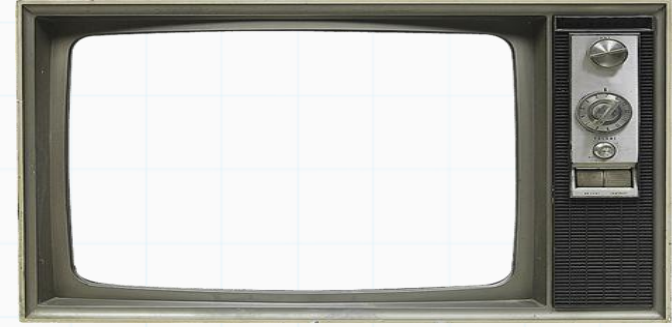


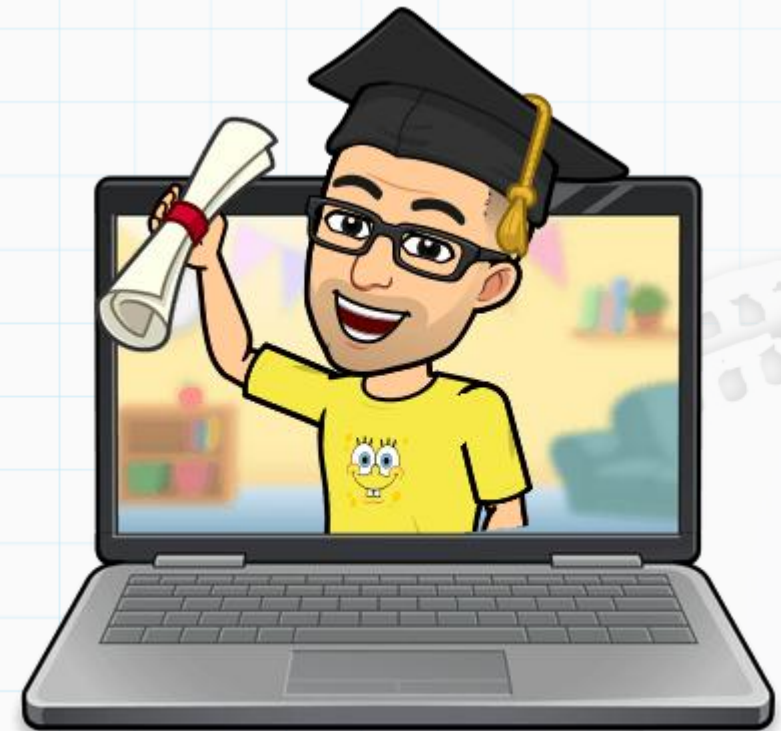
Programação Estruturada

Professor : Yuri Frota

yuri@ic.uff.br

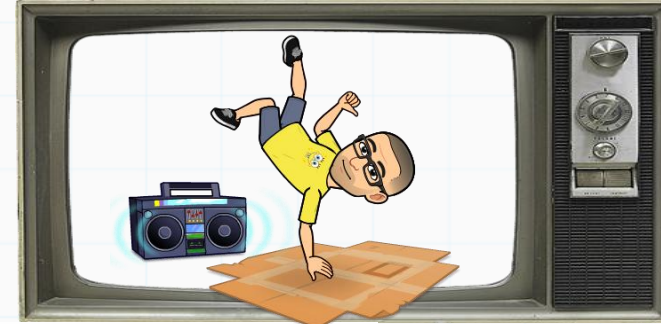


- Utilize os arquivos fornecidos main.c e tad.c com o TAD básico de pilhas em vetores para fazer as questões a seguir

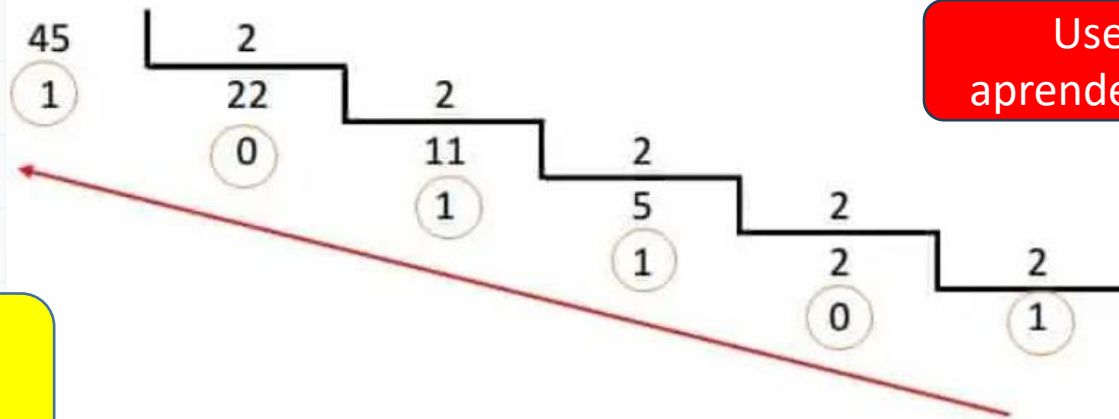


TAD - Pilha

1) Binário: Dado um número X decimal sabemos que podemos convertê-lo para binário realizando a divisão sucessiva por 2 (base do sistema binário). O novo número binário será dado pelo último quociente e o agrupamento dos restos de divisão na ordem inversa.



Veja exemplo: para X=45 o binário seria 1011101



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC) QUE NÃO SEJA UMA PILHA!

Faça uma função que dado um inteiro decimal X, imprima sua conversão para binário.

Veja exemplo de execução:

O número binário será dado pelo último quociente e o agrupamento dos restos de divisão.

Lembre-se:

```
int x;
```

```
x%2; // resto da divisão inteira
```

```
x/2; // divisão inteira
```

TAD - Pilha

1) Binário

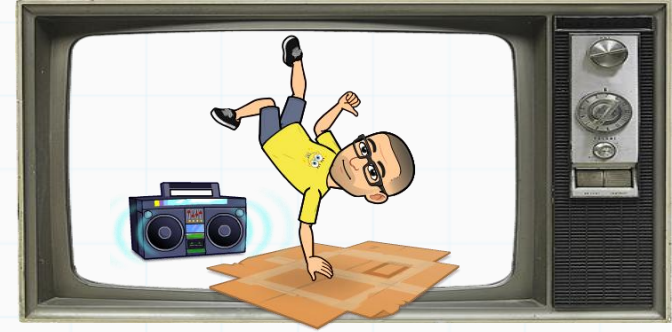
Código da main.c

```
int main()
{
    decimal_binario(45);
    decimal_binario(109);
    decimal_binario(2015);

    return 0;
}
```

Exemplo de execução:

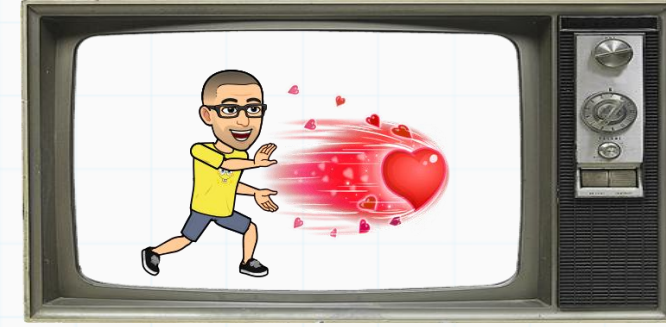
```
Numero binario: 101101
Numero binario: 1101101
Numero binario: 11111011111
```



TAD - Pilha

2) Altera: Implemente uma função que dado uma pilha, e dois inteiros k e el , altere a posição k da pilha para o valor el .

Exemplo: para $k=1$ e $el=300$

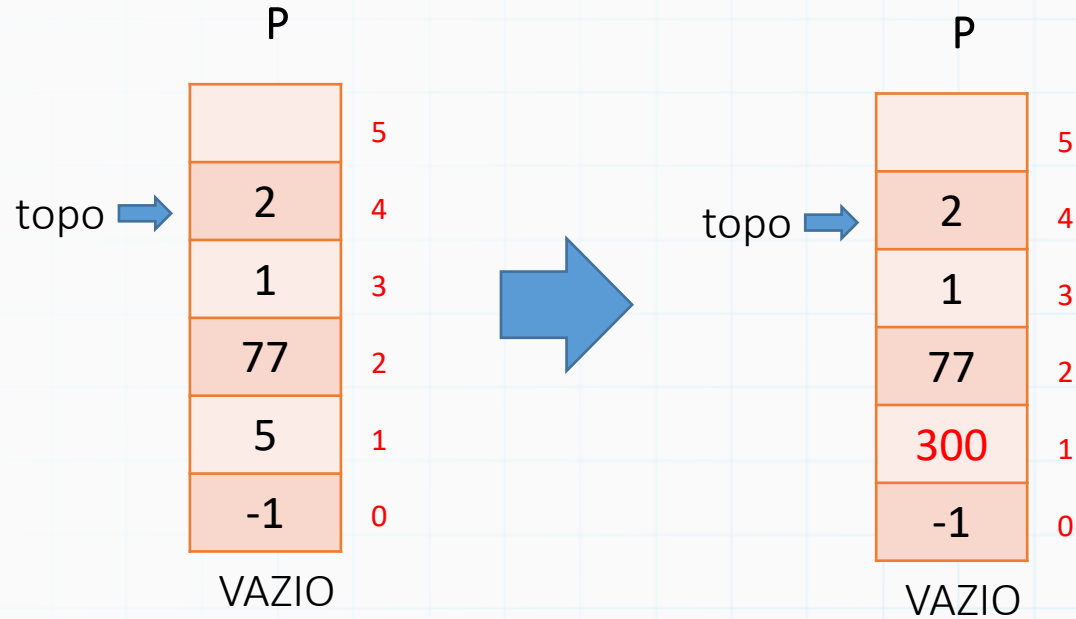


Use só o que aprendemos até hoje

A função deve apenas interagir com a pilha com as funções de pilha:



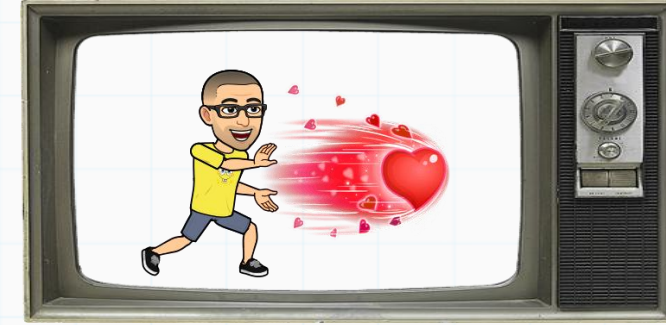
```
void push_pilha(pilha * P, int el)
int pop_pilha(pilha * P)
int top_pilha(pilha * P)
int vazia_pilha(pilha * P)
int tam_pilha(pilha * P)
void libera_pilha(pilha * P)
void cria_pilha(pilha * P, int tam)
```



Veja exemplo de execução:

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC) MAS PODEMOS CRIAR E USAR UMA PILHAS AUXILIARES DE DENTRO DA FUNÇÃO.

TAD - Pilha



2) Altera

Código da main.c

```
int main()
{
    pilha P;
    cria_pilha(&P, 10);

    push_pilha(&P, 8);
    push_pilha(&P, 5);
    push_pilha(&P, 77);
    push_pilha(&P, 1);
    push_pilha(&P, 2);
    imprime_pilha(&P);

    altera_k_el(&P, 2, 800);
    imprime_pilha(&P);
    altera_k_el(&P, 0, 333);
    imprime_pilha(&P);
    altera_k_el(&P, 5, 555);
    imprime_pilha(&P);

    return 0;
}
```

Exemplo de execução:

```
| 2 | <- topo
| 1 |
| 77 |
| 5 |
| 8 |

| 2 | <- topo
| 1 |
| 800 |
| 5 |
| 8 |

| 2 | <- topo
| 1 |
| 800 |
| 5 |
| 333 |

posicao irregular
| 2 | <- topo
| 1 |
| 800 |
| 5 |
| 333 |
```

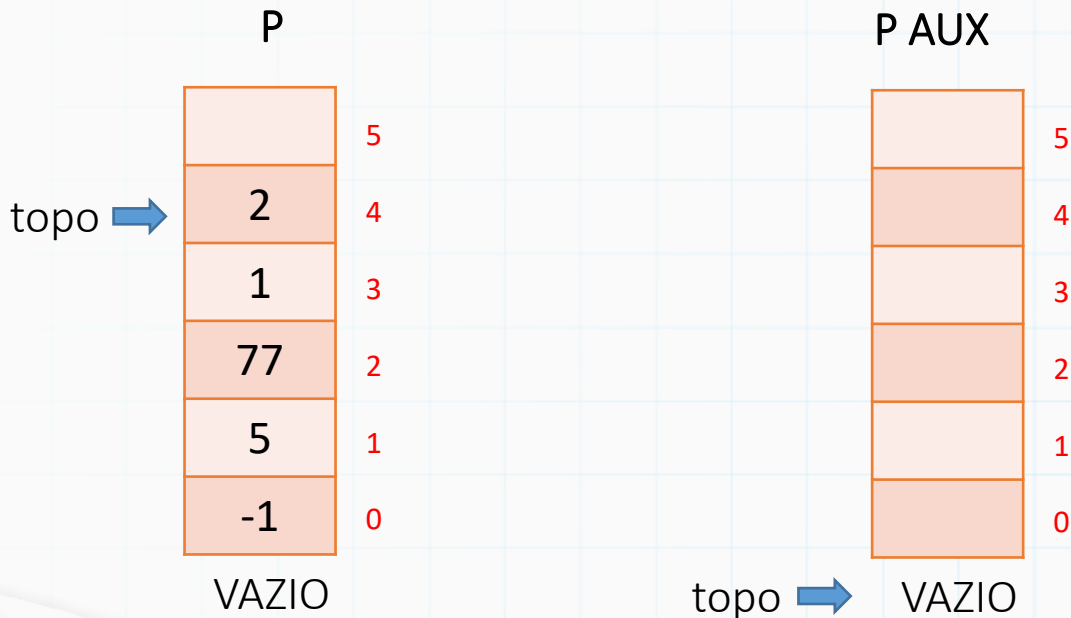
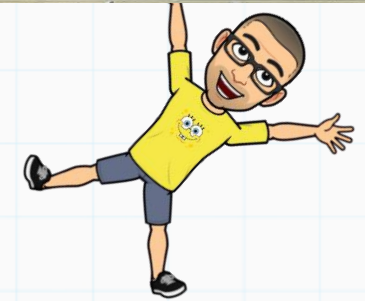
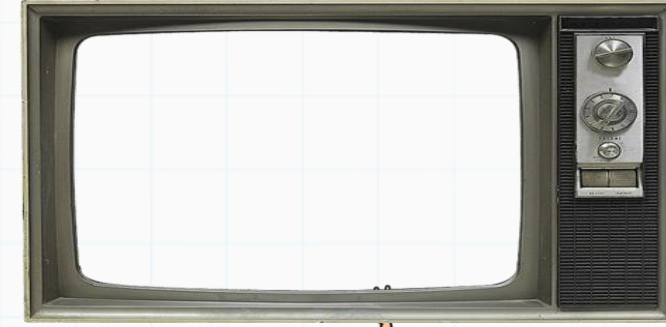
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



1) Vamos retirar os elementos de P e inserir em P AUX

Use só o que
aprendemos até
hoje

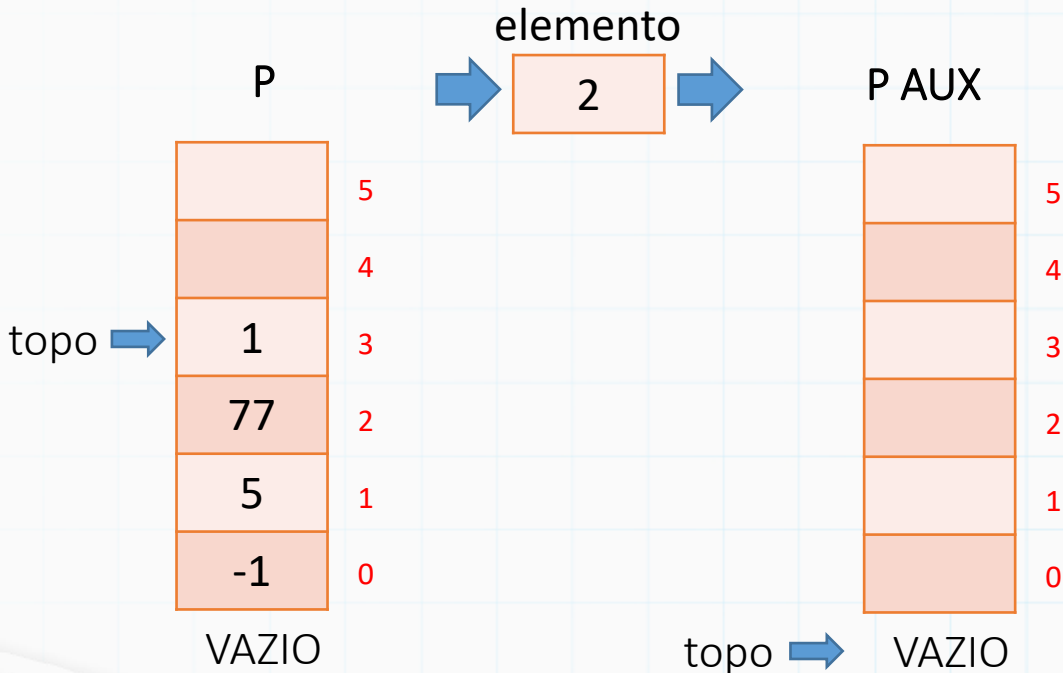
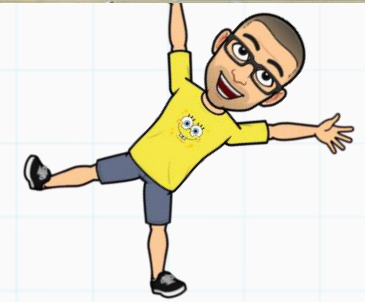
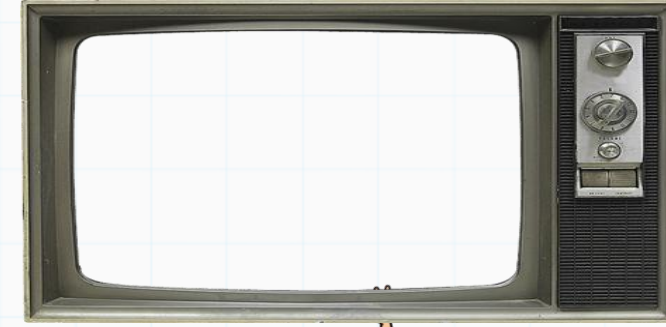
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



1) Vamos retirar os elementos de P e inserir em P AUX

Use só o que
aprendemos até
hoje

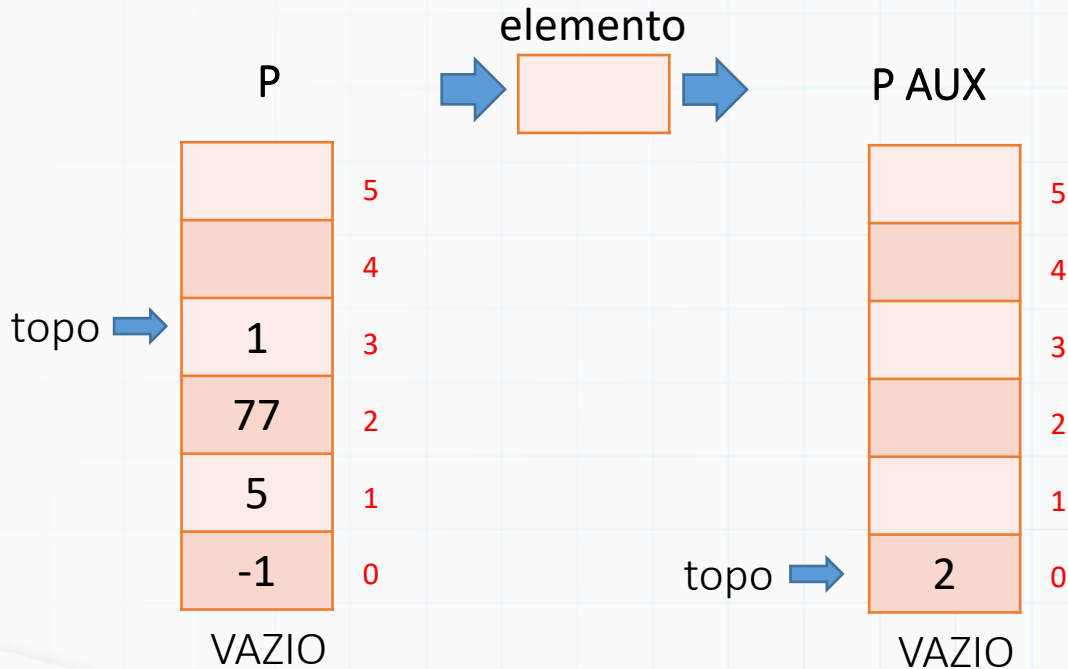
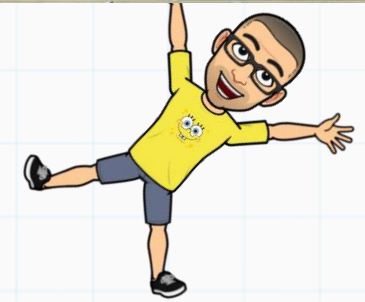
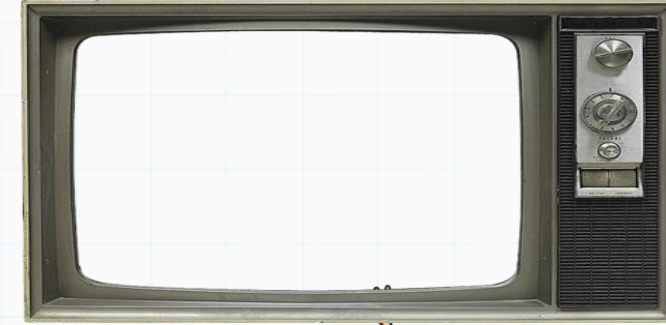
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX

Use só o que
aprendemos até
hoje

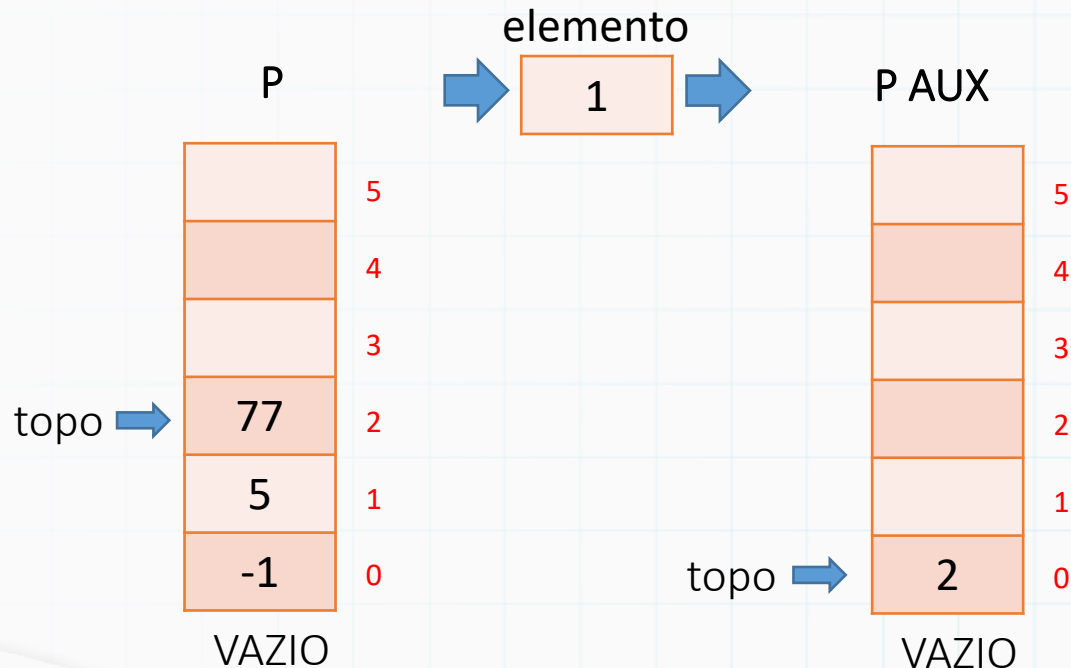
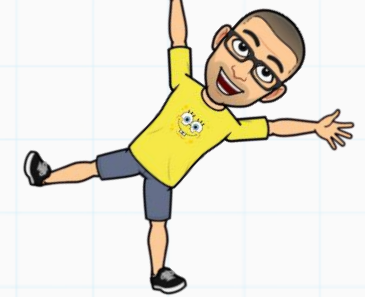
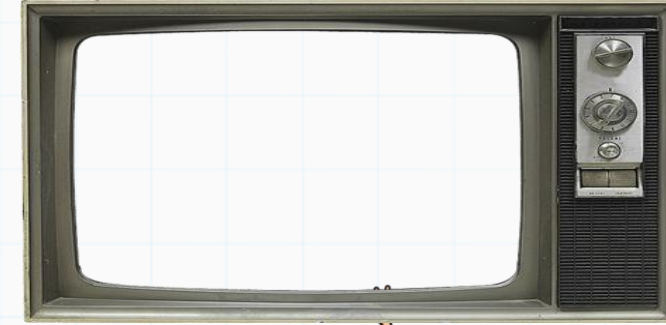
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

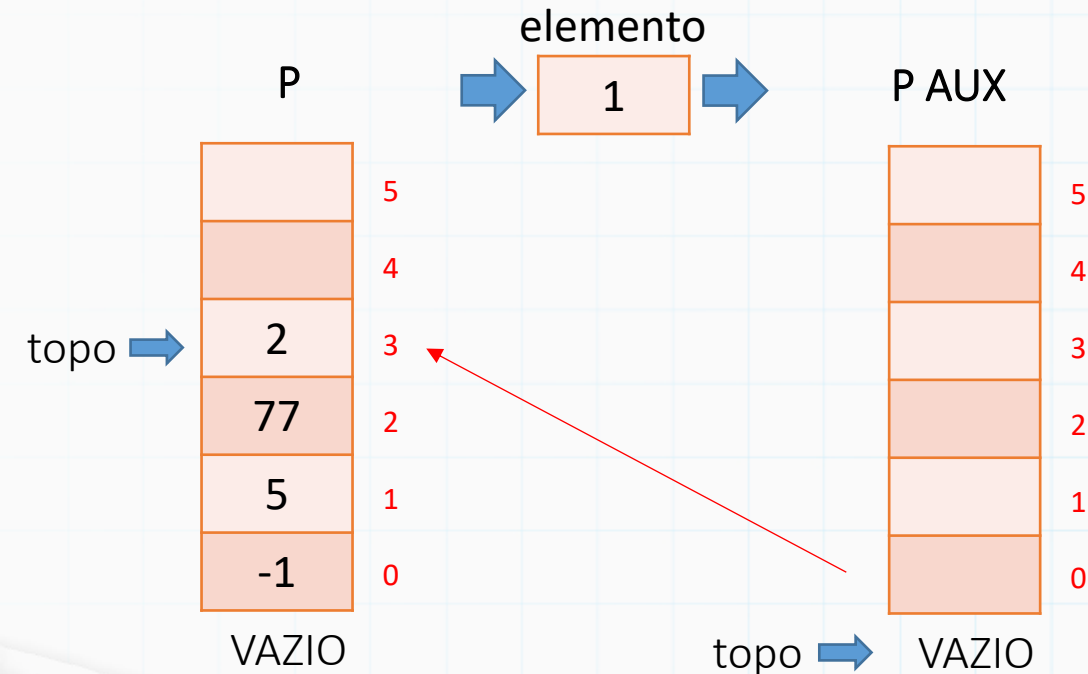
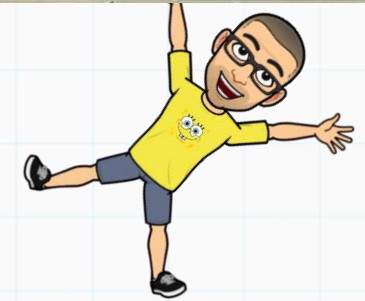
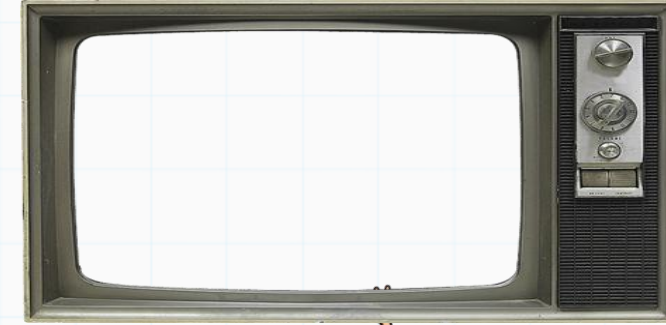
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

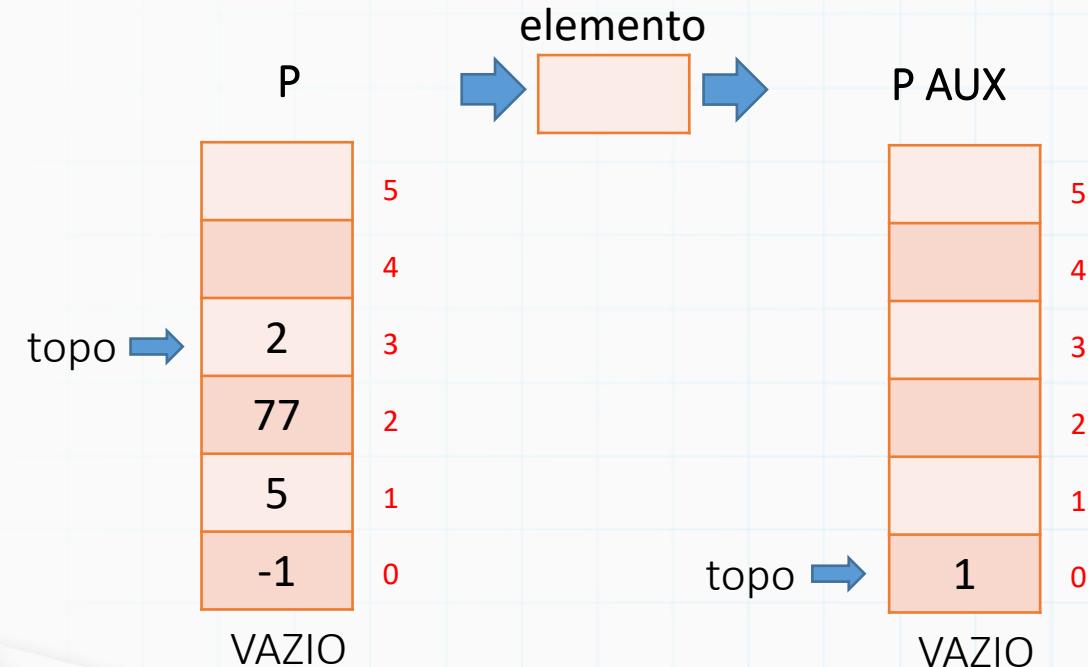
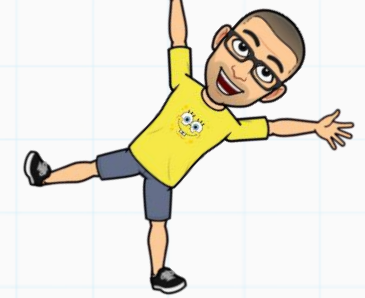
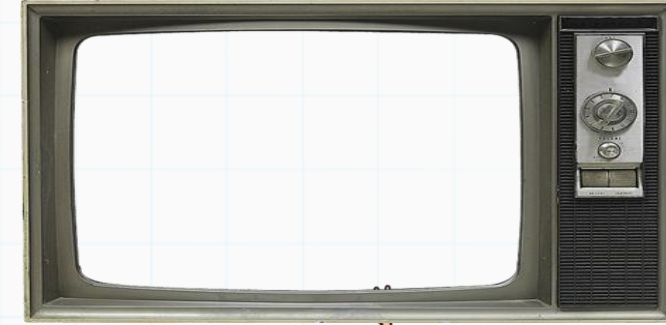
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

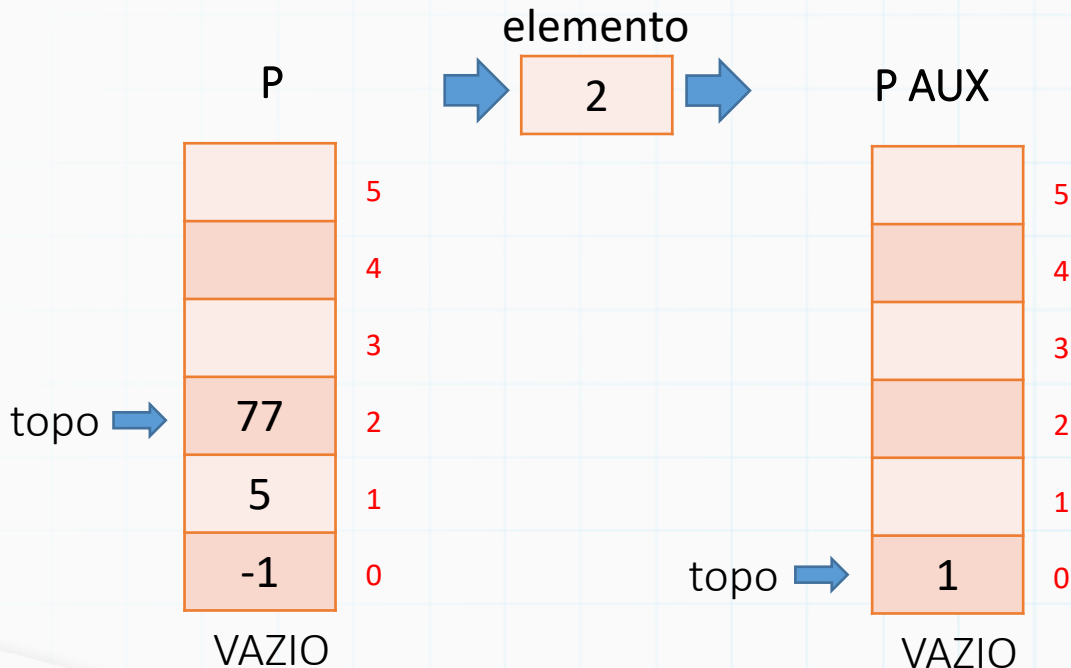
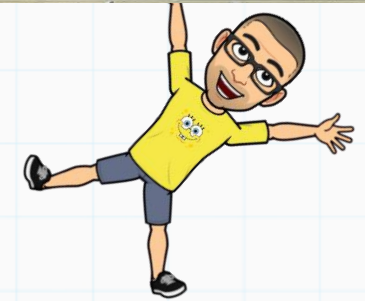
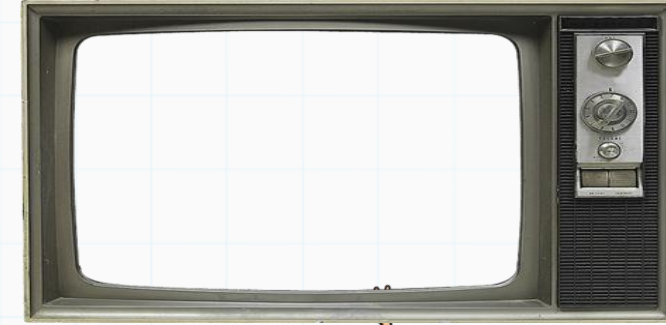
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

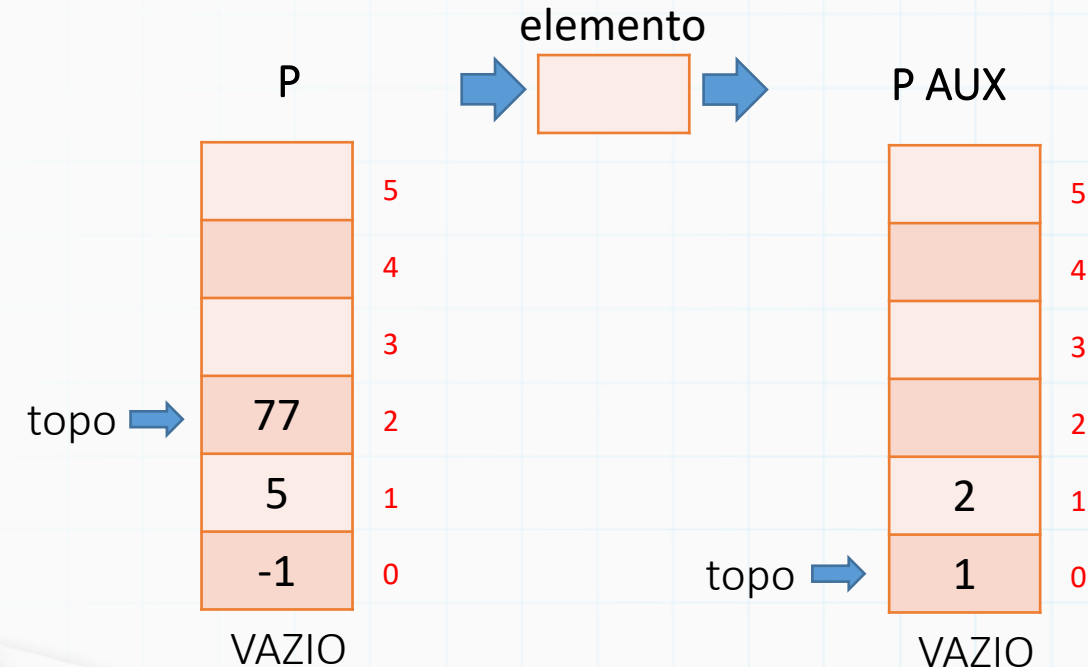
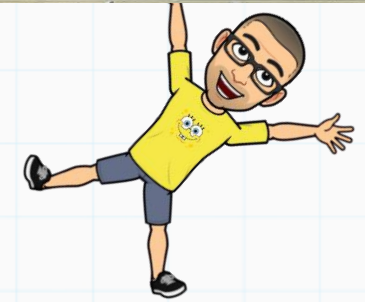
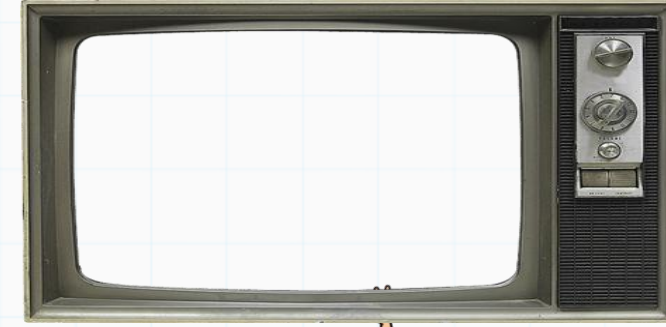
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

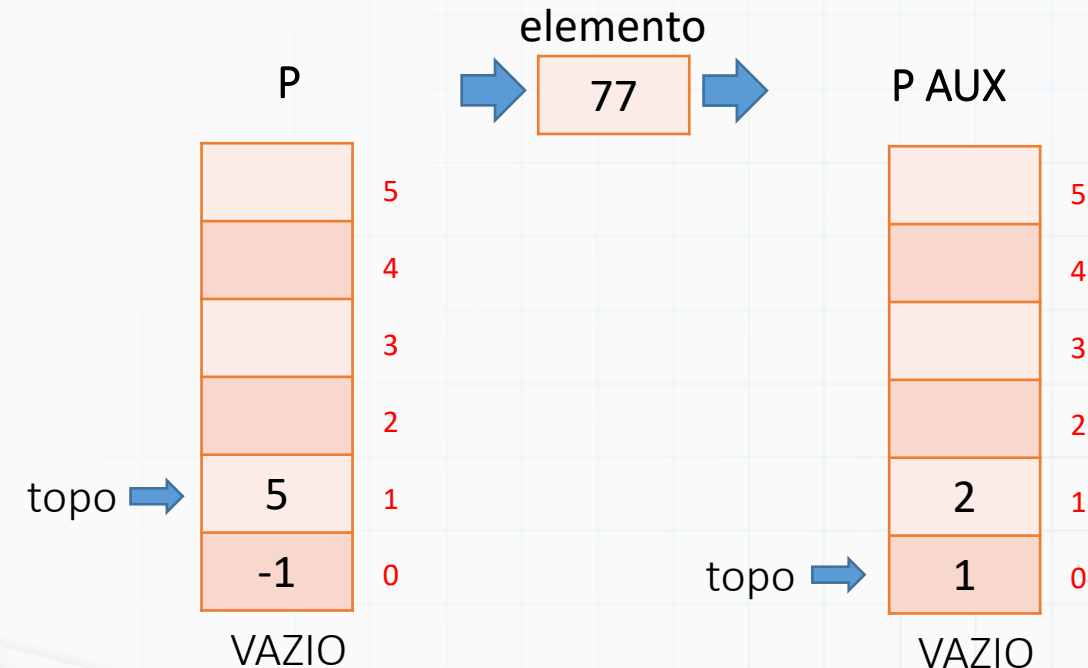
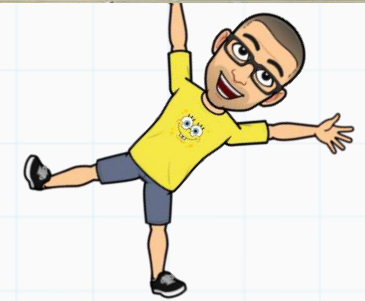
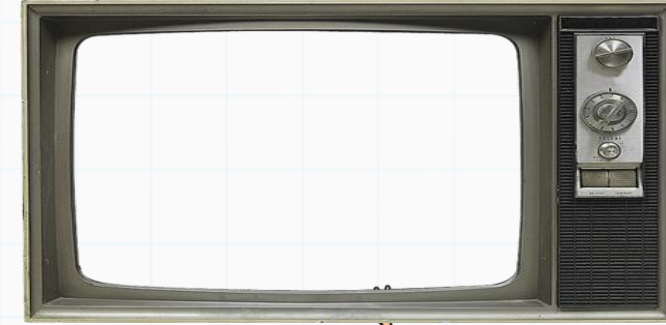
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

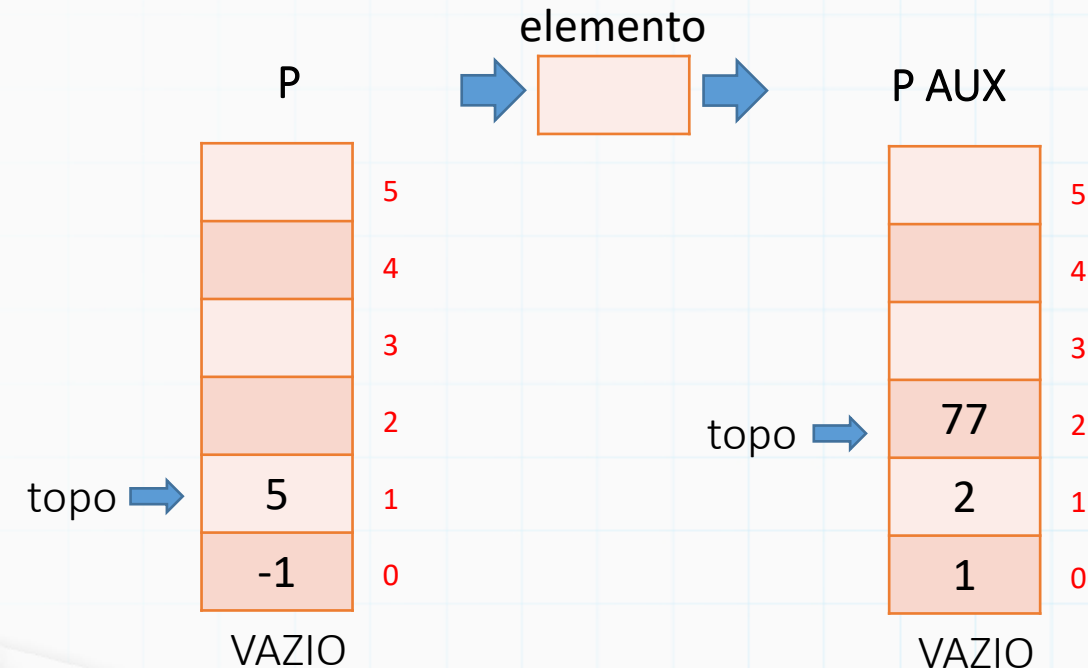
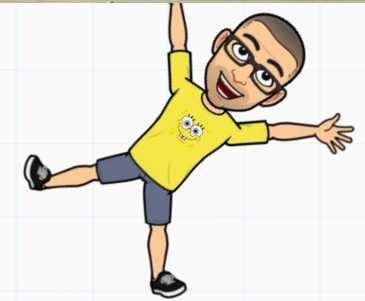
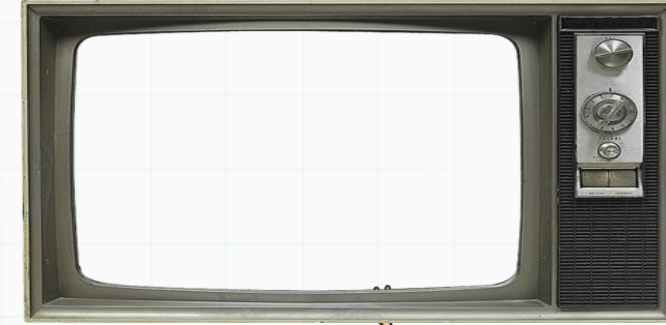
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

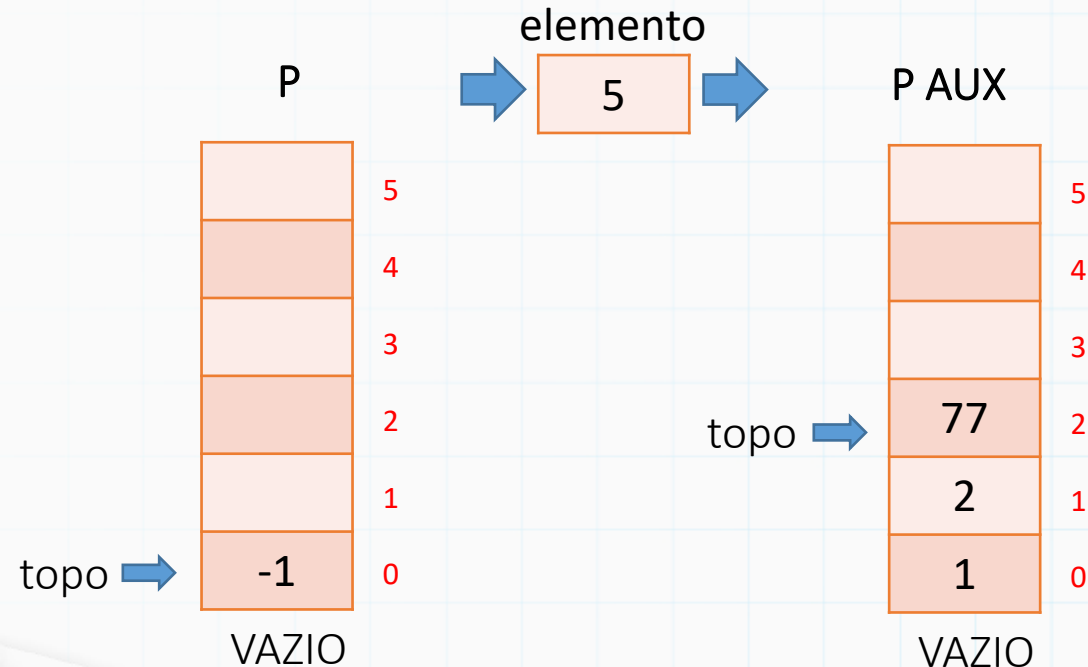
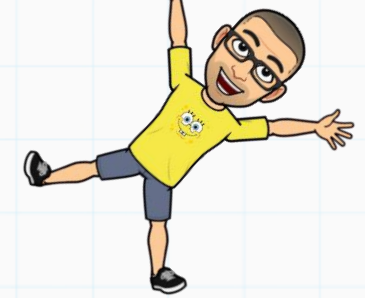
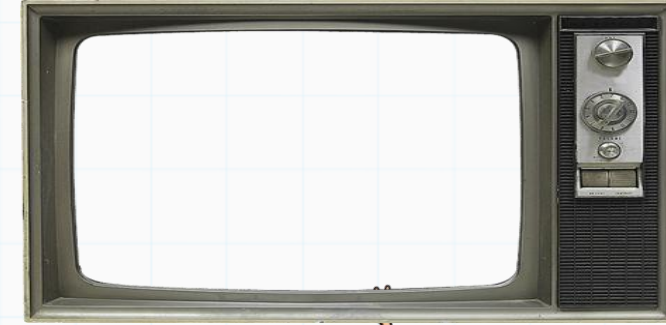
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

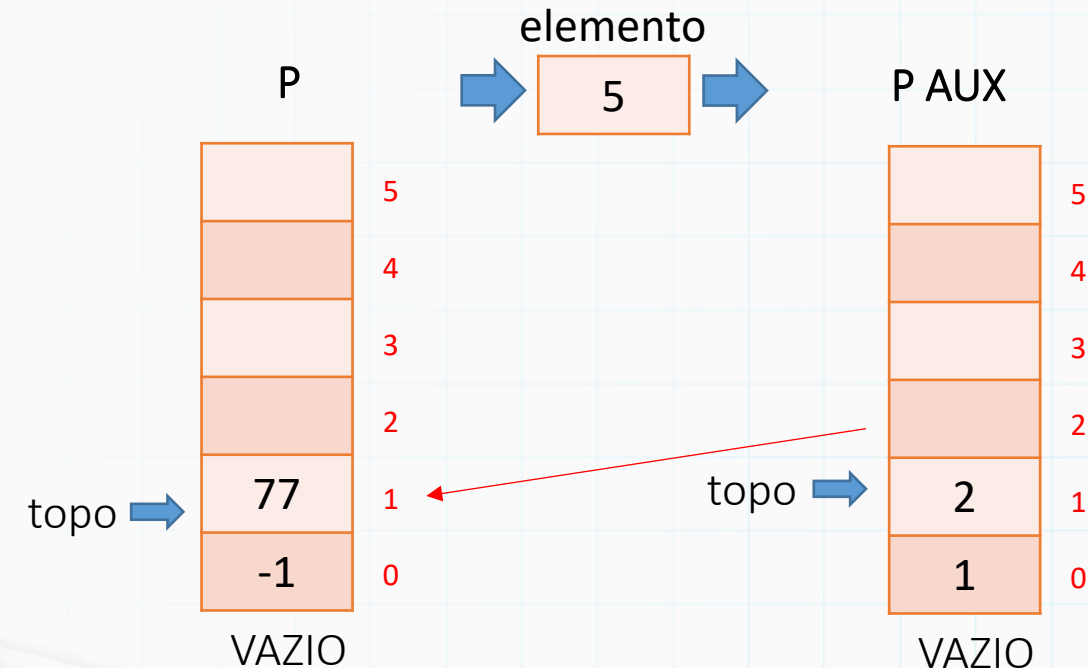
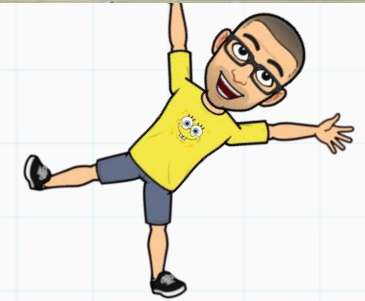
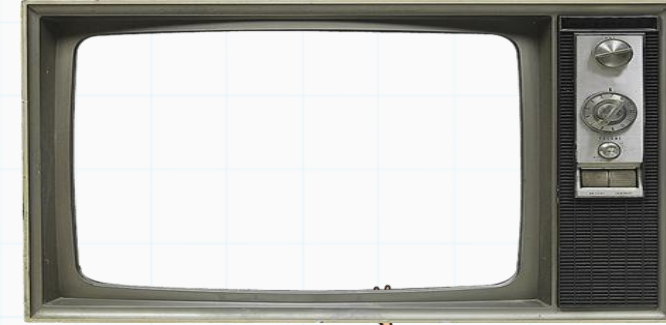
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

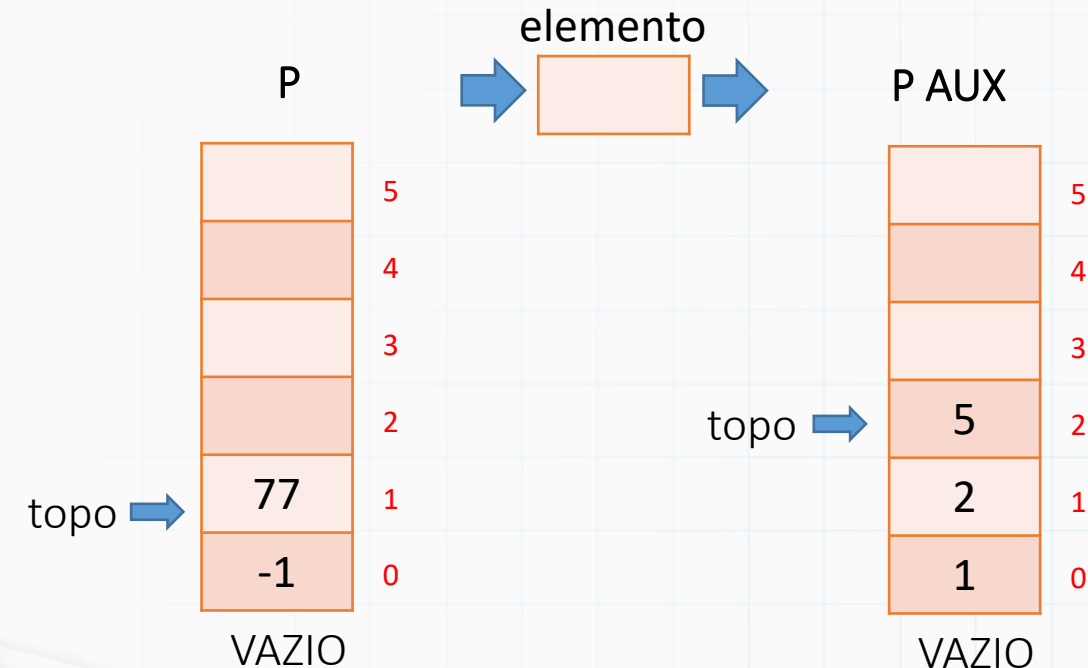
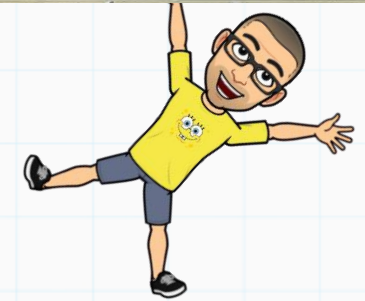
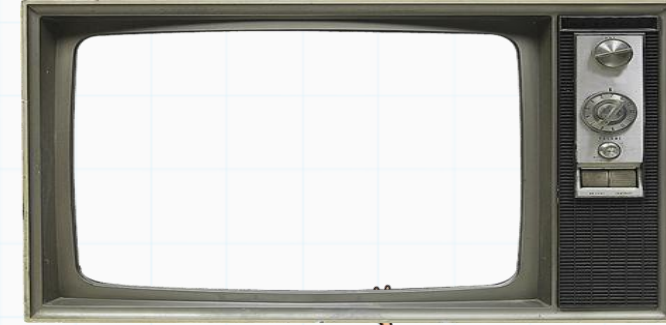
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

