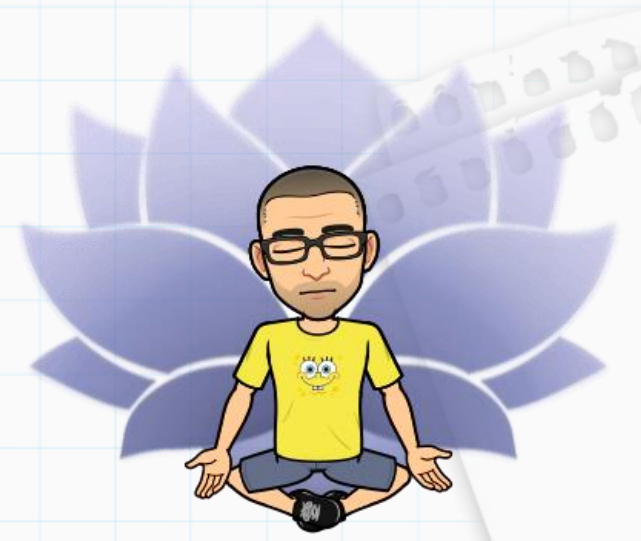
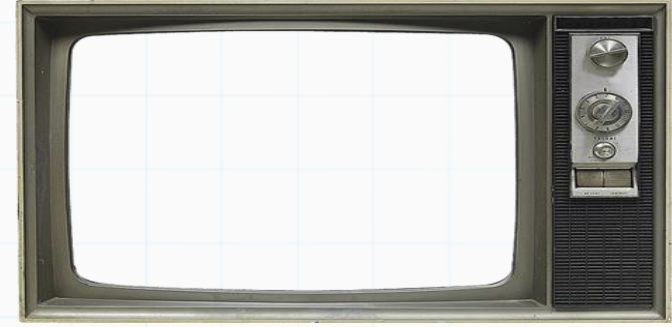


Programação Estruturada

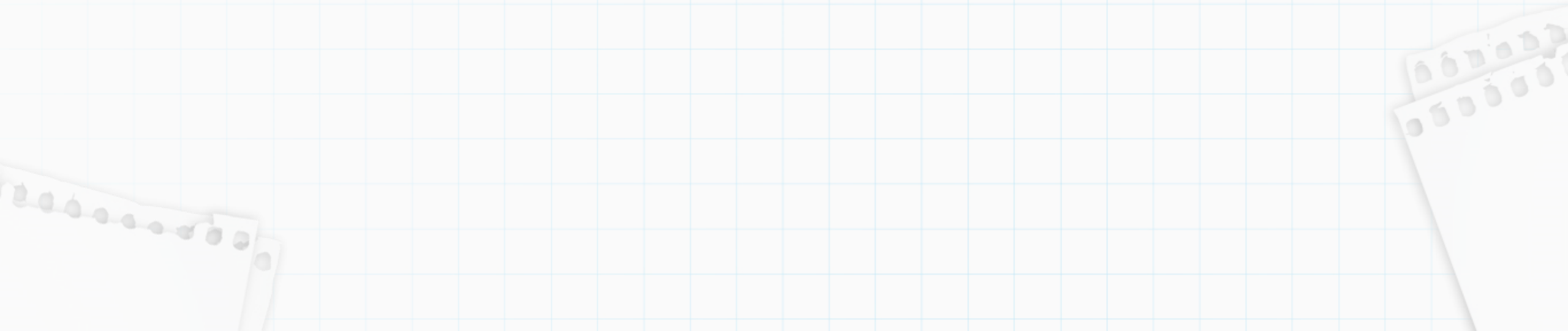
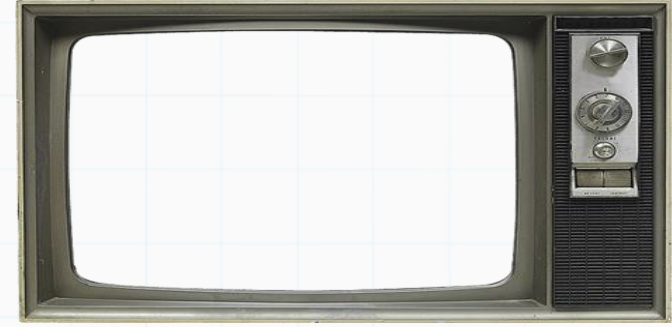
Professor : Yuri Frota

