inserção é feita na pilha?

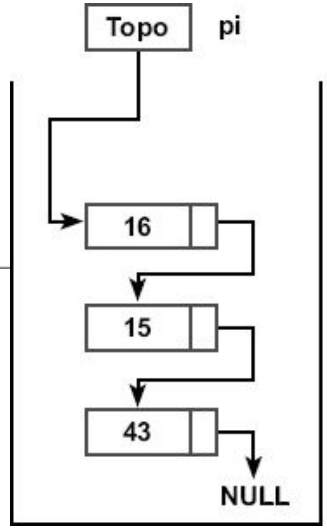
# Pilhas Dinâmicas (LIFO): inserção

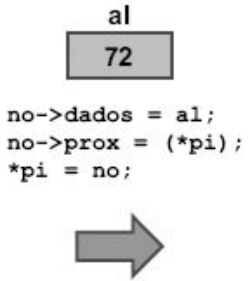


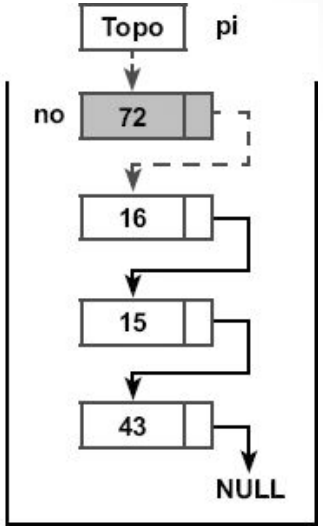
# Pilhas Dinâmicas (LIFO): inserção

**Inserindo um elemento na pilha**

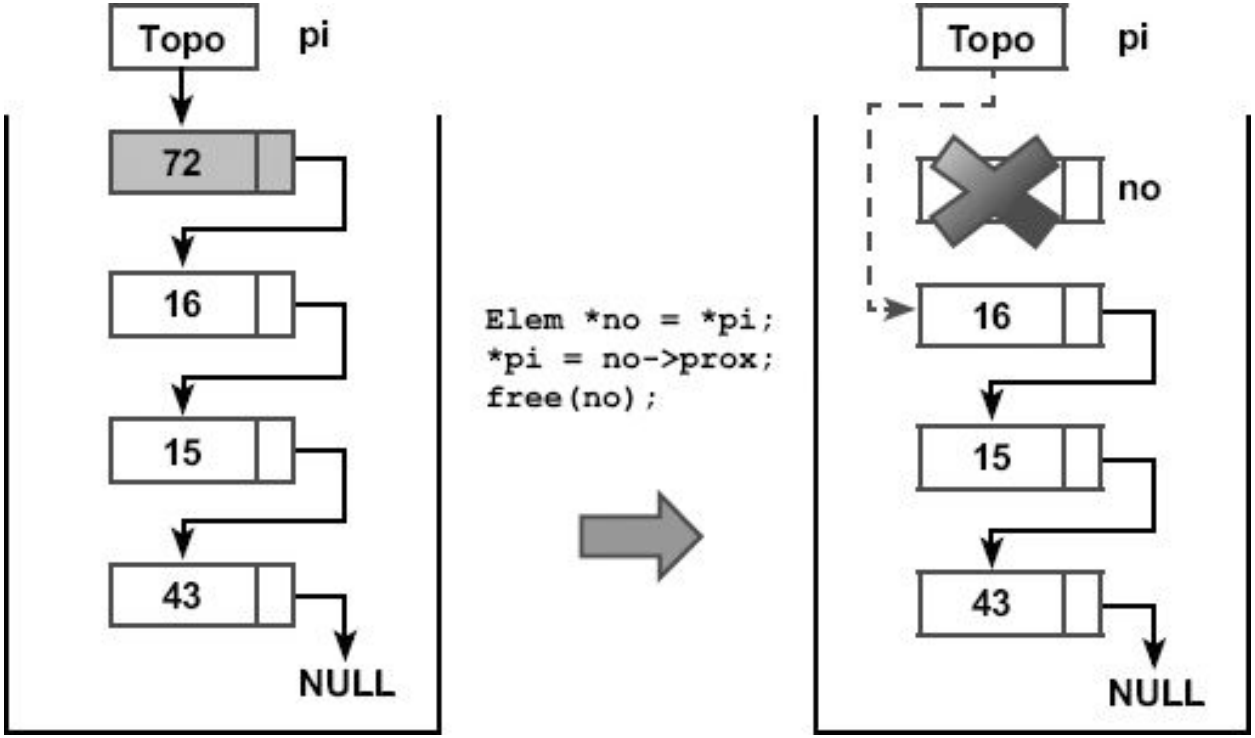
```
01    int insere_Pilha(Pilha* pi, struct aluno al){
02        if(pi == NULL)
03            return 0;
04        Elem* no;
05        no = (Elem*) malloc(sizeof(Elem));
06        if(no == NULL)
07            return 0;
08        no->dados = al;
09        no->prox = (*pi);
10        *pi = no;
11        return 1;
12    }
```