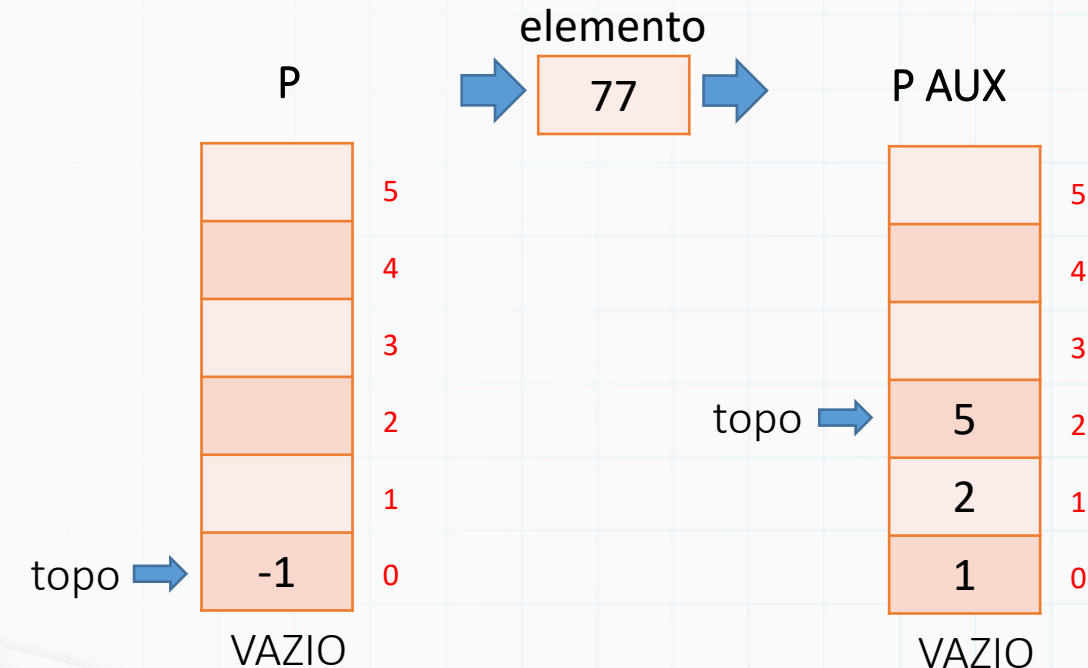
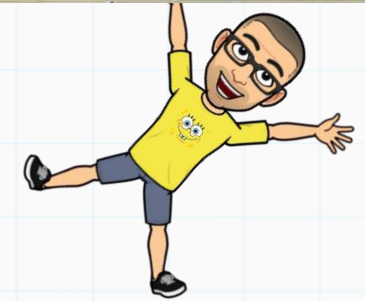
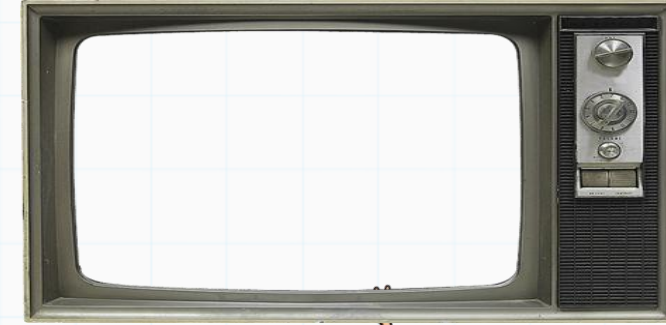
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

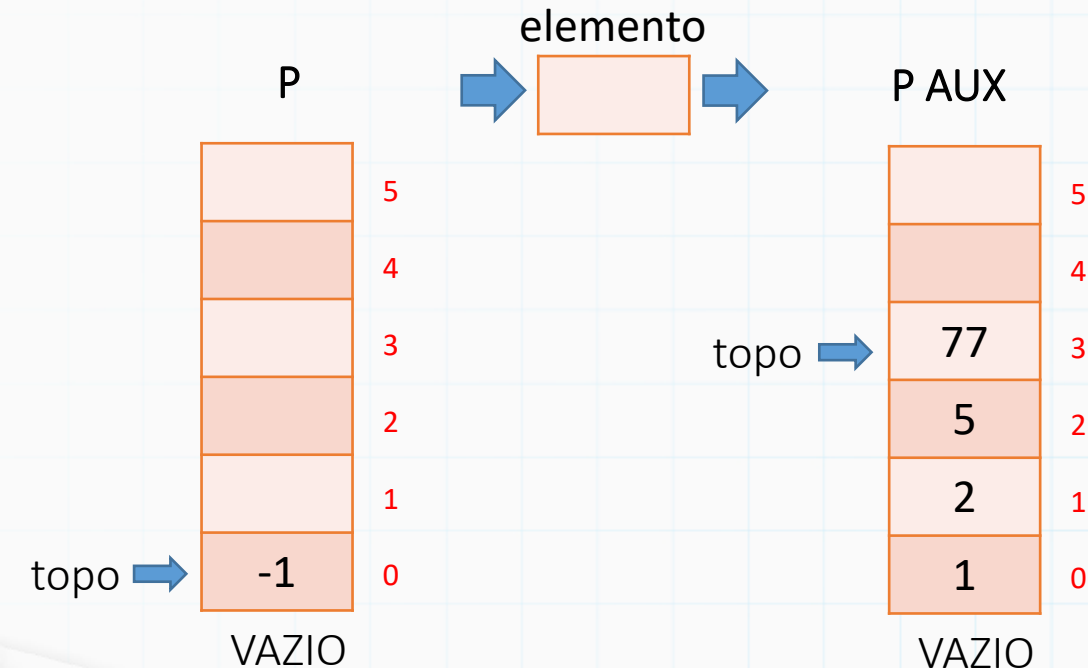
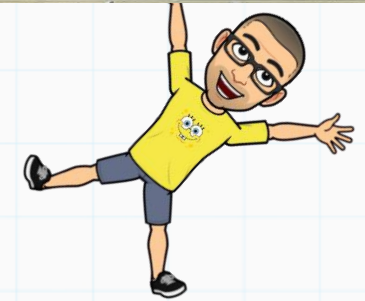
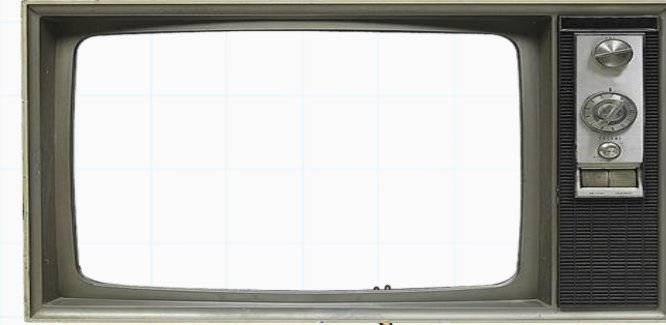
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

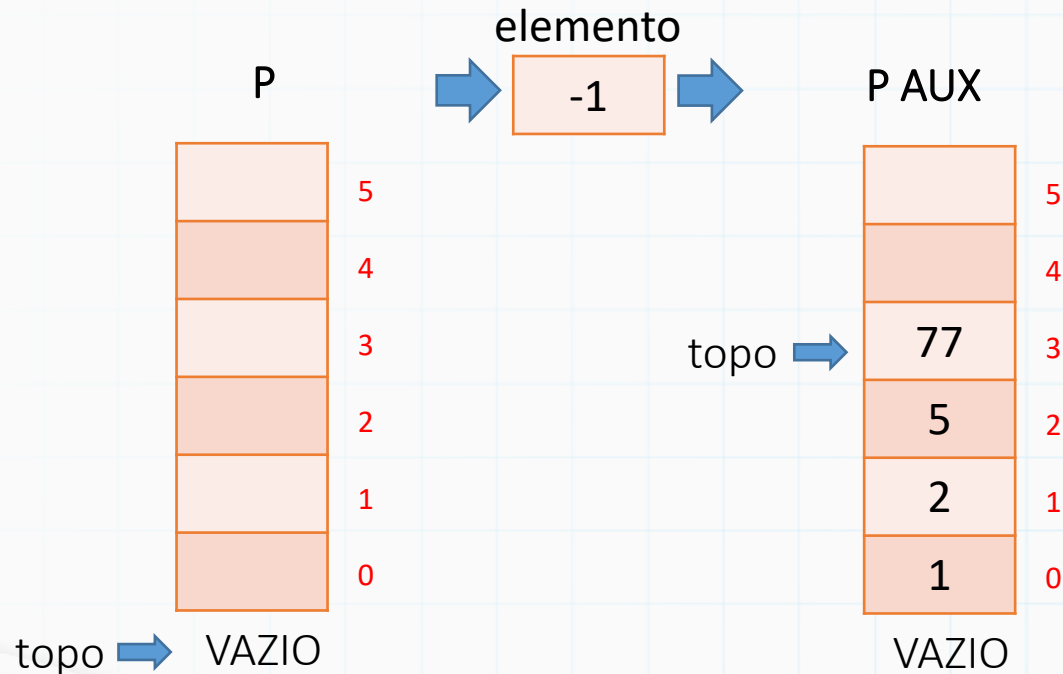
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

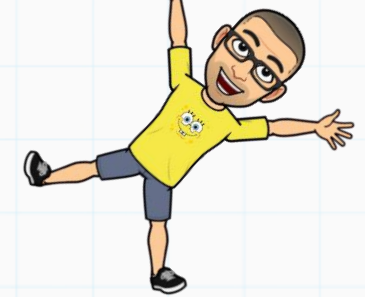
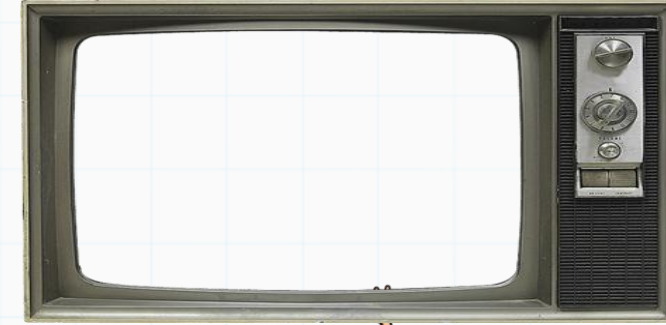
**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje



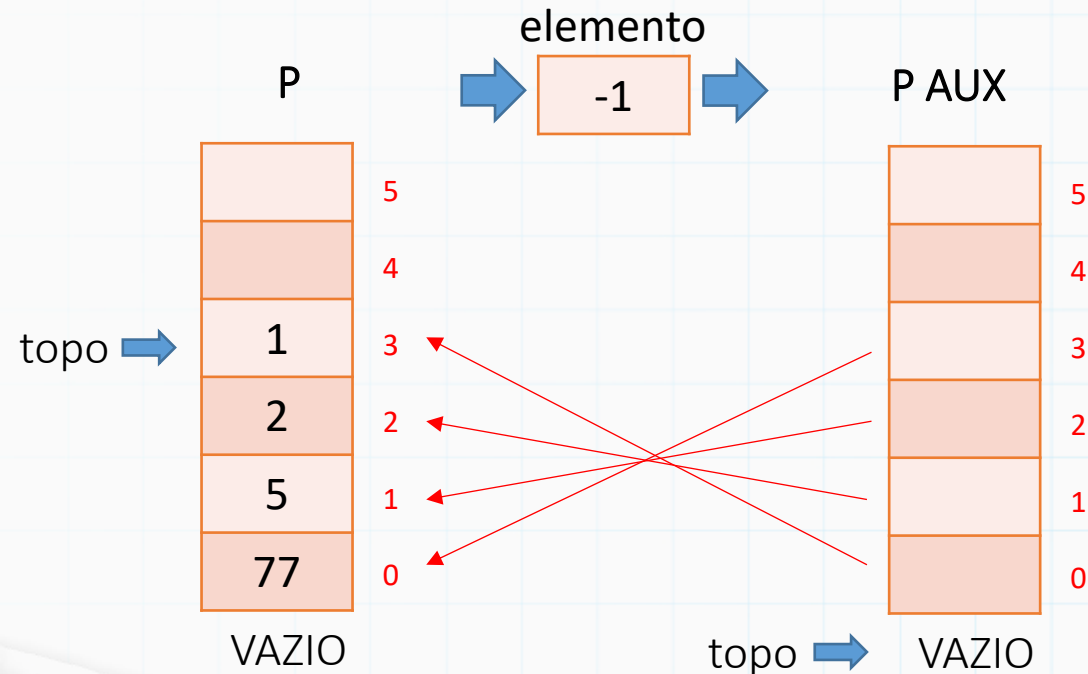
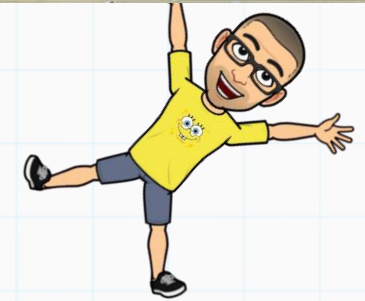
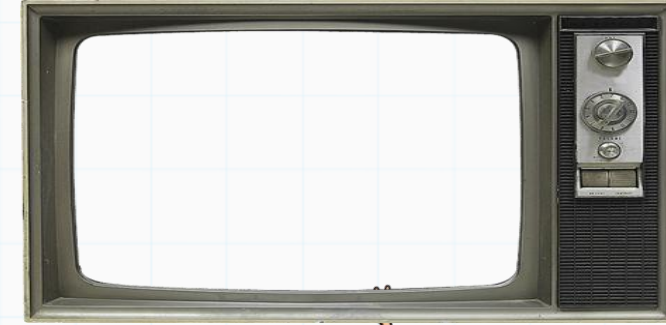
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

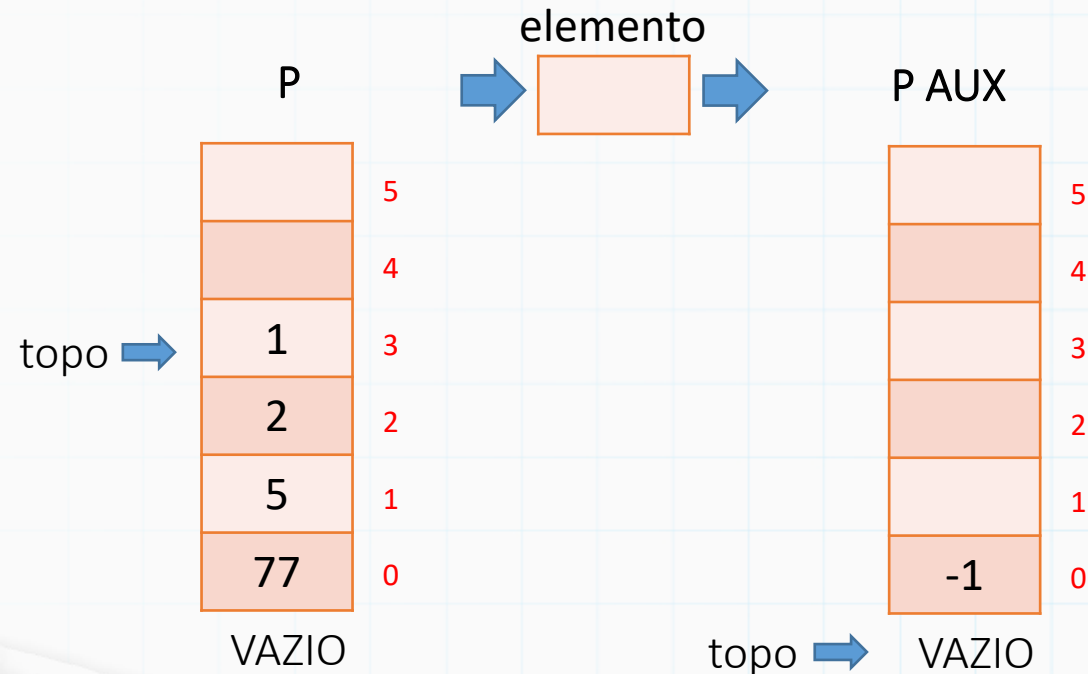
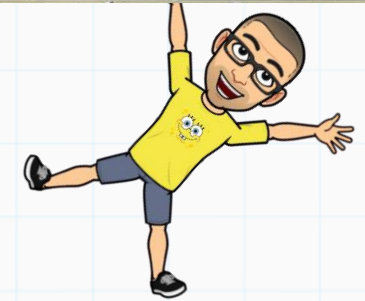
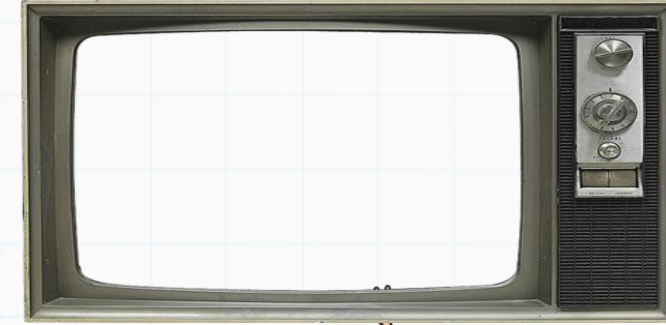
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

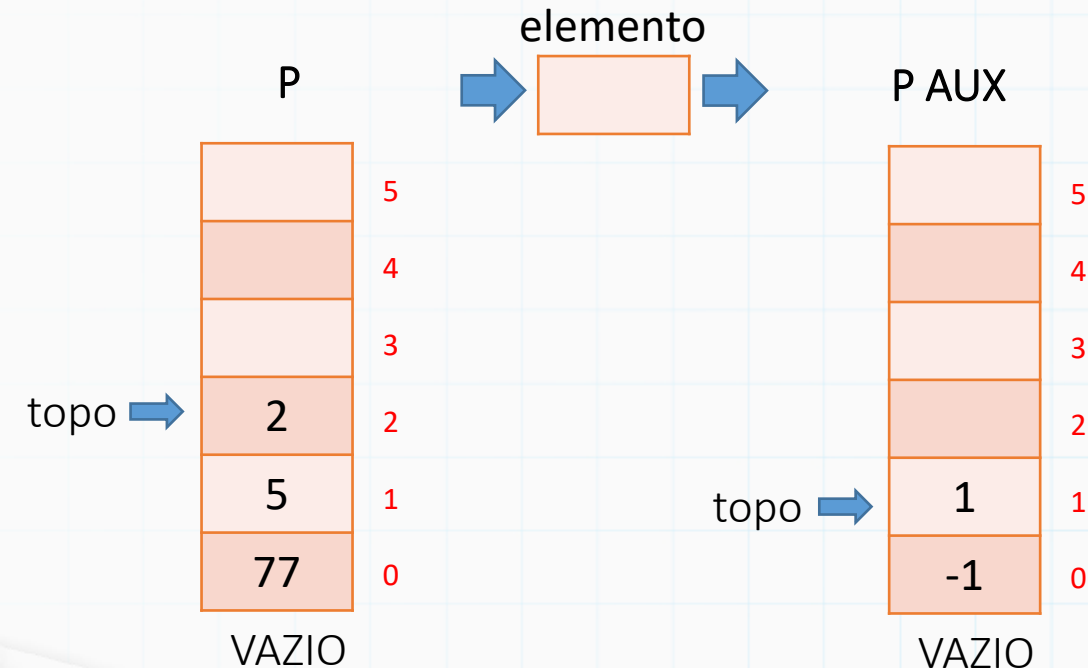
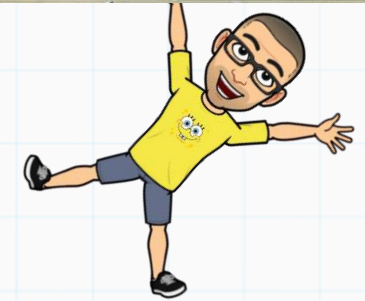
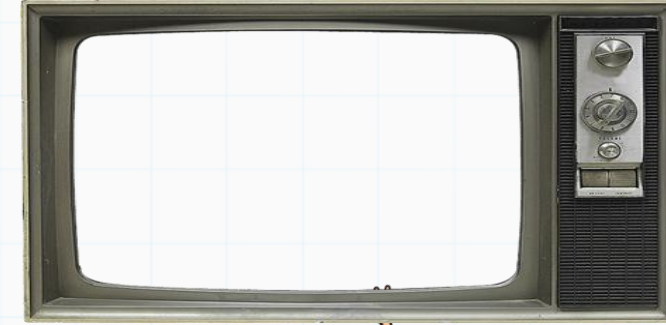
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

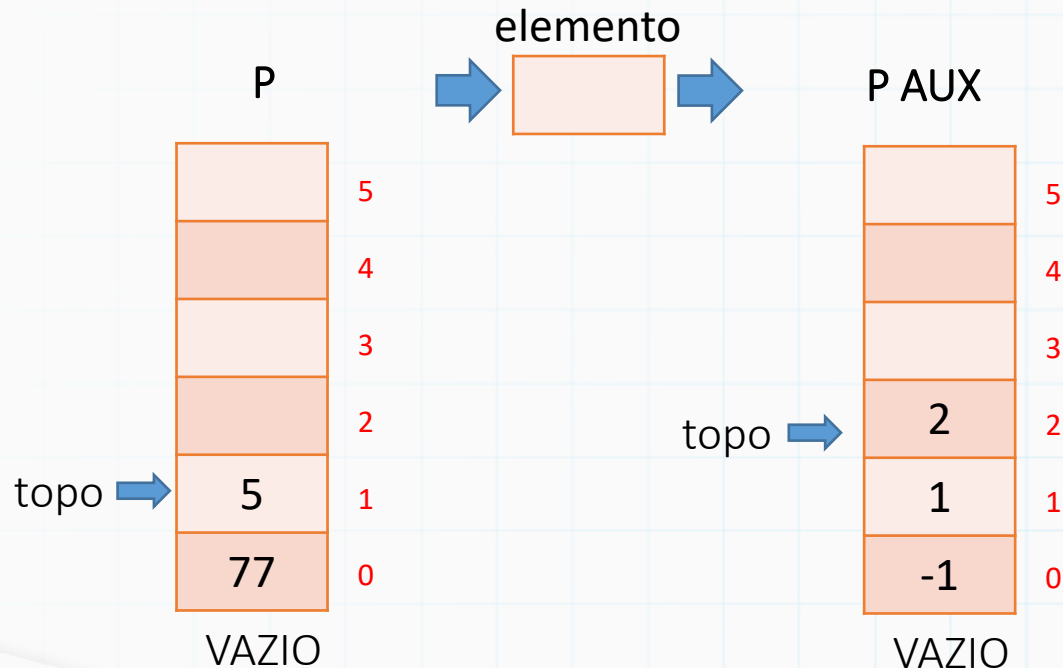
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

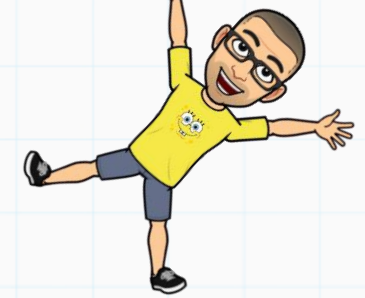
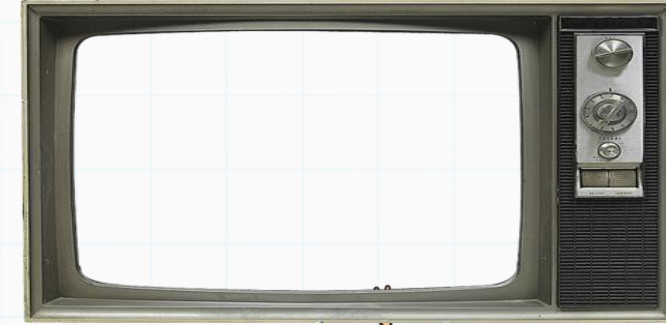
**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje



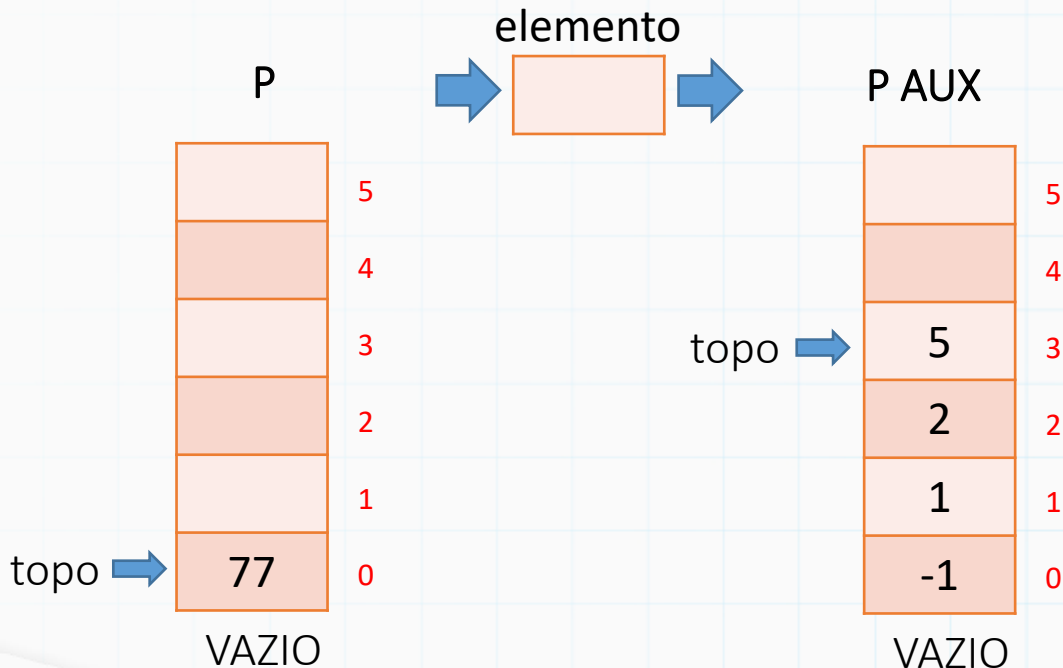
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

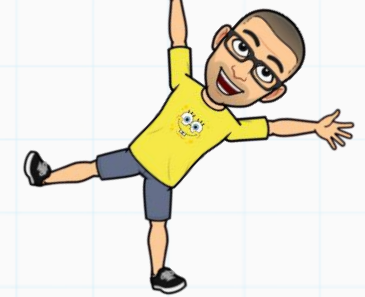
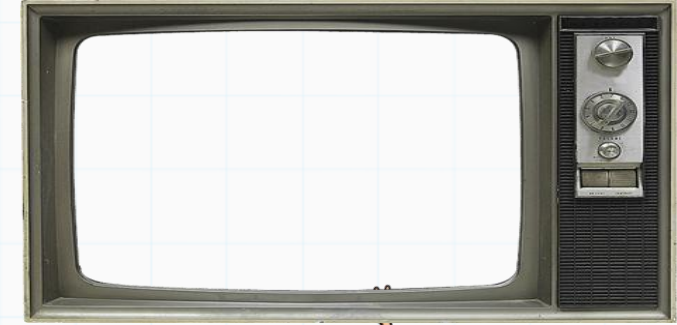
**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje



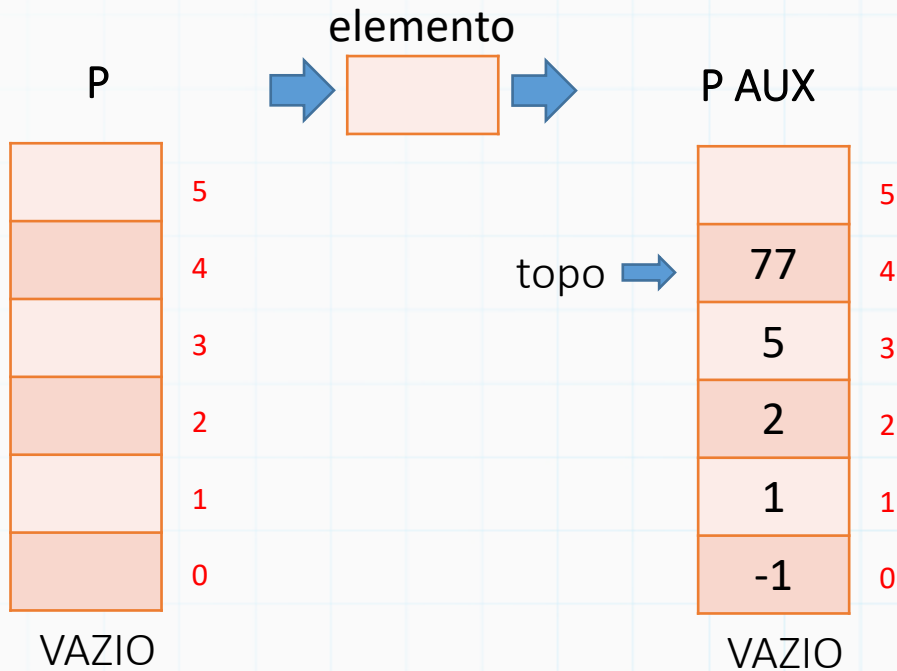
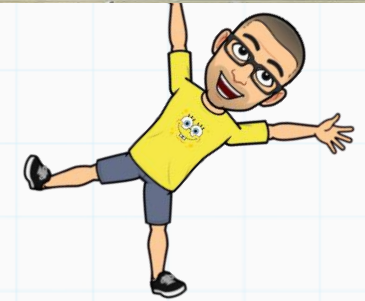
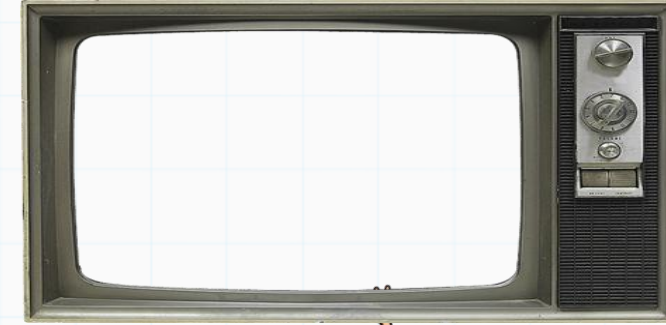
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento

Use só o que
aprendemos até
hoje

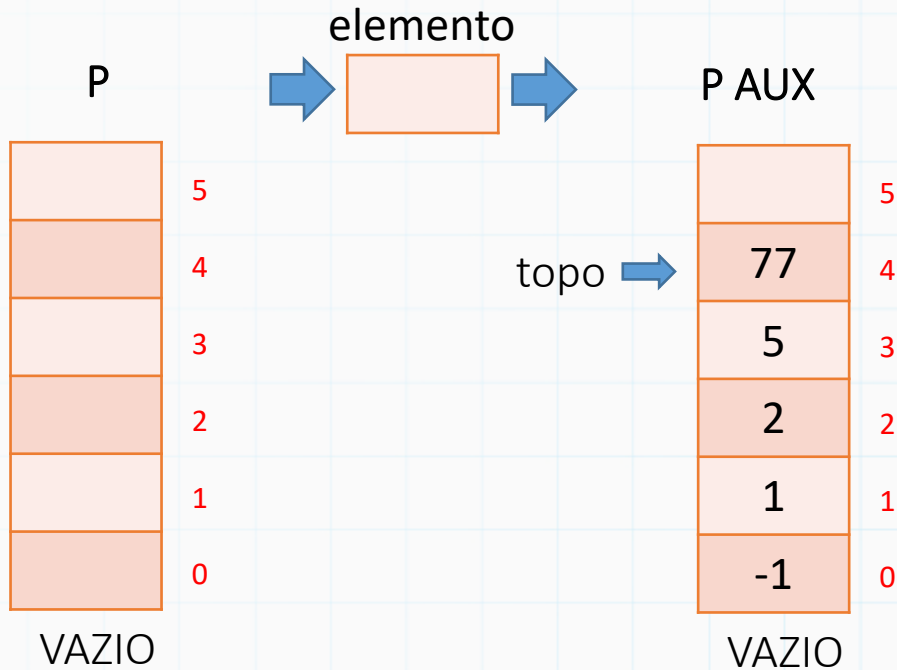
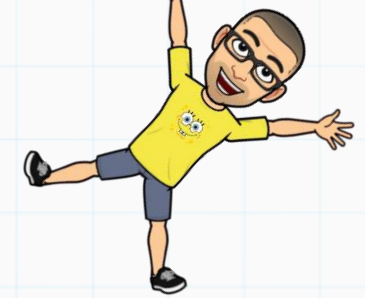
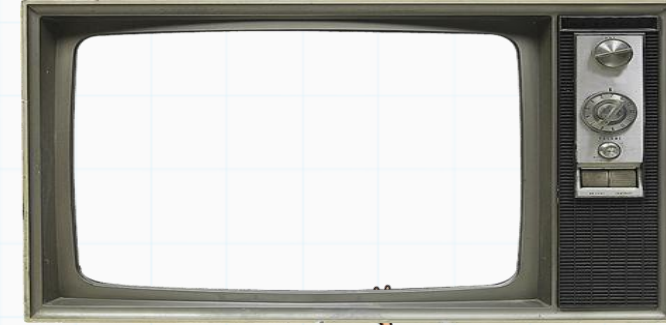
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento
- 4) Depois que P vazio, joga de P AUX de volta para P

Use só o que
aprendemos até
hoje

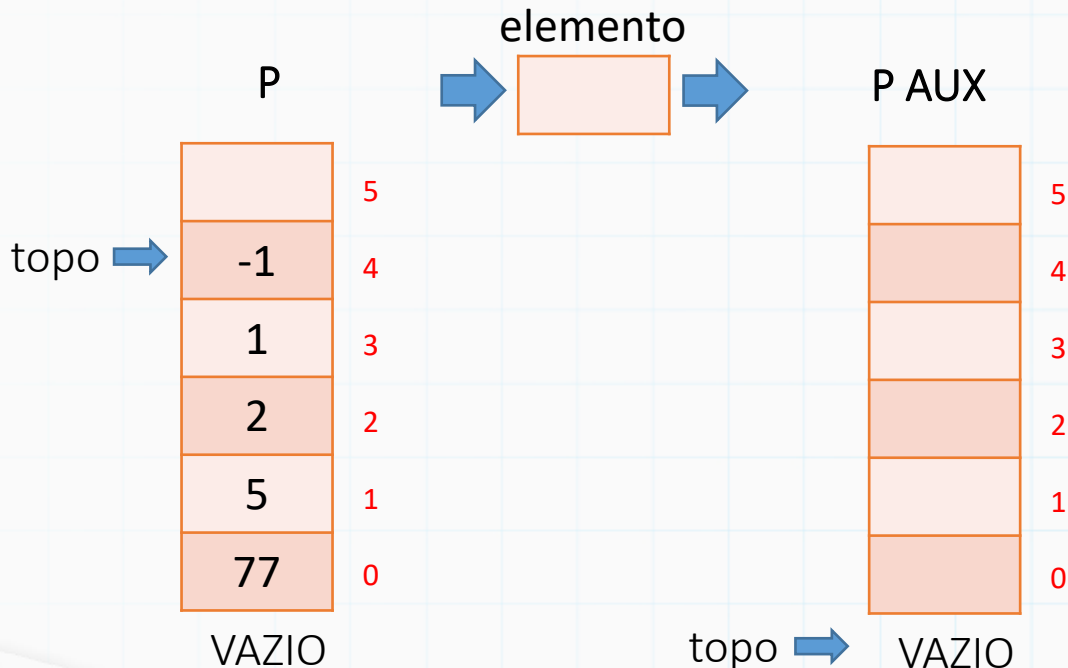
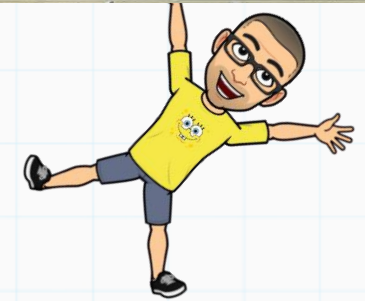
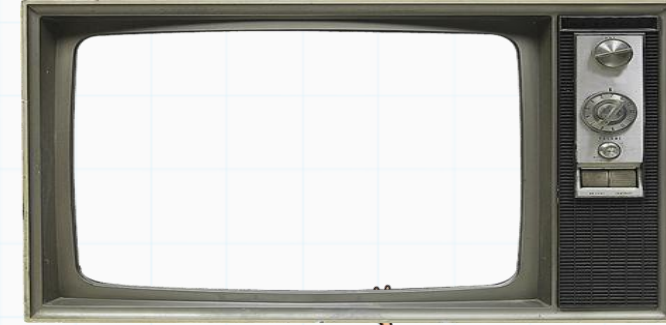
TAD - Pilha

3) Stack-Sort: Dado uma pilha, queremos ordena-la (crescente). Para isso, escreva uma função `ordena_pilha` para ordenar seus elementos.

```
void ordena_pilha(pilha* P)
```

**NÃO USE OUTRAS
ESTRUTURAS (VETORES,
MATRIZES, LISTAS, ETC)**

- A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop`, `top`, etc...)
- Vamos usar uma pilha auxiliar dentro da função. Veja exemplo



- 1) Vamos retirar os elementos de P e inserir em P AUX
- 2) Mas só insere em P AUX se o elemento for maior que o topo de P AUX
- 3) Se não for, joga o topo de P AUX de volta para P, até que o topo de P AUX seja menor (ou igual) que o elemento
- 4) Depois que P vazio, joga de P AUX de volta para P

Veja o exemplo de execução abaixo:



Use só o que aprendemos até hoje

TAD - Pilha

3) Stack-Sort

Código da main.c

```
int main()
{
    pilha P;
    cria_pilha(&P, 10);

    push_pilha(&P, -1);
    push_pilha(&P, 5);
    push_pilha(&P, 77);
    push_pilha(&P, 1);
    push_pilha(&P, 2);
    imprime_pilha(&P);

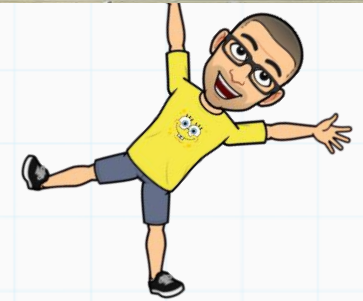
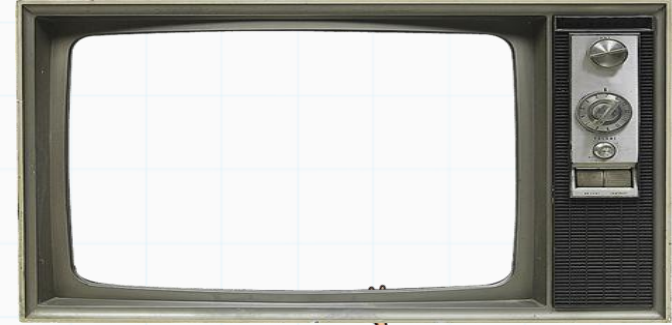
    ordena_pilha(&P);
    imprime_pilha(&P);

    return 0;
}
```

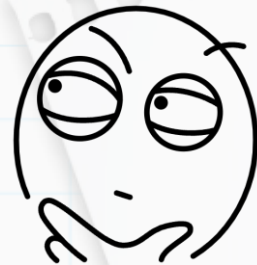
Exemplo de execução:

```
| 2 | <- topo
| 1 |
| 77 |
| 5 |
| -1 |

| -1 | <- topo
| 1 |
| 2 |
| 5 |
| 77 |
```



para uma pilha dada por (0, 5, 30, 40, 9, 22, 100), onde 0 é a base e 100 é o topo da pilha, ao aplicarmos o Stack-Sort como descrito, quantas operações de push na pilha auxiliar seriam realizadas ao ordenar a pilha ?



TAD - Pilha

4) Push, push, pop: Faça uma função que leia uma sequência de números inteiros arbitrariamente longa x_1, x_2, \dots colocando cada um deles no topo de uma pilha inicialmente vazia.

Sempre que o valor no topo da pilha for um número negativo, você deve tentar tirar dois elementos da pilha, somá-los e colocar o resultado no topo.

Pare quando esse procedimento não for mais possível (isto é, a pilha tem menos que dois elementos para serem retirados) ou quando o topo da pilha contiver o valor zero, mesmo que ainda haja números na entrada.

Como saída, imprima cada operação de inserção e remoção que for realizada. Ao empilhar um elemento, imprima uma string no formato "**push %d**". Ao desempilhar, imprima uma string no formato "**pop %d**".

Exemplo com sequencia = 1, 2, -3, 5

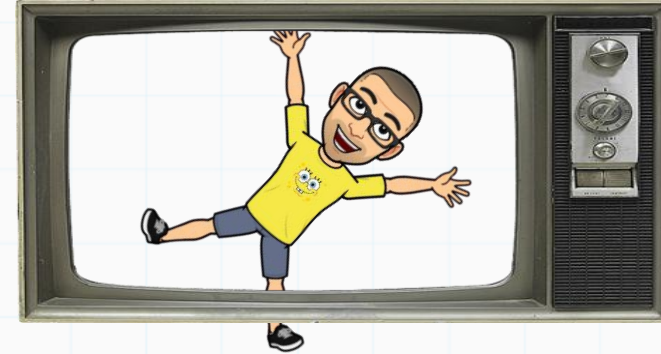
```
1 // entrada
push 1 // saída
2 // entrada
push 2 // saída
-3 // entrada
push -3 // saída
pop -3 // saída (elemento retirado negativo, retira outro, soma, e insere o resultado)
pop 2 // saída
push -1 // saída
pop -1 // elemento retirado negativo, retira outro, soma, e insere o resultado
pop 1 // saída
push 0 // Parou pois o elemento do topo é 0 (entrada 5 nunca é recebida)
```

Exemplo de
execução

Use só o que
aprendemos até
hoje



TAD - Pilha



4) Push, push, pop:

Código da main.c

```
int main()
{

    pilha P;
    cria_pilha(&P, 10);

    push_push_pop(&P);

    libera_pilha(&P);

    return 0;

}
```

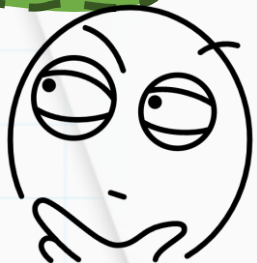
Exemplo de execução sequencia = 1, 2, -3, 5 :

```
e1 = 1
push 1
e1 = 2
push 2
e1 = -3
push -3
pop -3
pop 2
push -1
pop -1
pop 1
push 0
FIM
```

Exemplo de execução sequencia = 5, 5, 2, -1, -1 :

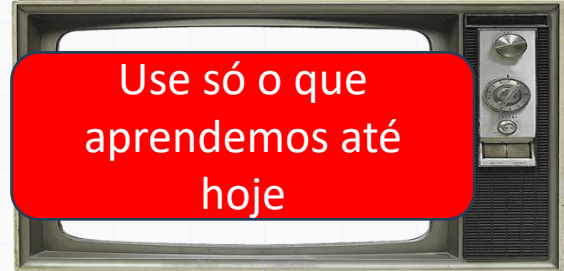
```
e1 = 5          push -1
push 5         pop -1
e1 = 5         pop 1
push 5         push 0
e1 = 2         FIM
push 2
e1 = -1
push -1
pop -1
pop 2
push 1
e1 = -1
```

para uma sequencia dada por (1, 2, 3, 4, 5, -6, 3, 3, 3, 3, -30), qual seria a soma dos os elementos retirados (pops) durante o processo ?



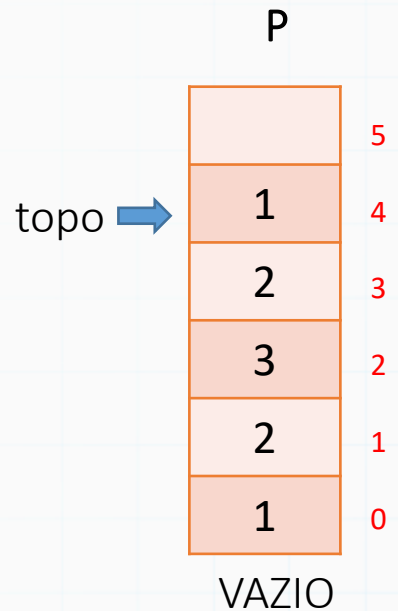
TAD - Pilha

4.5) Palíndromo: Faça uma função que dada uma pilha P, verifique se a sequência de elementos armazenados nela forma um palíndromo. Exemplos:



Use só o que aprendemos até hoje

PALINDROMO



PALINDROMO

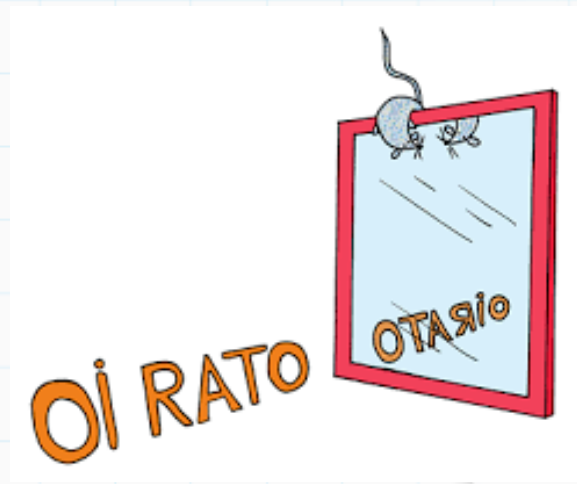
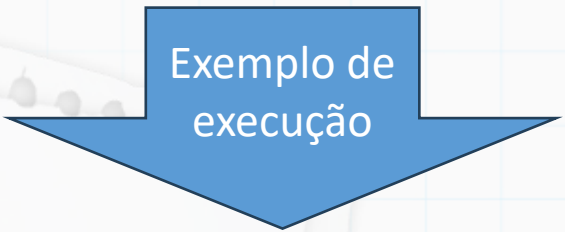


NÃO É PALINDROMO



NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC) MAS PODEMOS CRIAR E USAR PILHAS AUXILIARES DE DENTRO DA FUNÇÃO.

Dica: 2 pilhas auxiliares, uma com a ordem original dos elementos, e outra com ordem invertida.



TAD - Pilha

4.5) Palíndromo:

Código da main.c

```
int main()
{
    pilha P;
    cria_pilha(&P, 10);

    push_pilha(&P, 1); push_pilha(&P, 2);
    push_pilha(&P, 3); push_pilha(&P, 2);
    push_pilha(&P, 1);

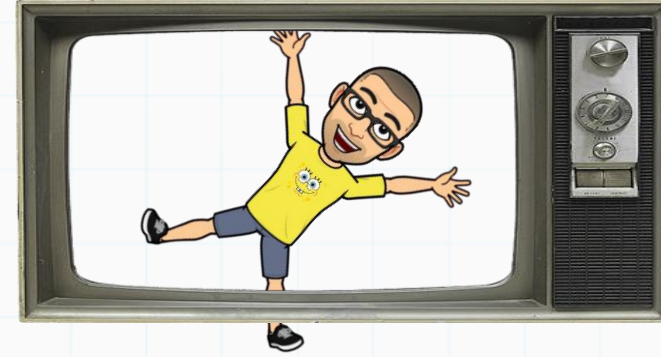
    imprime_pilha(&P);
    printf("resposta = %d\n", palindromo_pilha(&P));

    return 0;
}
```

Exemplo de execução:

```
| 1 | <- topo
| 2 |
| 3 |
| 2 |
| 1 |

resposta = 1
```



TAD - Pilha

5) Parênteses: Escreva uma função que receba uma `string` que representa uma expressão matemática, e diga se os parênteses da expressão estão balanceados. **Vamos usar uma pilha auxiliar dentro da função.**



Um símbolo de abertura (“{”, “[” e “(”) é considerado balanceado se existe um símbolo de fechamento (“}”, “]” e “)”) correspondente posterior na expressão.

Exemplo:

{a + (f * b) }	Equilibrado
(b - (c * d) / p] + k)	Desequilibrado
(m + n) * }v / p { + u	Desequilibrado

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC) MAS PODEMOS CRIAR E USAR UMA PILHA AUXILIAR DE DENTRO DA FUNÇÃO.

Use só o que aprendemos até hoje



Ideia:

- Percorrer a expressão (string) e para cada caractere:
 - Se o caractere for de abertura então guarda na pilha (a pilha guardará os caracteres de abertura),
 - Porém se o caractere for de fechamento, retira o topo da pilha e checa qual foi a última abertura encontrada:
 - Checa se a pilha está vazia (não tem uma abertura correspondente para o fechamento) OU se o caractere do topo da pilha (abertura) e o caractere da string (fechamento) não são do mesmo tipo. Se uma das duas coisas acontecer temos um desbalanceamento na expressão.
- Veja exemplo de execução abaixo

No C um caractere (char) pode ser armazenado em um inteiro (int), e vice e versa

TAD - Pilha



5) Parênteses Código da main.c

```
int main()
{
    char * exp = "{a*[(a+b)*c]+g-[(q+e)/(f*b)]}";
    printf("\nexpressao = %s\n",exp);
    if (balanceado(exp, strlen(exp)) == 1)
        printf("expressao balanceada\n");
    else
        printf("expressao desbalanceada\n");

    char * exp2 = "{a+(b-(c*d)/p)+k)-m}";
    printf("\nexpressao = %s\n",exp2);
    if (balanceado(exp2, strlen(exp2)) == 1)
        printf("expressao balanceada\n");
    else
        printf("expressao desbalanceada\n");

    char * exp3 = "{m+n)*[v/p]+k*u(p-m)}";
    printf("\nexpressao = %s\n",exp3);
    if (balanceado(exp3, strlen(exp3)) == 1)
        printf("expressao balanceada\n");
    else
        printf("expressao desbalanceada\n");

    char * exp4 = "({o*p}*r+k)+a*u[r/e]}";
    printf("\nexpressao = %s\n",exp4);
    if (balanceado(exp4, strlen(exp4)) == 1)
        printf("expressao balanceada\n");
    else
        printf("expressao desbalanceada\n");

    return 0;
}
```

Exemplo de execução:

```
expressao = {a*[(a+b)*c]+g-[(q+e)/(f*b)]}
{[(*)][(*) (*)]}
expressao balanceada

expressao = {a+(b-(c*d)/p)+k)-m}
{[(*)] <- erro
expressao desbalanceada

expressao = {m+n)*[v/p]+k*u(p-m)}
{} <- erro
expressao desbalanceada

expressao = (({o*p}*r+k)+a*u[r/e])
({{}}[[]]) <- erro
expressao desbalanceada
```

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

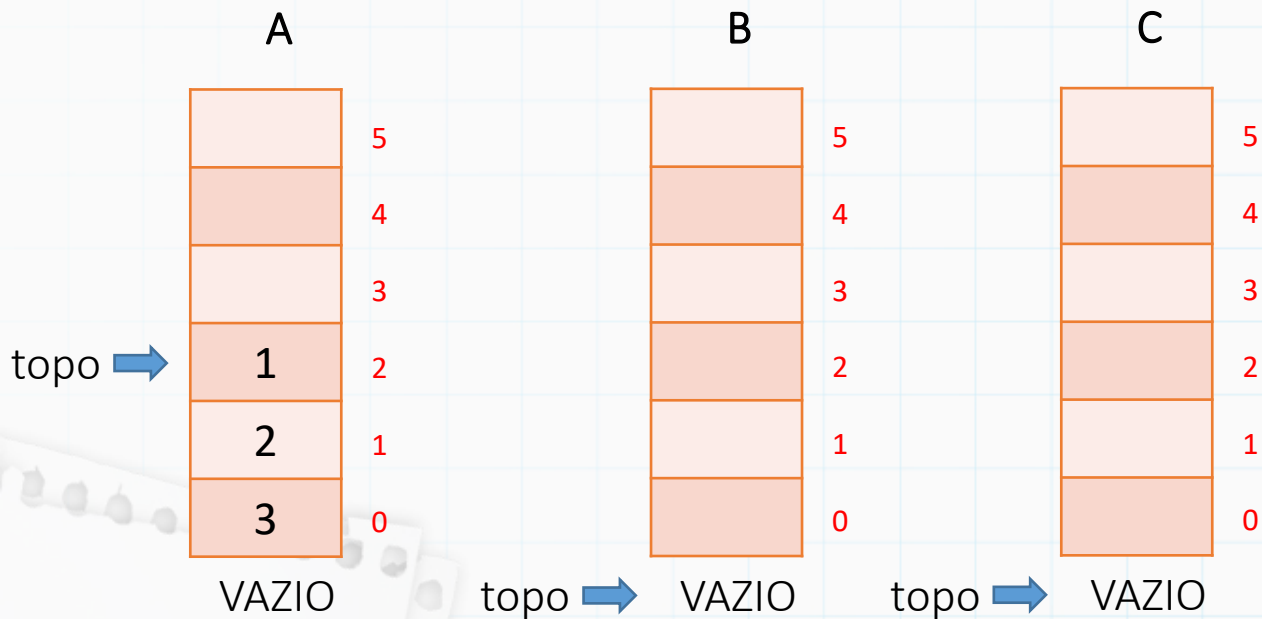
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

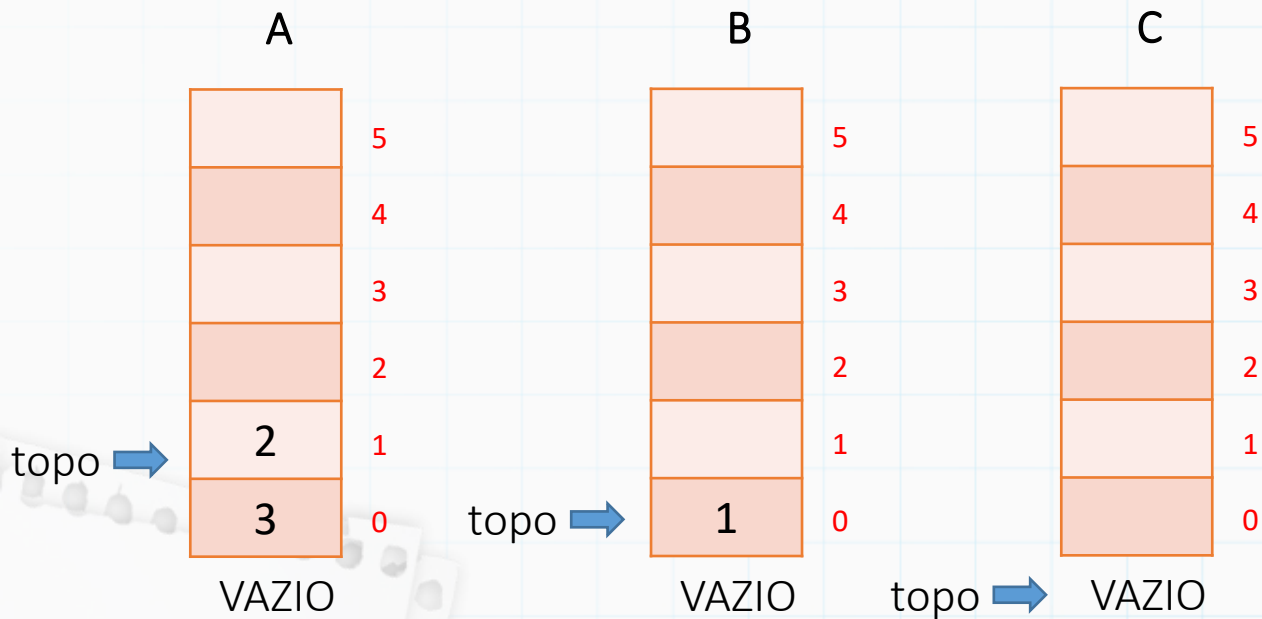
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

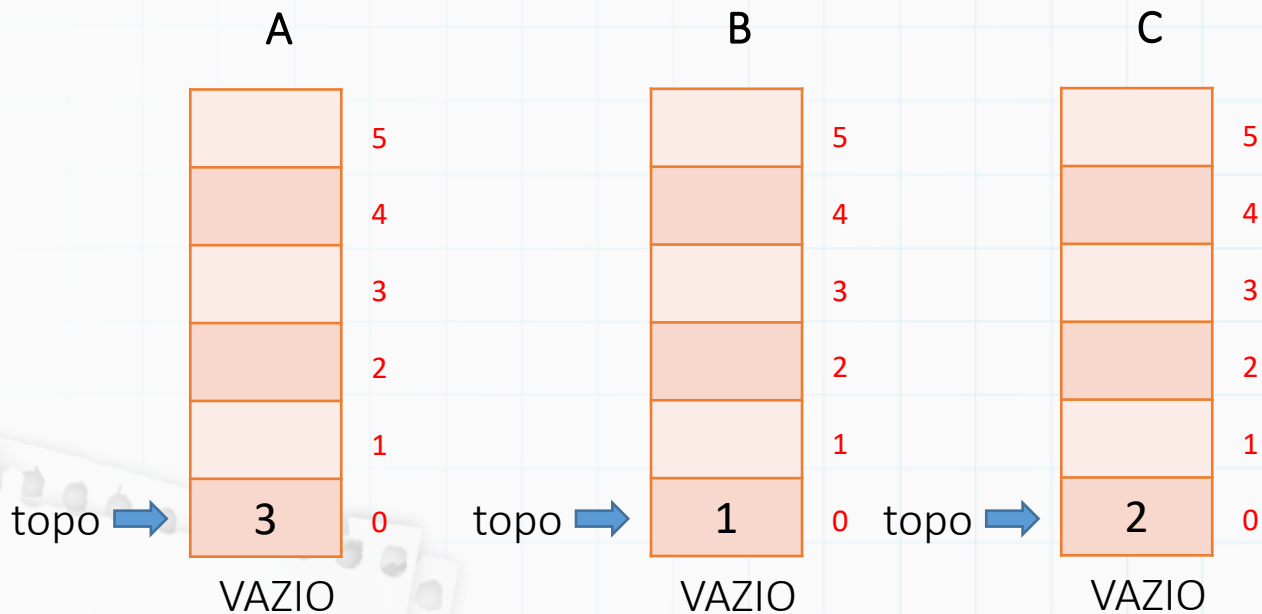
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

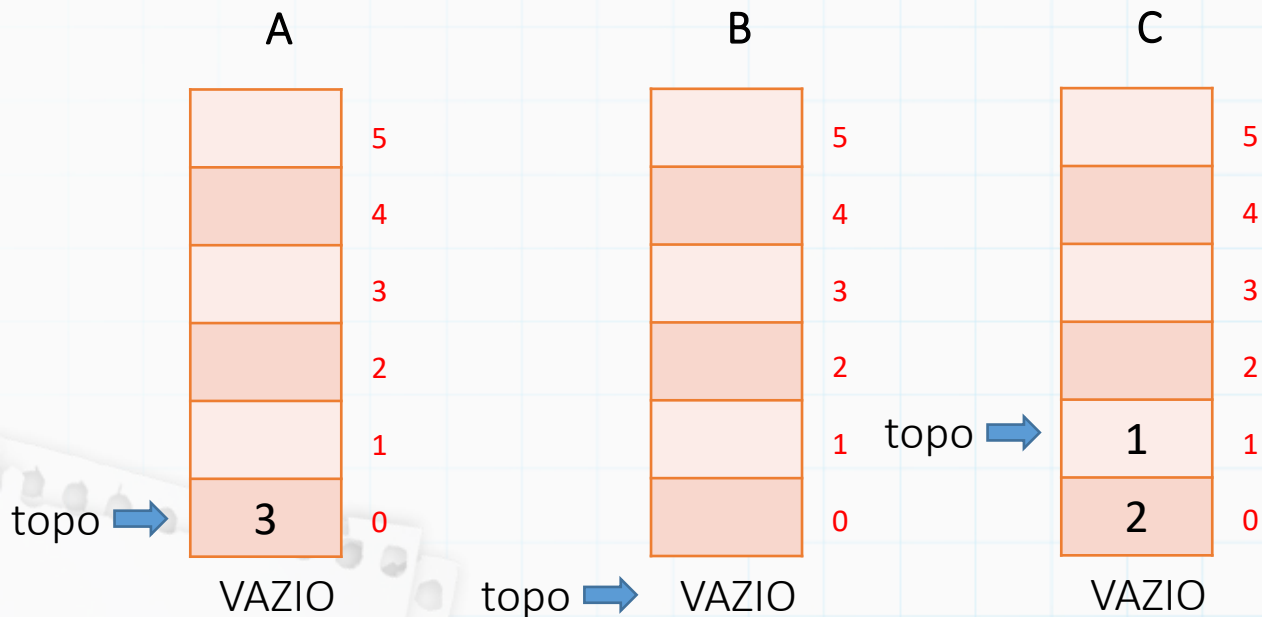
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3

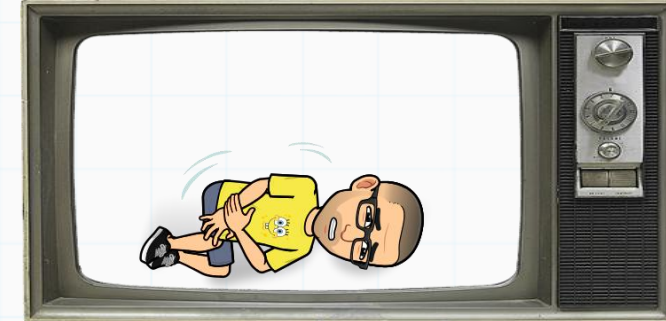


DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

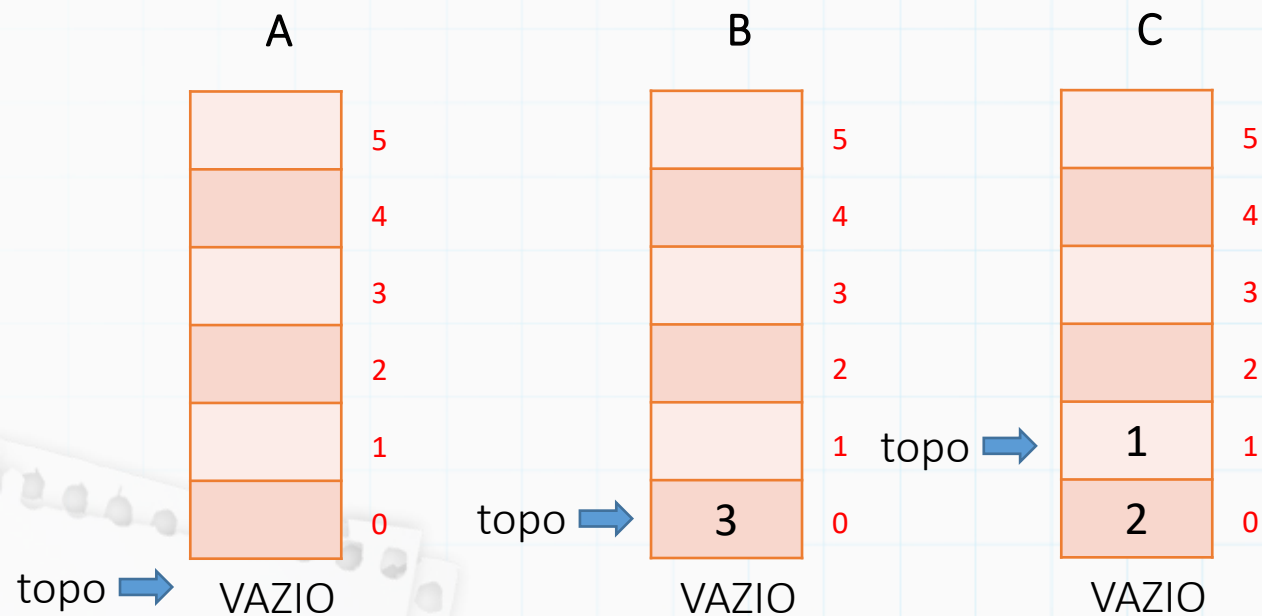
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

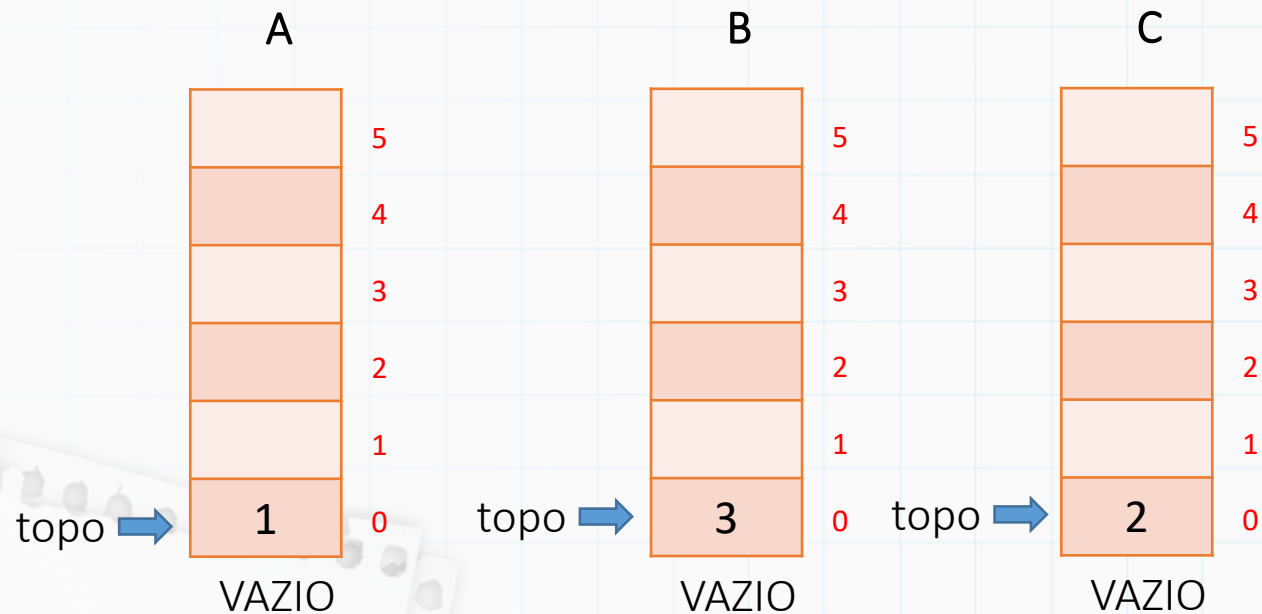
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

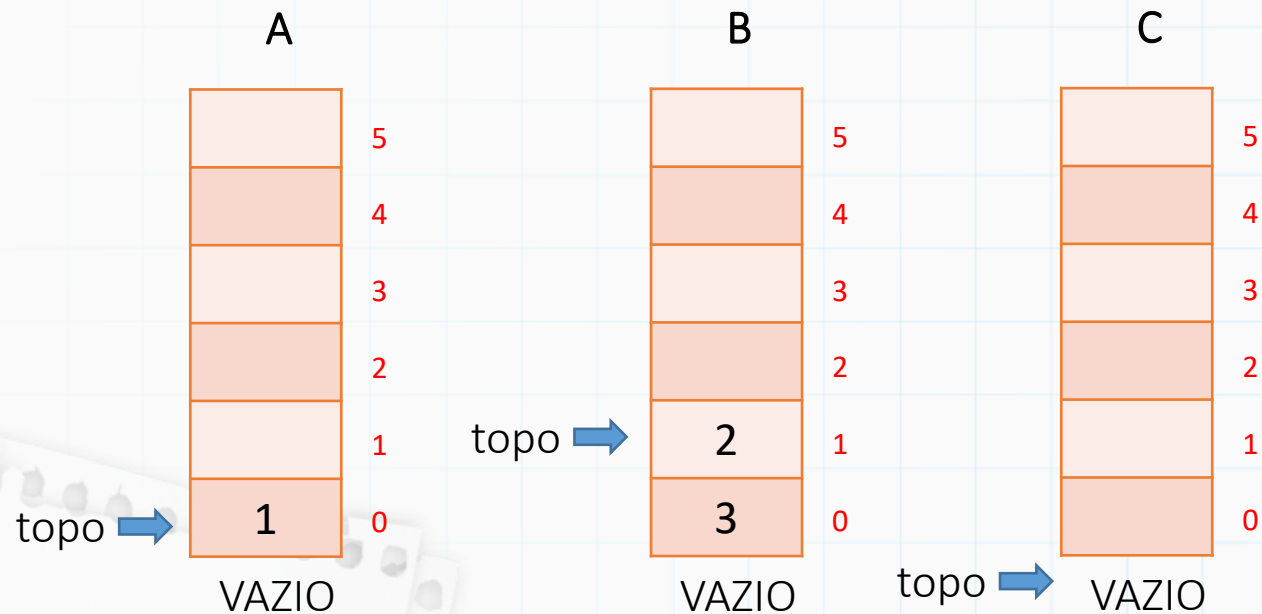
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

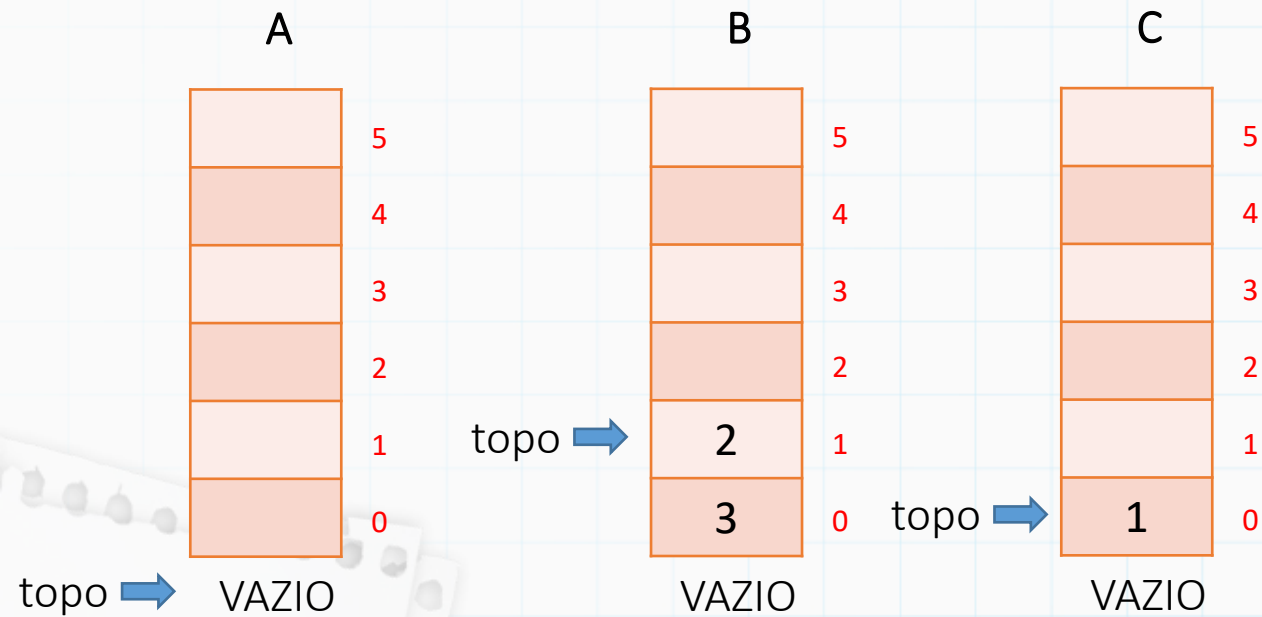
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

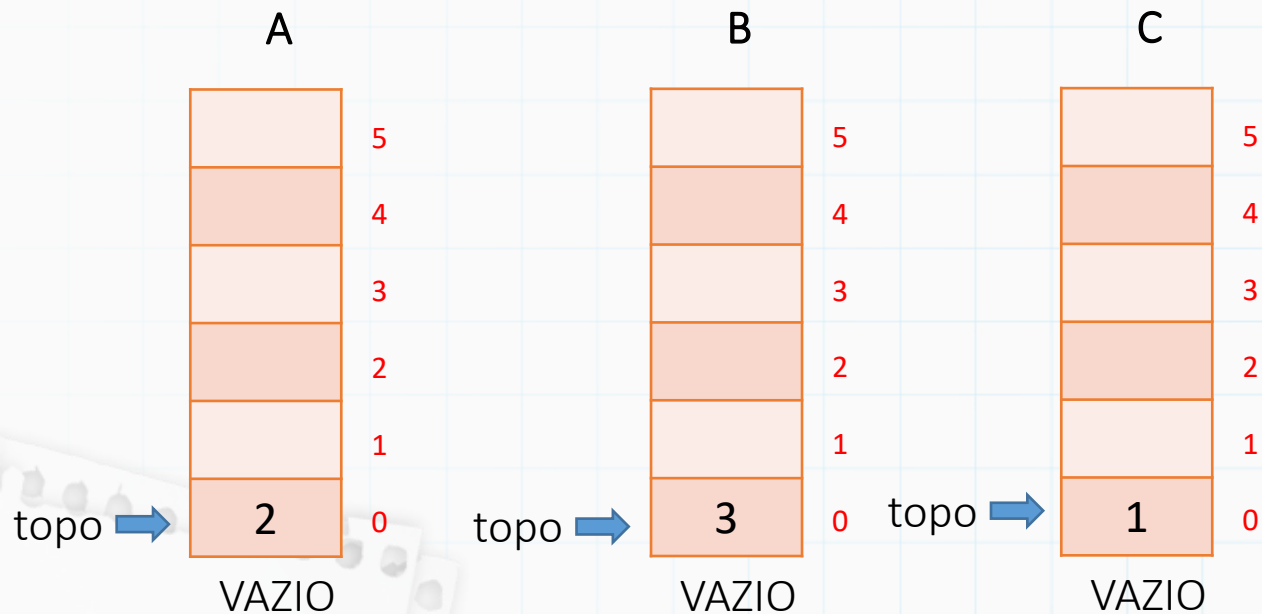
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

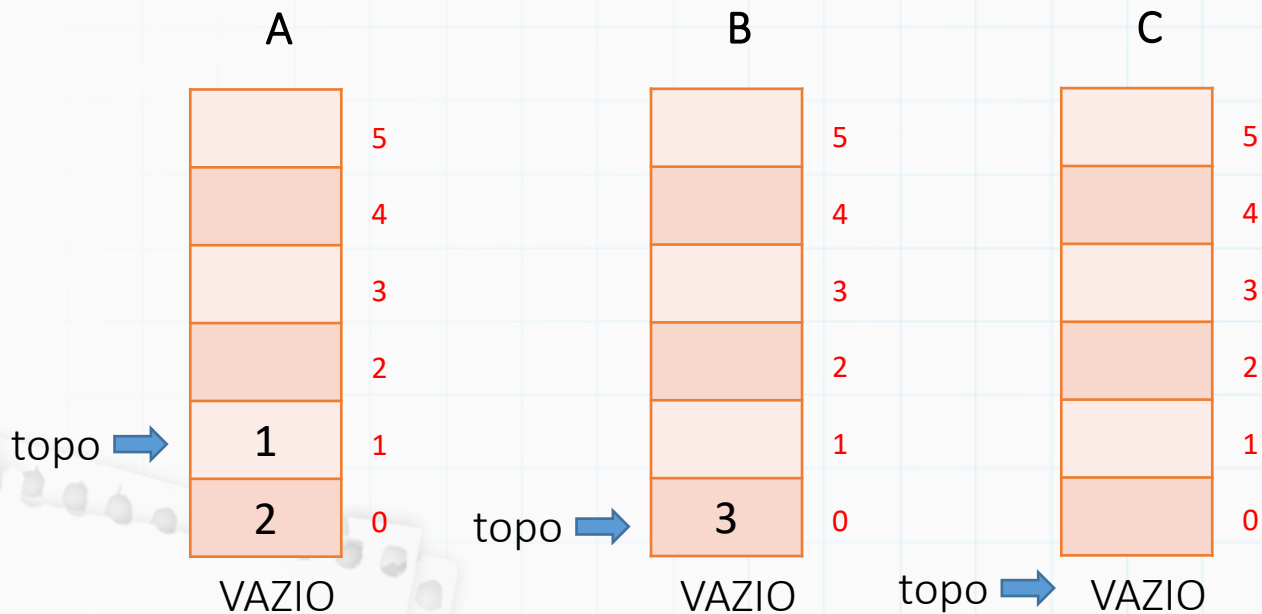
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

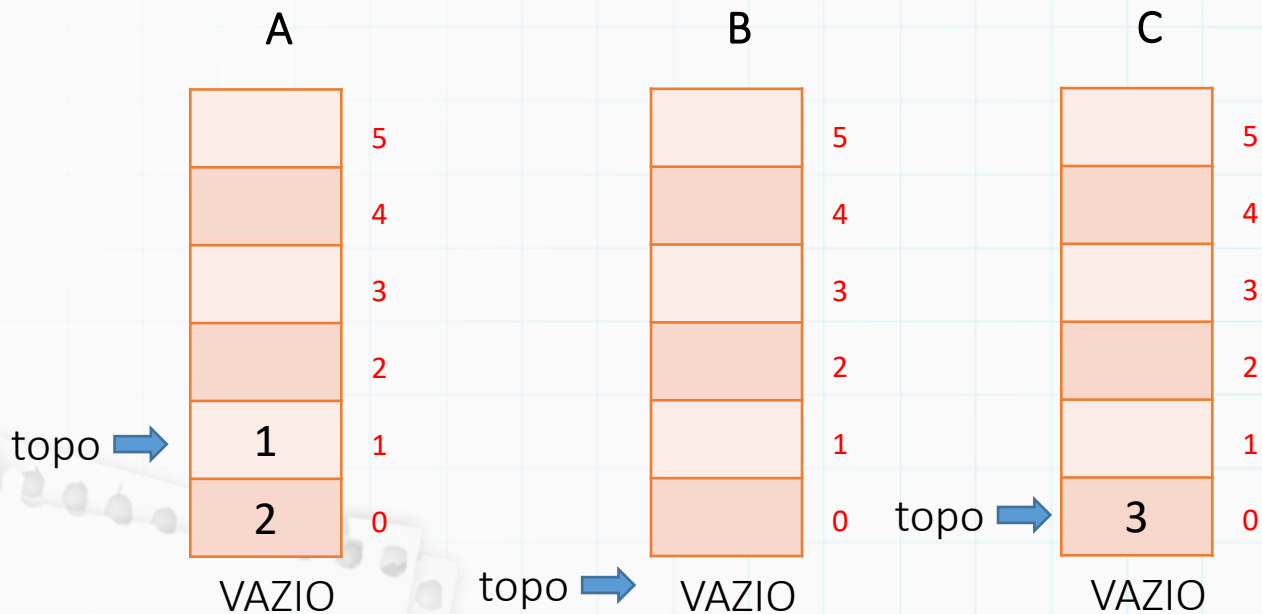
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

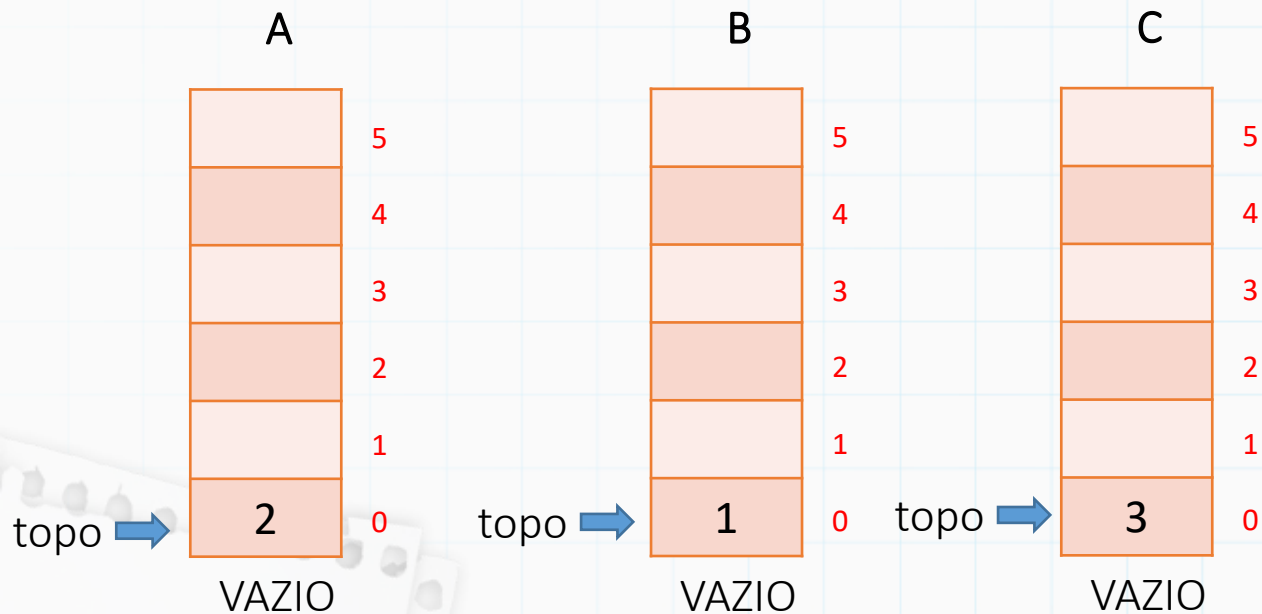
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

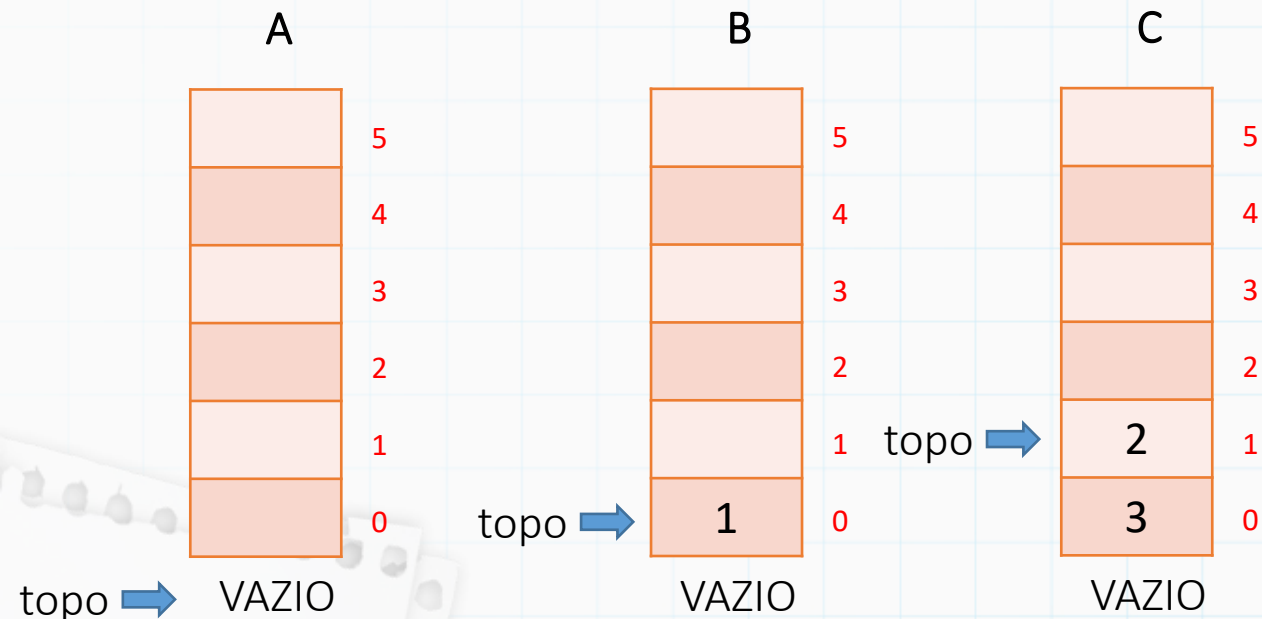
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:

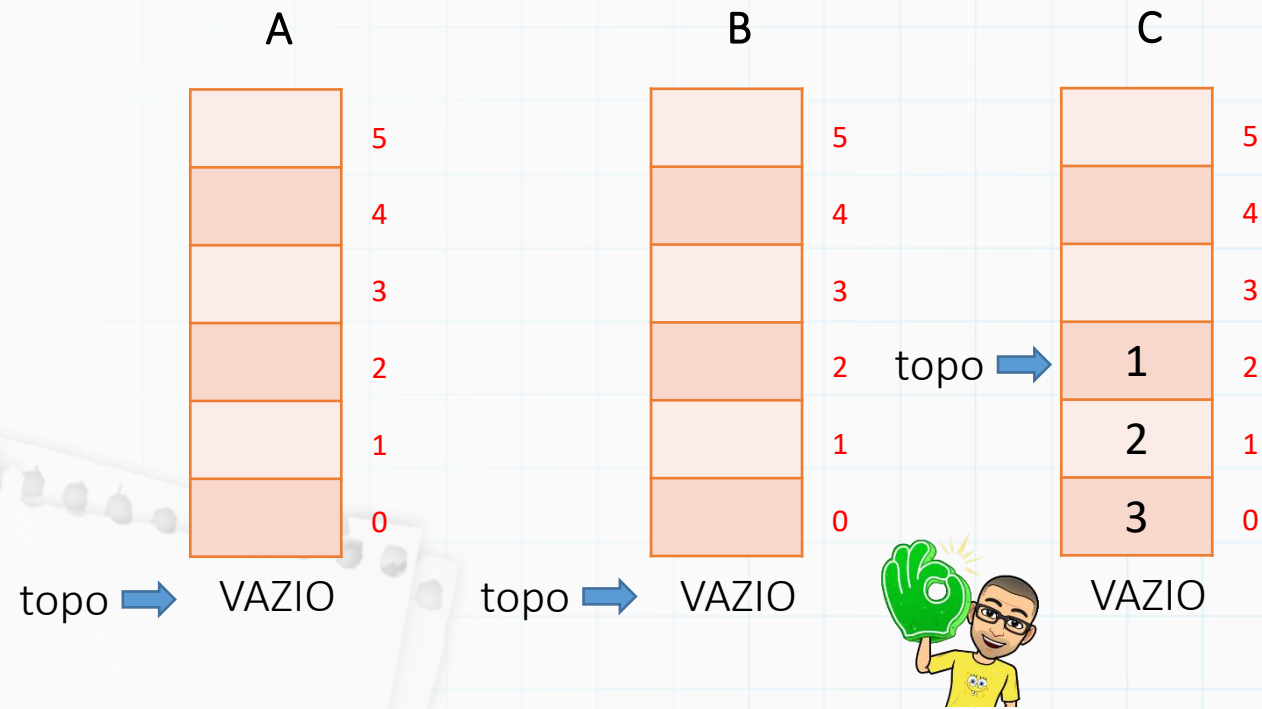
- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele



Use só o que aprendemos até hoje

NÃO USE OUTRAS ESTRUTURAS (VETORES, MATRIZES, LISTAS, ETC). APENAS USE 3 PILHAS A,B,C

Vamos ver a sequencia de movimentos da solução para n=3



DESAFIO

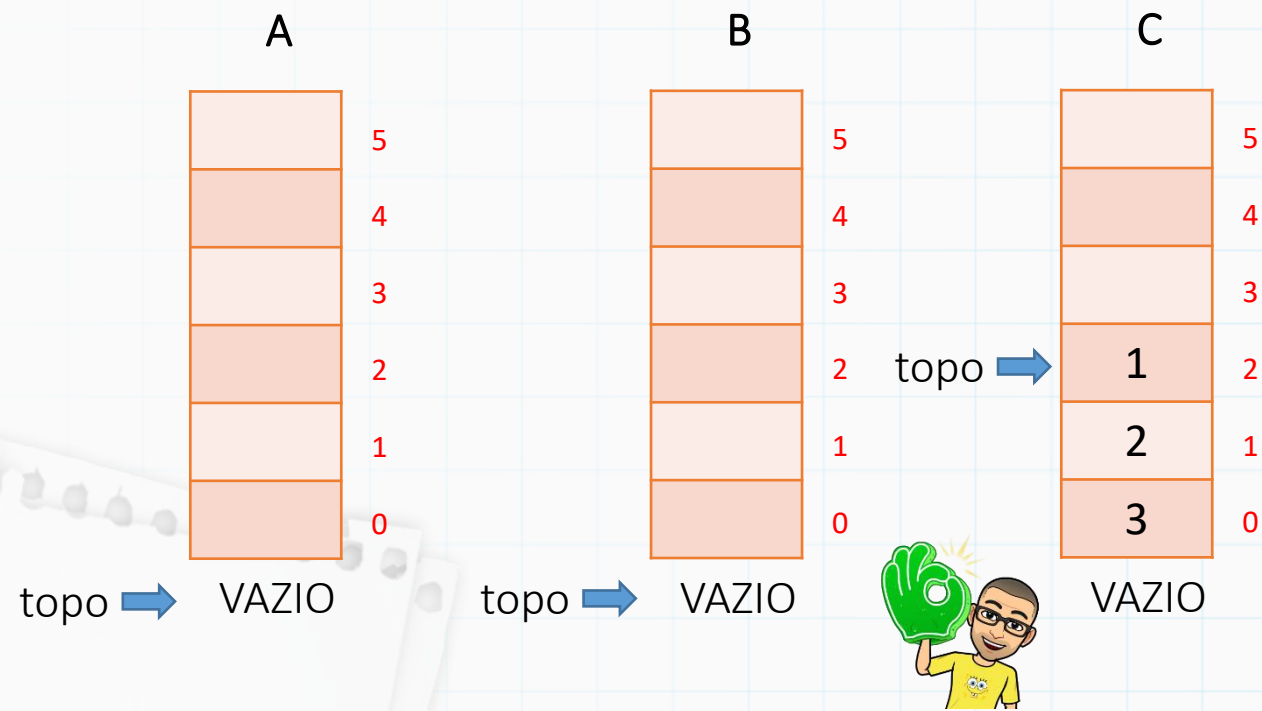
TAD - Pilha

6) Torres de Hanoi: As torres de Hanoi são um puzzle matemático em que temos 3 pilhas (A, B e C) onde inicialmente a pilha A possui n elementos ordenados do maior para o menor. O objetivo do jogo é levar todos os elementos para a pilha C, seguindo as seguintes regras:



- Apenas um elemento pode ser movido por vez
- Um elemento não pode ser empilhado em cima de outro elemento menor que ele

Vamos ver a sequencia de movimentos da solução para n=3



A solução pode ser alcançado pela repetição dos seguintes passos ordenados:

- 1) Move elemento entre A e B: Move de A para B se for possível SENÃO de B para A se for possível
- 2) Move elemento entre A e C: Move de A para C se for possível SENÃO de C para A se for possível
- 3) Move elemento entre B e C: Move de B para C se for possível SENÃO de C para B se for possível

Um movimento de X para Y é possível se X não for vazio E o valor do topo de X não é maior do que o topo de Y

Veja a execução a seguir:



6) Torres de Hanoi

TAD - Pilha



Código da main.c

```
void imprime_pilhas_hanoi(pilha * A, pilha * B, pilha * C, int n)
{
    for (int i=n-1; i>=0; i--)
    {
        if (A->topo >= i)
            printf("| %2d |\t\t", A->v[i]);
        else
            printf("|      |\t\t");

        if (B->topo >= i)
            printf("| %2d |\t\t", B->v[i]);
        else
            printf("|      |\t\t");

        if (C->topo >= i)
            printf("| %2d |\n", C->v[i]);
        else
            printf("|      |\n");

    }
    printf("\n -A-  \t\t -B-  \t\t -C-  \n\n");
}
```

```
int main()
{
    int n = 3;

    pilha A, B, C;
    cria_pilha(&A, n);
    cria_pilha(&B, n);
    cria_pilha(&C, n);

    for (int i=n; i>=1; i--)
        push_pilha(&A, i);
    imprime_pilhas_hanoi(&A, &B, &C, n);

    hanoi(&A, &B, &C, n);

    return 0;
}
```

DESAFIO

TAD - Pilha



6) Torres de Hanoi

Execução

```
| 1 | | |
| 2 | | |
| 3 | | |
```

-A- -B- -C-

move entre A e B

```
| | | |
| 2 | | |
| 3 | | 1 |
```

-A- -B- -C-

move entre A e C

```
| | | |
| 3 | | 1 |
| | | 2 |
```

-A- -B- -C-

move entre B e C

```
| | | |
| | | 1 |
| 3 | | 2 |
```

-A- -B- -C-

move entre A e B

```
| | | |
| | | 1 |
| | | 3 |
| | | 2 |
```

-A- -B- -C-

move entre A e C

```
| | | |
| 1 | | 3 |
| | | 2 |
```

-A- -B- -C-

move entre B e C

```
| | | |
| 1 | | 2 |
| | | 3 |
```

-A- -B- -C-

move entre A e B

```
| | | |
| | | 1 |
| | | 2 |
| | | 3 |
```

-A- -B- -C-

move entre B e C

```
| | | |
| | | 2 |
| | | 3 |
| | | 1 |
```

-A- -B- -C-



TAD - Pilha

6) Torres de Hanoi Execução



move entre A e B

2	3	1	

-A- -B- -C-

move entre A e C

1			
2	3		

-A- -B- -C-

move entre B e C

1			
2		3	

-A- -B- -C-

move entre A e B

2	1	3	

-A- -B- -C-

move entre A e C

		1		2	3

-A- -B- -C-

move entre B e C

				1	
				2	
				3	

-A- -B- -C-

para n=7, ao aplicarmos o método descrito para resolver as torres de Hanoi, quantas operações de pop na pilha C seriam realizadas ?



TAD - Pilha

para uma sequencia dada por (1, 2, 3, 4, 5, -6, 3, 3, 3, 3, -30), qual seria a soma de dos os elementos retirados (pops) durante o processo ?



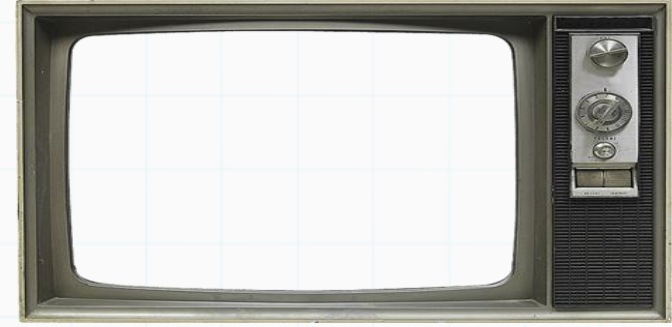
para uma pilha dada por (0, 5, 30, 40, 9, 22, 100), onde 0 é a base e 100 é o topo da pilha, ao aplicarmos o Stack-Sort como descrito, quantas operações de push na pilha auxiliar seriam realizadas ao ordenar a pilha ?



para $n=7$, ao aplicarmos o método descrito para resolver as torres de Hanoi, quantas operações de pop na pilha C seriam realizadas ?



Até a próxima



Slides baseados no curso de Aline Nascimento