yuri@ic.uff.br



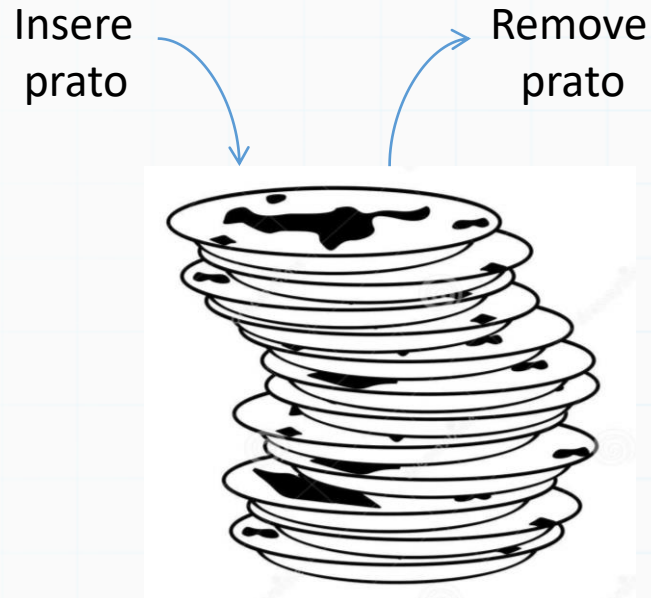
Pilhas

- Pilhas são listas com acessos com operações especiais (acessos simplificados e restritos)
- São estruturas do tipo **LIFO** (last in – first out)

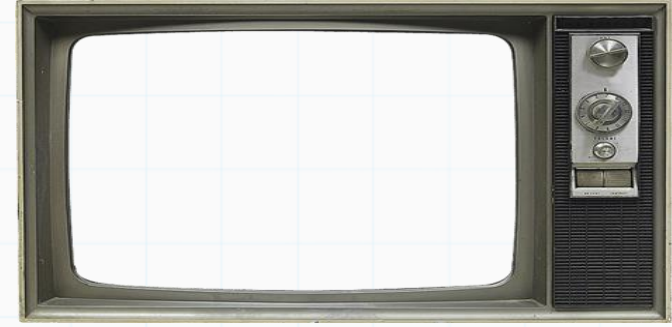


Pilhas

- Pilhas são listas com acessos com operações especiais (acessos simplificados e restritos)
- São estruturas do tipo **LIFO** (last in – first out)

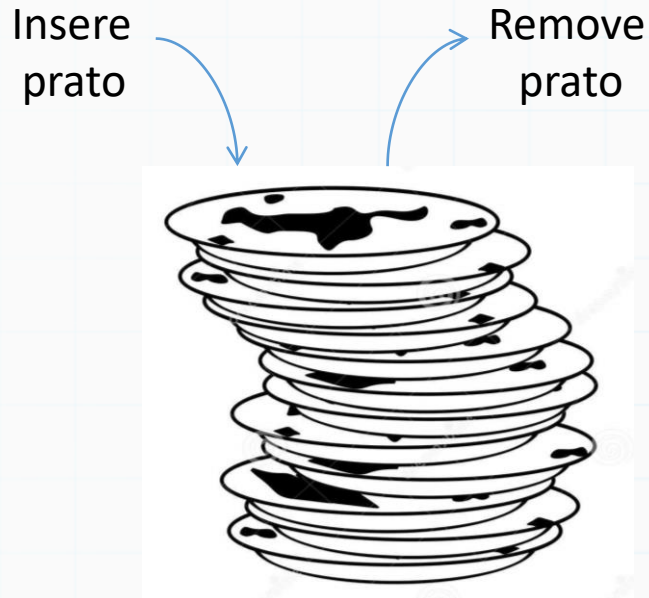


se tirar ou colocar no
meio vai cair tudo

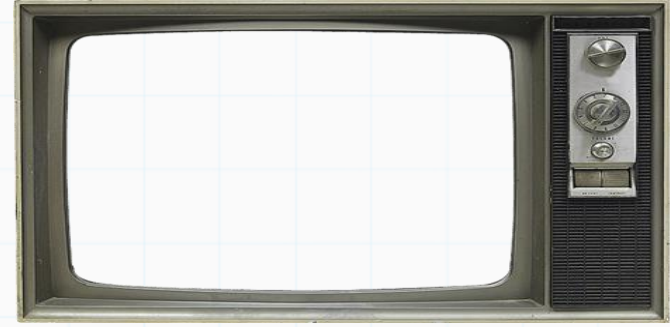


Pilhas

- Pilhas são listas com acessos com operações especiais (acessos simplificados e restritos)
- São estruturas do tipo **LIFO (last in – first out)**



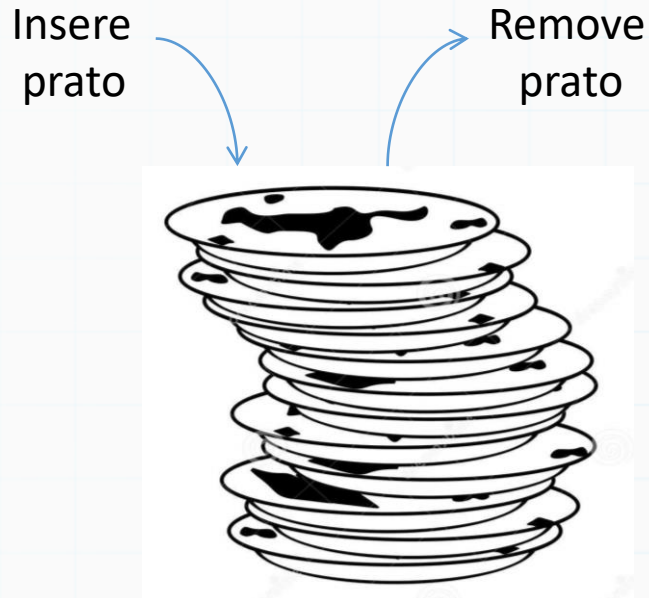
se tirar ou colocar no meio vai cair tudo



- Funções Recursivas são implementadas com pilhas:
 - A cada nova chamada da função recursiva, uma instancia da função é empilhada.
 - Ao retornar, as instancias são desempilhadas iterativamente.

Pilhas

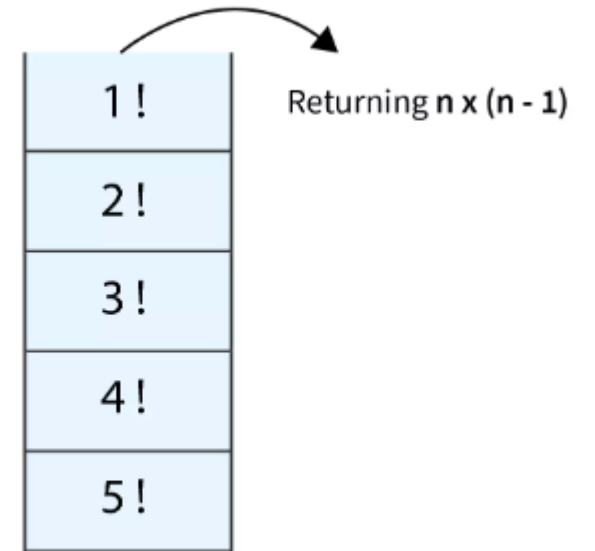
- Pilhas são listas com acessos com operações especiais (acessos simplificados e restritos)
- São estruturas do tipo **LIFO (last in – first out)**



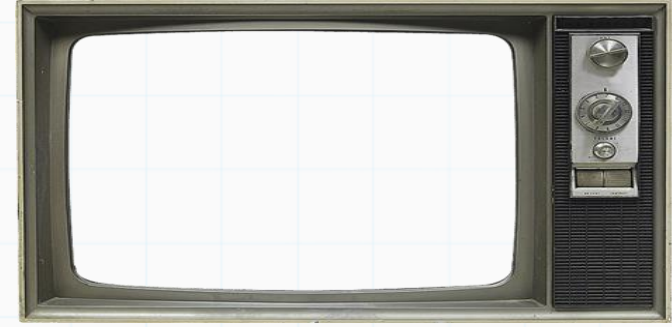
se tirar ou colocar no meio vai cair tudo