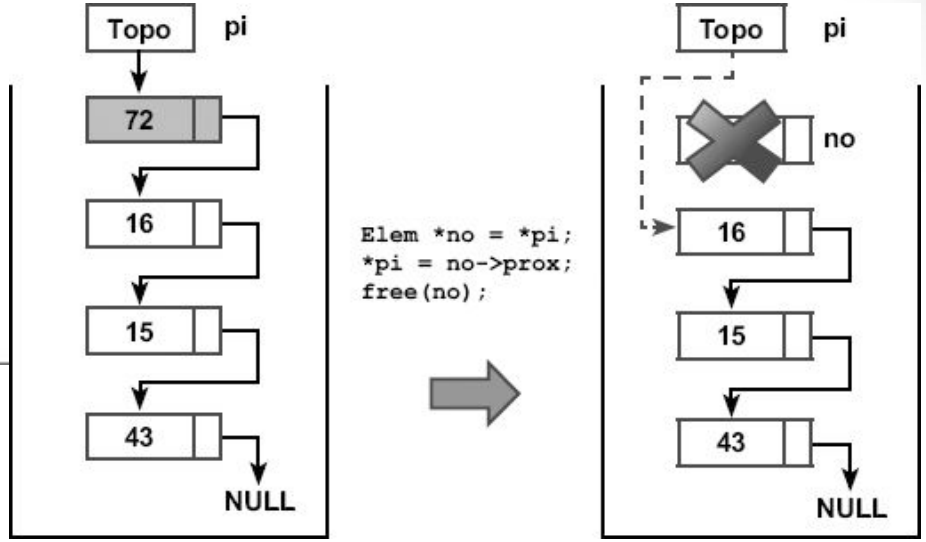
# Pilhas Dinâmicas (LIFO)



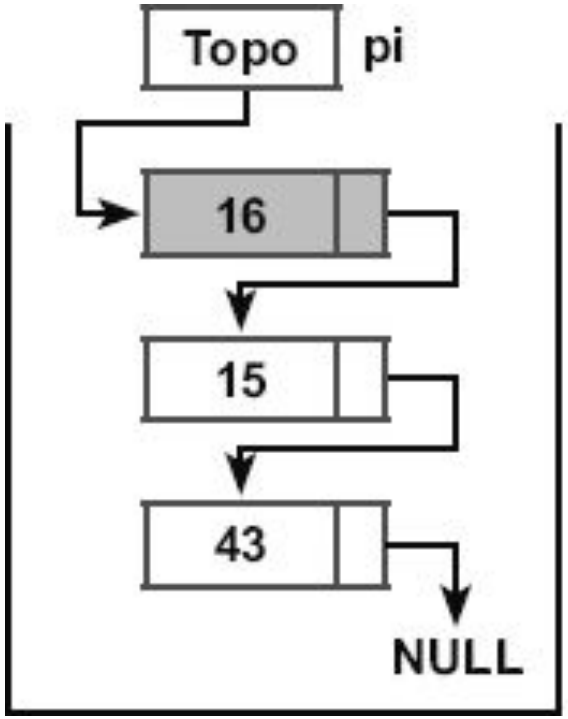
# Pilhas Dinâmicas (LIFO): remoção

**Removendo um elemento da pilha**

```
01    int remove_Pilha(Pilha* pi){
02        if(pi == NULL)
03            return 0;
04        if((*pi) == NULL)
05            return 0;
06        Elem *no = *pi;
07        *pi = no->prox;
08        free(no);
09        return 1;
10    }
```



# Pilhas Dinâmicas (LIFO): acesso



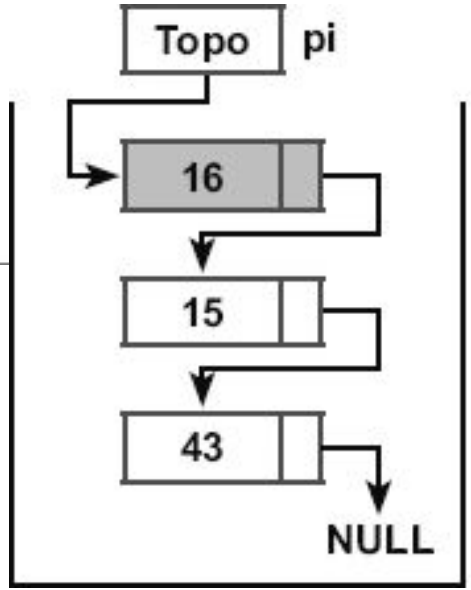

Acesso:

```
*al = (*pi)->dados;
```

# Pilhas Dinâmicas (LIFO): acesso

**Acessando o topo da pilha**

```
01    int acessa_topo_Pilha(Pilha* pi, struct aluno *al){
02        if(pi == NULL)
03            return 0;
04        if((*pi) == NULL)
05            return 0;
06        *al = (*pi)->dados;
07        return 1;
08    }
```




**Acesso:**

```
*al = (*pi)->dados;
```

# Referências


- BACKES, André Ricardo. **Algoritmos e Estruturas de Dados em C**. Rio de Janeiro: LTC, 2023.

# Obrigado!

raphaelguedes@ufg.br
raphaelguedes@inf.ufg.br

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS