Estrutura de Dados

Estrutura Pilha

Prof. Luiz Gustavo Almeida Martins e
adaptado pela Profa. Gina M. B. Oliveira

Introdução

- **Pilha** é uma lista linear que respeita a política de acesso **LIFO** (***Last In, First Out***)
  - Elementos removidos na ordem inversa da inserção

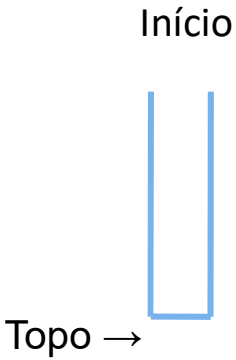
    Analogia: Pilha de pratos

- Estrutura de dados mais simples e a mais utilizada em programação

- Todo acesso a elementos deve ser feito pelo topo da pilha.

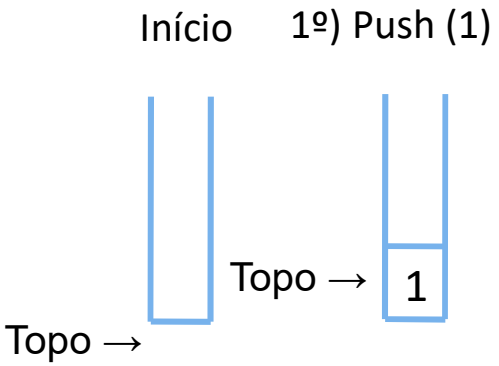
Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



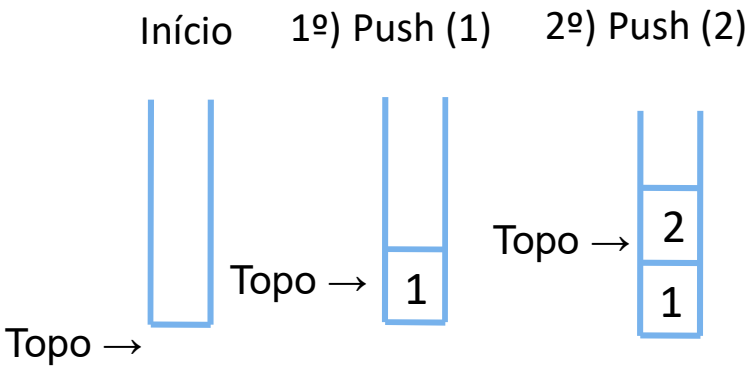
Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



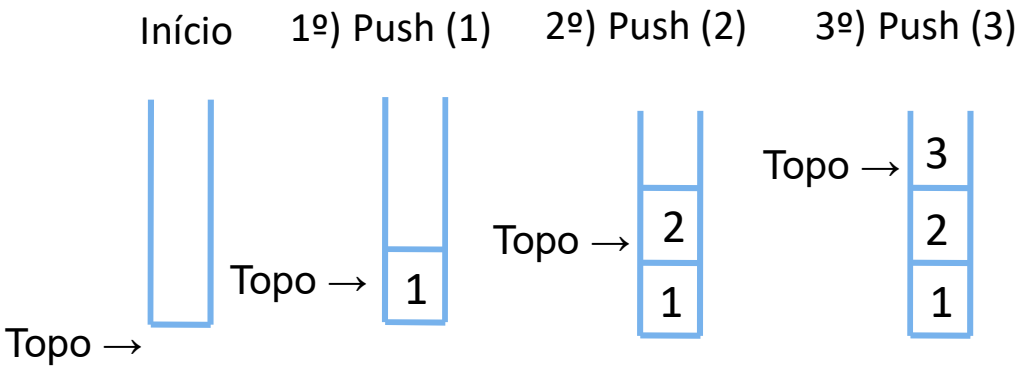
Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



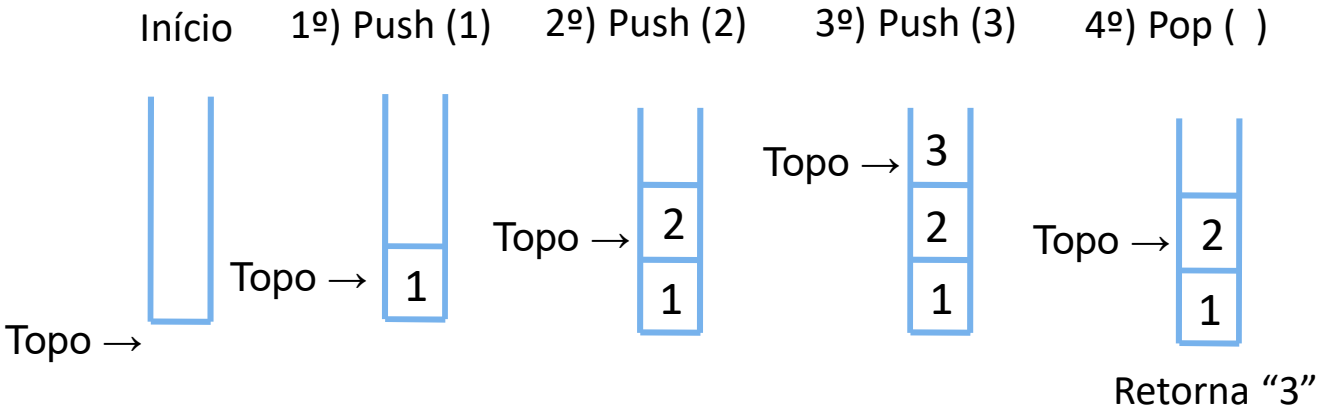
Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



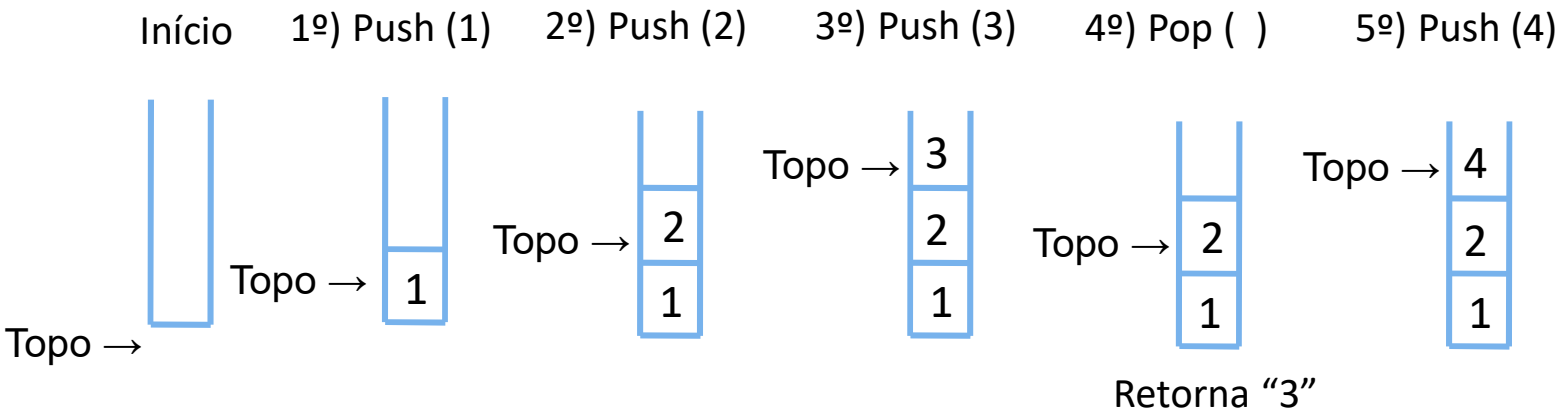
Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



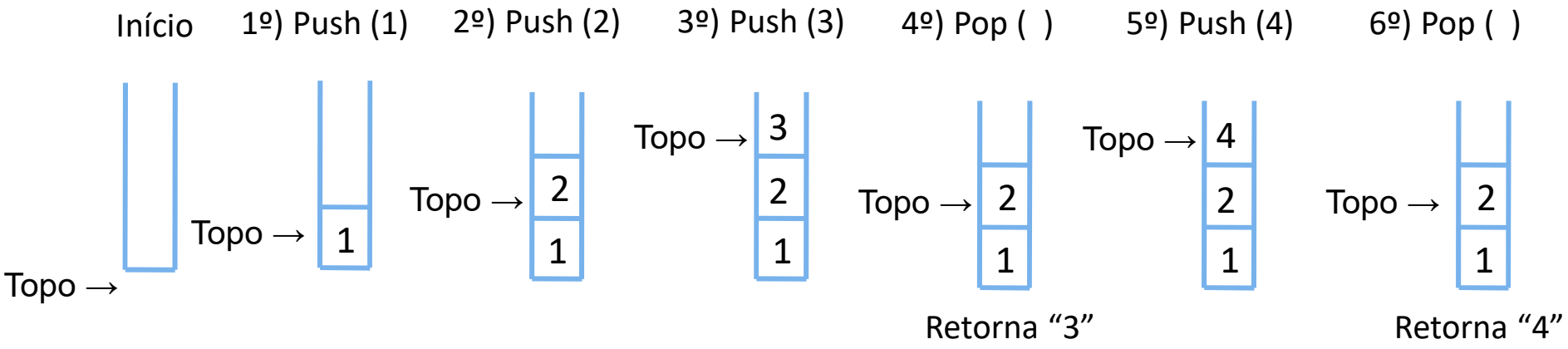
Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



Introdução

- **Principais operações básicas:**
  - ***Push*: empilhar** um novo elemento no topo
  - ***Pop*: desempilhar e retornar** o elemento do topo



TAD Pilha

- **Cabeçalho:**
  - **Nome: Pilha**
  - **Tipo de dado:** número inteiro
  - **Lista de operações:** cria_pilha, pilha_vazia, pilha_cheia, empilha (***push***), desempilha (***pop***), le_topo

TAD Pilha

- **Operação Cria_Pilha:**
  - **Entrada:** nenhuma
  - **Pré-condição:** nenhuma
  - **Processo:** cria uma pilha e a coloca no estado de pilha vazia
  - **Saída:** endereço da pilha criada
  - **Pós-condição:** nenhuma

TAD Pilha

- **Operação Pilha_Vazia:**
  - **Entrada:** endereço da pilha
  - **Pré-condição:** nenhuma
  - **Processo:** verifica se a pilha está vazia
  - **Saída:** retorna 1 se pilha vazia ou 0 caso contrário
  - **Pós-condição:** nenhuma

TAD Pilha

- **Operação Pilha_Cheia:**
  - **Entrada:** endereço da pilha
  - **Pré-condição:** nenhuma
  - **Processo:** verifica se a pilha está cheia
  - **Saída:** retorna 1 se pilha cheia ou 0 caso contrário
  - **Pós-condição:** nenhuma

TAD Pilha

- **Operação Empilha (*push*):**
  - **Entrada:** endereço da pilha e o elemento a ser inserido
  - **Pré-condição:** pilha não estar cheia
  - **Processo:** inserir o elemento informado no topo da pilha
  - **Saída:** retorna 1 se a operação foi bem sucedida ou 0 caso contrário
  - **Pós-condição:** a pilha de entrada com um elemento a mais

TAD Pilha

- **Operação Desempilha (*pop*):**
  - **Entrada:** endereço da pilha e o endereço de retorno do elemento do topo da pilha
  - **Pré-condição:** pilha não estar vazia
  - **Processo:** remover o elemento que está no topo da pilha e retorná-lo
  - **Saída:** retorna 1 se a operação foi bem sucedida ou 0 caso contrário
  - **Pós-condição:** a pilha de entrada com um elemento a menos

TAD Pilha

- **Operação Le_Topo:**
  - **Entrada:** endereço da pilha e o endereço de retorno do elemento do topo da pilha
  - **Pré-condição:** pilha não estar vazia
  - **Processo:** retornar o valor do elemento que está no topo da pilha **SEM removê-lo**
  - **Saída:** retorna 1 se a operação foi bem sucedida ou 0 caso contrário
  - **Pós-condição:** nenhuma

Implementação Estática/Sequencial

- **Forma de representação:**
  - Utiliza a **MESMA** estrutura de representação da **lista linear**

Implementação Estática/Sequencial

- **Forma de representação:**
  - Utiliza a **MESMA** estrutura de representação da **lista linear**

Exemplo: pilha de inteiros

pilha.c

```
# define max 20
struct pilha {
        int vetor [max];
        int topo;
};
```

pilha.h

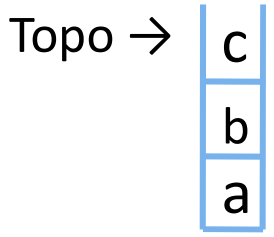
```
typedef struct pilha * Pilha;
```

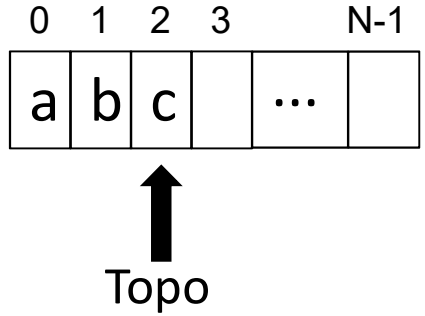
Implementação Estática/Sequencial

- Dinâmica de controle do campo ***topo***:
  - Indicar **1ª posição livre** (adotada em lista)
  - Indicar **última posição ocupada**

- Usaremos ***topo*** indicando a **última posição ocupada**

Exemplo:



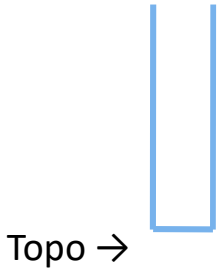


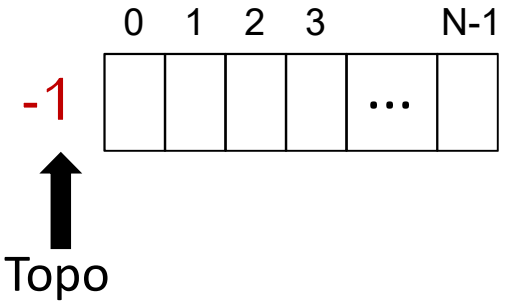
Implementação Estática/Sequencial

- **Pilha Vazia:**

  – TOPO aponta para uma **posição inválida**
    - Usar **-1** facilita operações, pois só precisa incrementar para indicar **próxima posição livre** (**1ª posição do vetor**)

- **Exemplo:**





Implementação Estática/Sequencial

- **Operações básicas:**
  - Cria_Pilha
  - Pilha_Vazia
  - Pilha_Cheia
  - Empilha (***push***)
  - Desempilha (***pop***)
  - Le_Topo

Implementação Estática/Sequencial

- **Operação cria_pilha:**
  - Aloca estrutura pilha
  - Coloca a pilha no estado de vazia
  - Retorna o endereço da pilha alocada

```
Pilha cria_pilha () {
        Pilha p;
        p = (Pilha) malloc (sizeof (struct pilha));
        if (p != NULL)
            p->topo = -1;
        return p;
}
```

Implementação Estática/Sequencial

- **Operação pilha_vazia:**
  – Verifica se a pilha está na condição de vazia

```
int pilha_vazia (Pilha p) {
    if (p->topo == -1)
        return 1;
    else
        return 0;
}
```

Implementação Estática/Sequencial

- **Operação pilha_cheia:**
  – Verifica se a pilha está na condição de cheia

```
int pilha_cheia (Pilha p) {
    if (p->topo == max-1)
        return 1;
    else
        return 0;
}
```

Implementação Estática/Sequencial

- **Operação empilha (*push*):**
  - Incrementa o indicador de topo
  - Insere o elemento no topo da pilha

```
int push (Pilha p, int elem) {
    if (p == NULL || pilha_cheia(p) == 1)
        return 0;
    // Insere o elemento no topo
    p->topo++;
    p->no[p->topo] = elem;
    return 1;
}
```

Implementação Estática/Sequencial

- **Operação desempilha (*pop*):**
  - Remove o elemento do topo da pilha
    - Decrementa o indicador de topo
  - Retorna o valor do elemento (por referência)

```
int pop (Pilha p, int *elem) {
    if (p == NULL || pilha_vazia(p) == 1)
        return 0;
    *elem = p->no[p->topo]; // Retorna o elemento
    p->topo--;          // Remove o elemento do topo
    return 1;
}
```

Implementação Estática/Sequencial

- **Operação le_topo:**
  - Retorna o valor do elemento do topo da pilha
  - Mesmo código da ***pop()*** sem a parte de remoção do elemento **(decremento do topo)**

```
int le_topo (Pilha p, int *elem) {
    if (p == NULL || pilha_vazia(p) == 1)
        return 0;
    *elem = p->no[p->topo]; // Retorna o elemento
    return 1;
}
```

Implementação Dinâmica/Encadeada

- **Forma de representação:**
  - Utiliza a **MESMA** estrutura de representação da **lista linear**

Implementação Dinâmica/Encadeada

- **Forma de representação:**
  - Utiliza a **MESMA** estrutura de representação da **lista linear**

Exemplo: pilha de inteiros

pilha.c

```
struct no {
        int info;
        struct no* prox;
};
```

pilha.h

```
typedef struct no * Pilha;
```

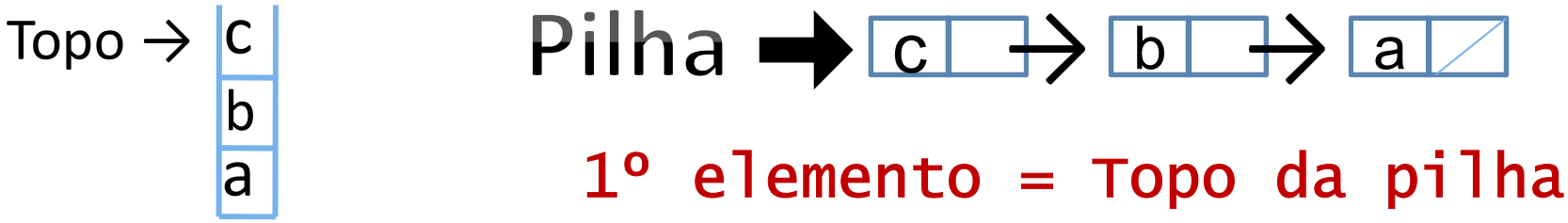
Implementação Dinâmica/Encadeada

- Dinâmica de controle do ***topo***:
  - Ponteiro do **tipo Pilha** **aponta para o topo**

Implementação Dinâmica/Encadeada

- Dinâmica de controle do ***topo***:
  - Ponteiro do **tipo Pilha** **aponta para o topo**

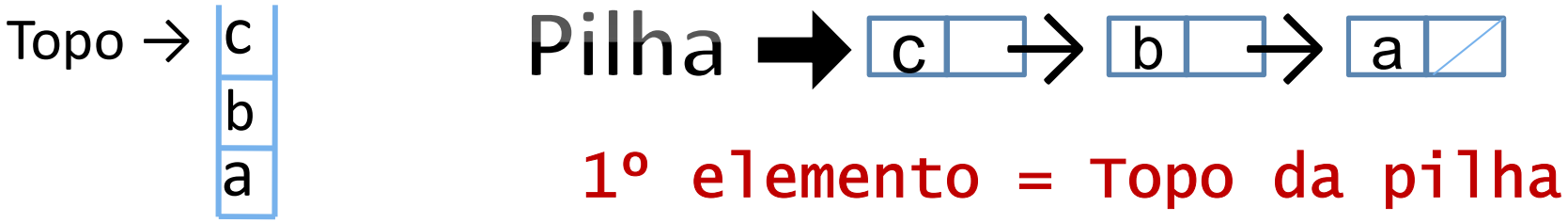
Exemplo:



Implementação Dinâmica/Encadeada

- Dinâmica de controle do ***topo***:
  - Ponteiro do **tipo Pilha** **aponta para o topo**

Exemplo:



Pilha vazia:

Implementação Dinâmica/Encadeada

- **Operações básicas:**
  - Cria_Pilha
  - Pilha_Vazia
  - Pilha_Cheia
  - Empilha (***push***)
  - Desempilha (***pop***)
  - Le_Topo

Implementação Dinâmica/Encadeada

- **Operação cria_pilha:**
  - Retorna uma pilha no estado de vazia
    - Topo igual a ***NULL***
  - Retorna o **endereço do topo** (***NULL***)

```
Pilha cria_pilha () {
        return NULL;
}
```

Implementação Dinâmica/Encadeada

- **Operação pilha_vazia:**
  - Verifica se a pilha está no estado de vazia
    - Ponteiro da Pilha igual a ***NULL***

```
int pilha_vazia (Pilha p) {
        if (p == NULL)
            return 1;
        else
            return 0;
}
```

Implementação Dinâmica/Encadeada

- **Operação pilha_cheia:**
  - **Não existe pilha cheia** na implementação dinâmica/encadeada
  - Tamanho da pilha é **limitada pelo espaço de memória**