Exemplo do fatorial



Recursion Stack

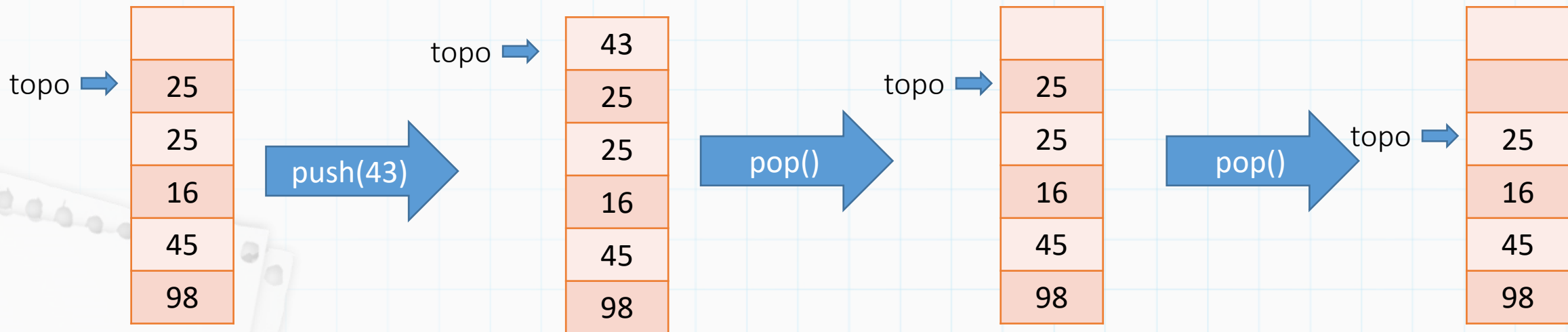
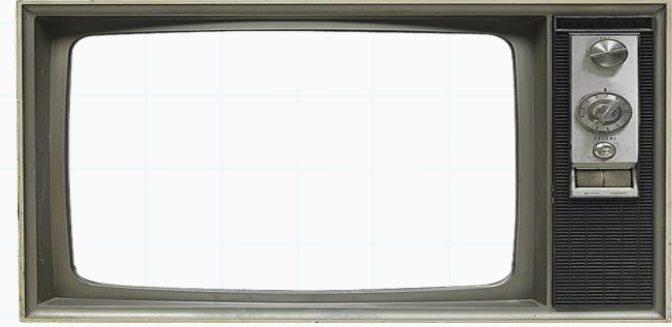


- Funções Recursivas são implementadas com pilhas:
 - A cada nova chamada da função recursiva, uma instancia da função é empilhada.
 - Ao retornar, as instancias são desempilhadas iterativamente.

Pilhas

TAD de Pilhas :

- Dados:
 - Elementos (são os dados)
 - Ponteiro para o topo da pilha
- Operações:
 - **PUSH(P, x)**: insere um elemento x no topo da pilha P (empilha).
 - **POP(P)**: retira o elemento do topo da pilha P (desempilha).



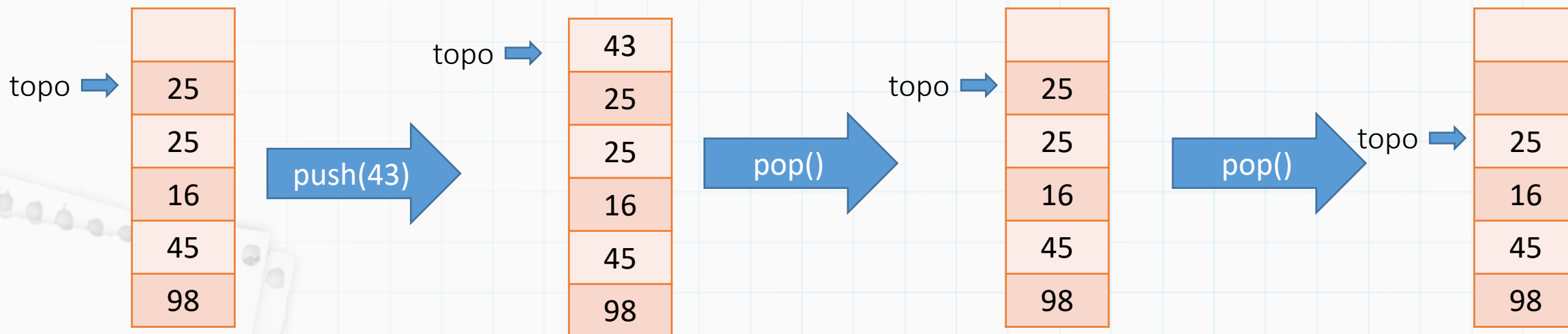
Pilhas

TAD de Pilhas :

- Dados:
 - Elementos (são os dados)
 - Ponteiro para o topo da pilha
- Operações:
 - **PUSH(P, x)**: insere um elemento x no topo da pilha P (empilha).
 - **POP(P)**: retira o elemento do topo da pilha P (desempilha).
 - **Top(P)**: Checa o valor do elemento no topo (sem retirar)
 - **Vazia(P)**: Checa se pilha vazia.
 - **Cheia(P)**: Checa se pilha cheia.

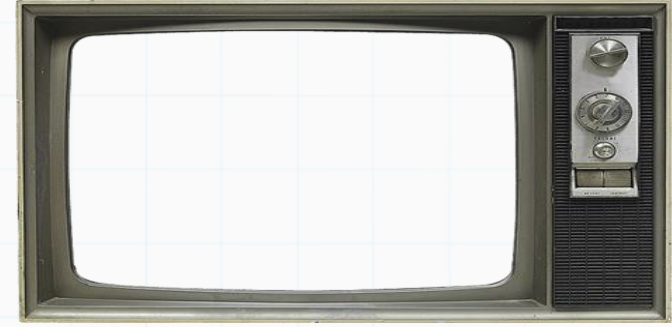


outras operações



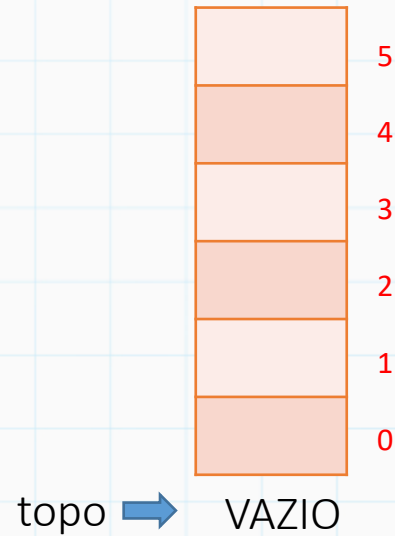
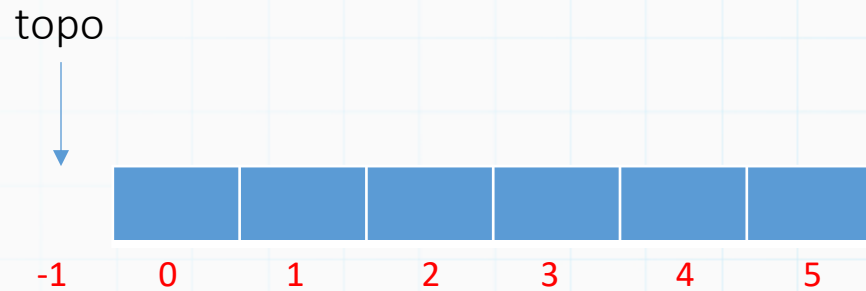
Pilhas

- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo



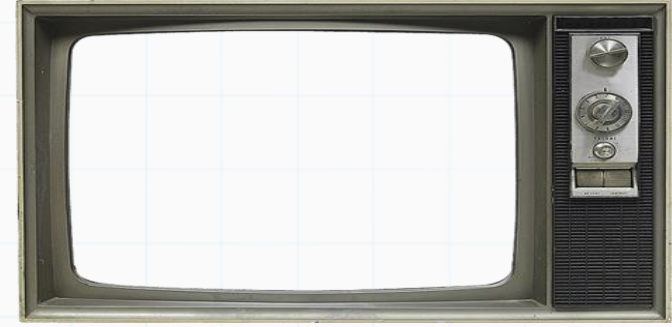
REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



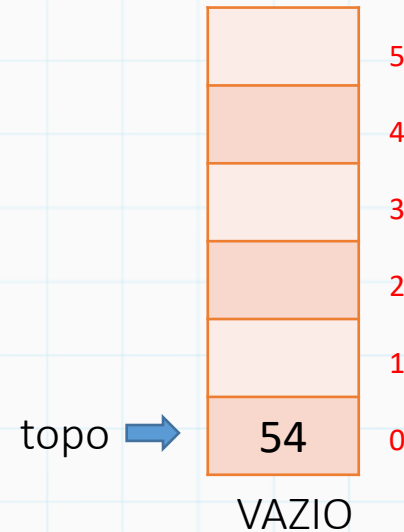
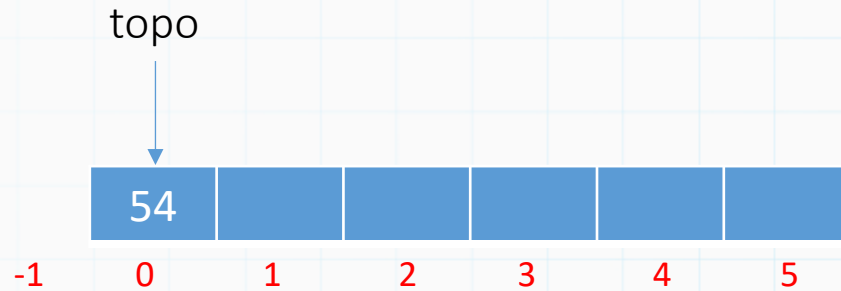
Pilhas

- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54-> PUSH(P,54)



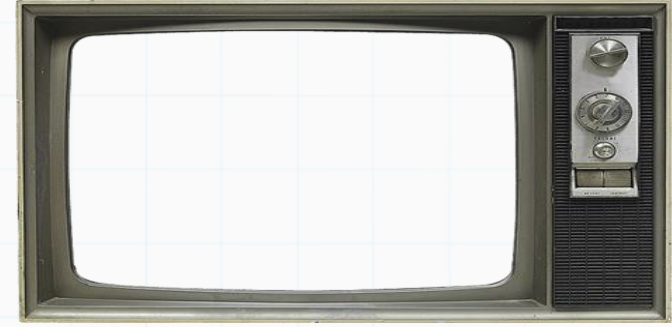
REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



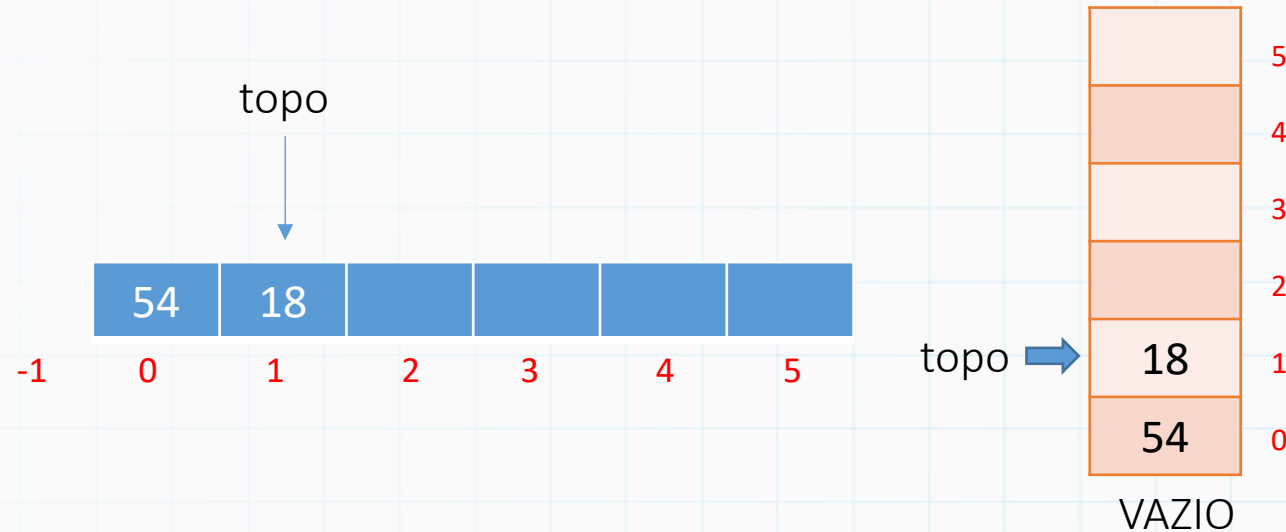
Pilhas

- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 -> PUSH(P,54), PUSH(P,18)



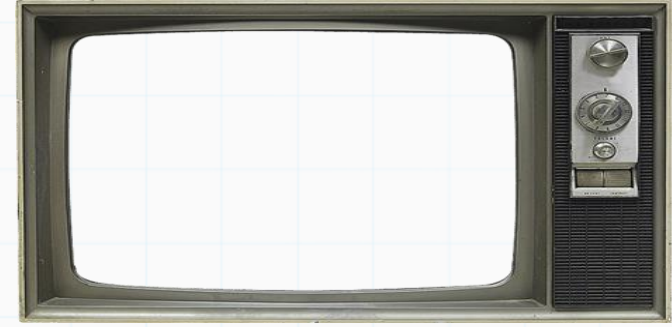
REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



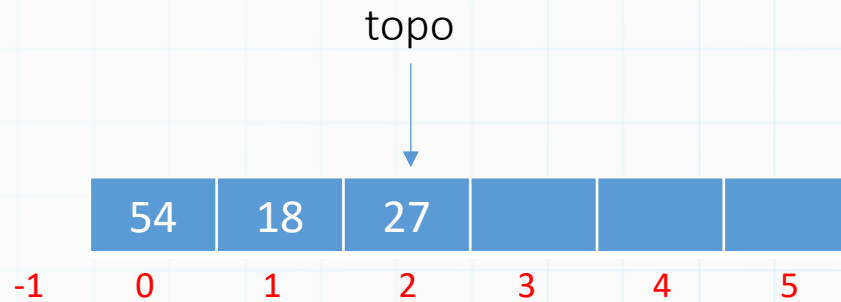
Pilhas

- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)



REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



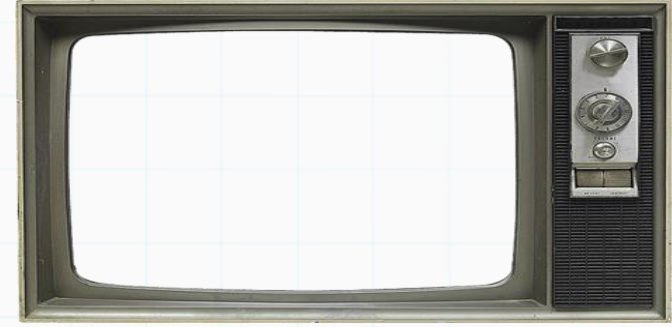
topo →



VAZIO

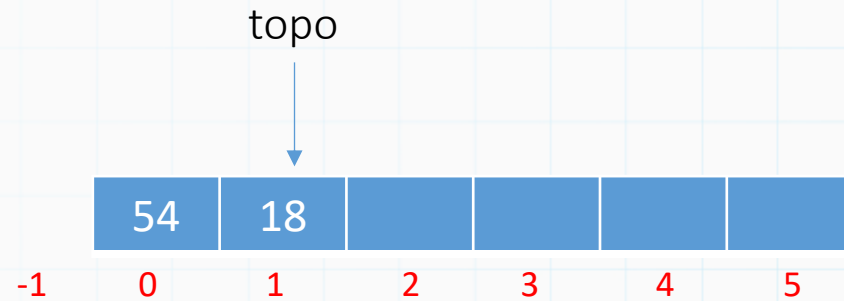
Pilhas

- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)

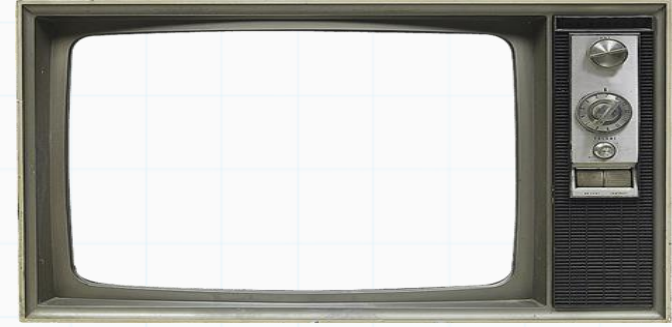


REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



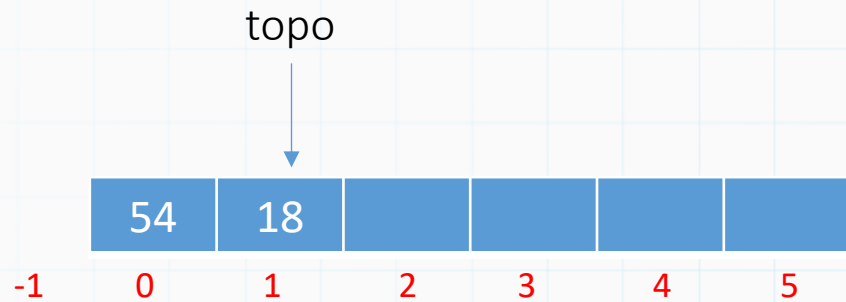
Pilhas



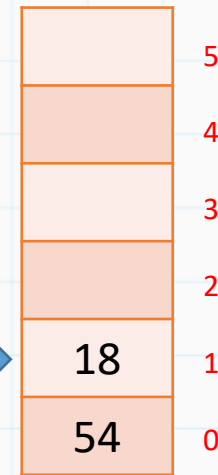
- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)

REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



topo →



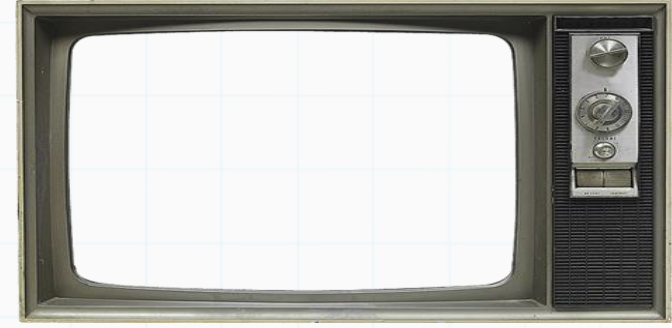
VAZIO

```
void cria_pilha(pilha * P, int tam)
{
    P->topo = -1;
    P->tam = tam;
    P->v = (int*) malloc( P->tam * sizeof(int));
}
```

```
struct PILHA {
    int tam;
    int topo;
    int *p;
};
typedef struct PILHA pilha;
```