Implementação Dinâmica/Encadeada

- **Operação empilha (*push*):**
  - Aloca um novo nó
  - Preenche os campos do novo nó
    - Campo ***info*** recebe o valor do elemento
    - Campo ***prox*** recebe o endereço do topo da pilha
  - Faz a pilha apontar para o novo nó

- **SIMILAR** à operação ***insere_elem()*** da lista
  - Ambas inserem o elemento **no início da estrutura** (**1º nó = topo da pilha**)

Implementação Dinâmica/Encadeada

- **Operação empilha (*push*):**

```
int push (Pilha *p, int elem) {
    Pilha N = (Pilha) malloc(sizeof(struct no));
    if (N == NULL)
        return 0;
    N->info = elem;
    N->prox = *p;
    *p = N;
    return 1;
}
```

Implementação Dinâmica/Encadeada

- **Operação desempilha (*pop*):**
  - Remove o elemento que está no ***topo*** da Pilha
    - Pilha passa a apontar para o sucessor do ***topo***
    - Libera memória alocada pelo antigo ***topo***
  - *Retorna o valor do elemento removido*
    - *Valor é armazenado na **variável de retorno***

Implementação Dinâmica/Encadeada

- **Operação desempilha (*pop*):**

```
int pop (Pilha *p, int *elem) {
    if (pilha_vazia(*p) == 1)
        return 0;
    Pilha aux = *p;
    *elem = aux->info;
    *p = aux->prox;
    free(aux);
    return 1;
}
```

Implementação Dinâmica/Encadeada

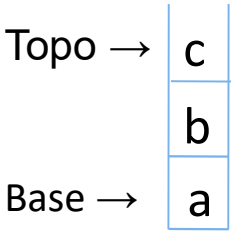
- **Operação le_topo:**
  - Simplificação do código da operação ***pop()***
    - Retorna o elemento **sem removê-lo**

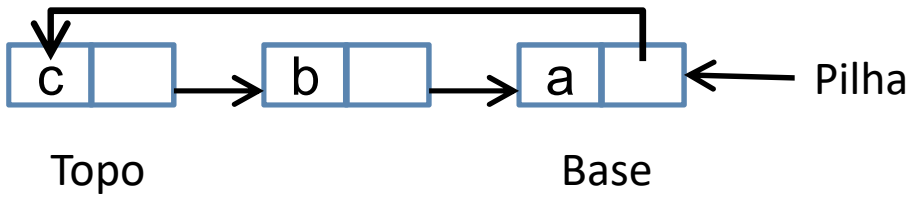
```
int le_topo (Pilha *p, int *elem) {
    if (pilha_vazia(*p) == 1)
        return 0;
    *elem = (*p)->info;
    return 1;
}
```

Outra Forma de Implementação

- Podemos utilizar o **encadeamento circular**
  - Pilha aponta para o **último nó** (**BASE da Pilha**)
  - ***Topo*** da pilha **é o sucessor do último nó**

- **Representação:**





- As operações para a implementação encadeada circular são mais complexas

Exercícios

1. *Implementar, utilizando a implementação **estática/sequencial**, o TAD pilha de números inteiros. Essa implementação deve contemplar as operações básicas: criar_pilha, pilha_vazia, pilha_cheia, push, pop e le_topo. Além disso, desenvolva um programa aplicativo que permita ao usuário criar uma pilha, empilhar e desempilhar elementos, e imprimir a pilha.*

   Teste este programa com a seguinte seqüencia de operações:

   - *Inicialize a pilha*
   - *Imprima a pilha*
   - *Empilhar os elementos {4,8,-1,19,2,7,8,5,9,22,45};*
   - *Imprima a pilha*
   - *Desempilhar*
   - *Imprima a pilha*
   - *Lê o elemento do topo*
   - *Imprima a pilha*

2. *Refaça o exercício anterior, utilizando a implementação **dinâmica/encadeada**.*

Referências

- *Backes, André, Linguagem C Descomplicada, portal de vídeo-aulas, https://programacaodescomplicada.wordpress.com/, acessado em 09/03/2016.*

- *Celes, W., Cerqueira, R. e Rangel, J. L. Introdução a estruturas de dados. Ed. Campus Elsevier, 2004.*