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

Pilhas

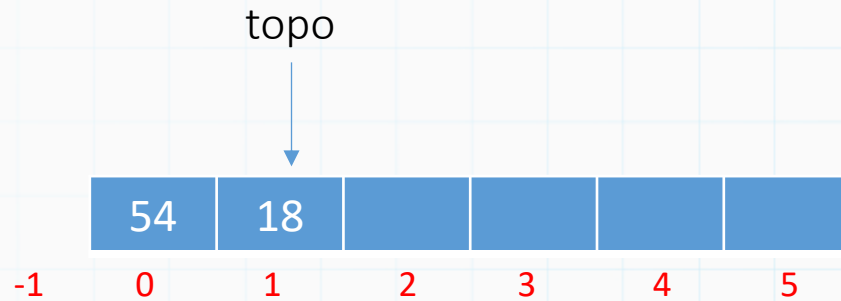


- Pilhas em vetores:

- Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
- Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
- Vamos retirar o topo da pilha -> POP(P)

REPRESENTAÇÃO GRÁFICA DE PILHA

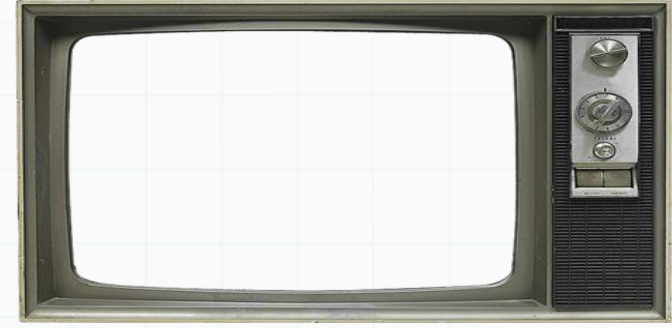
VETOR



```
int vazia_pilha(pilha * P)
{
    if (P->topo == -1)
        return 1;
    else
        return 0;
}
```

```
int cheia_pilha(pilha * P)
{
    if (P->topo == P->tam-1)
        return 1;
    else
        return 0;
}
```

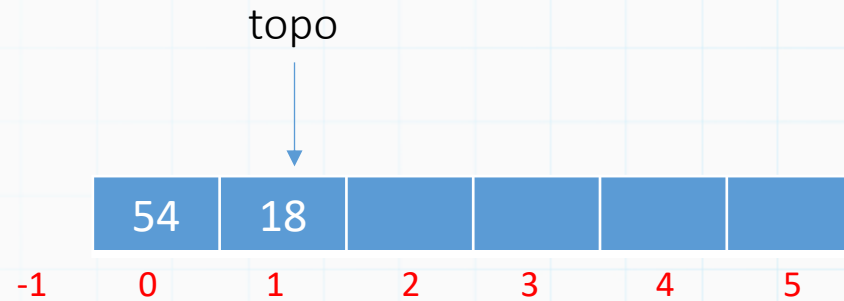
Pilhas



- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)

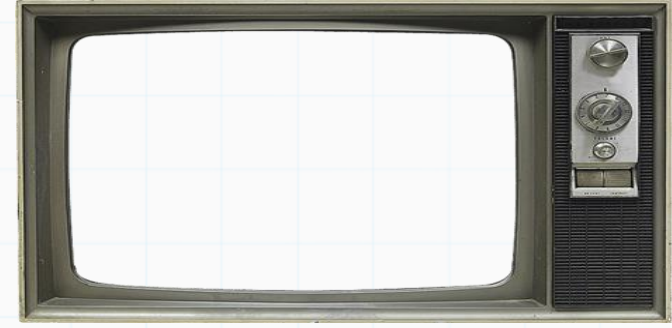
REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



```
void push_pilha(pilha * P, int el)
{
    if (P->topo < P->tam-1)
    {
        P->topo++;
        P->v[P->topo] = el;
    }
    else
        printf("pilha cheia\n");
}
```

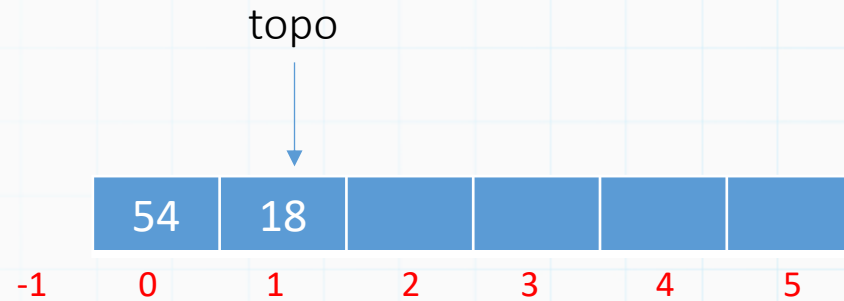

Pilhas



- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)

REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR

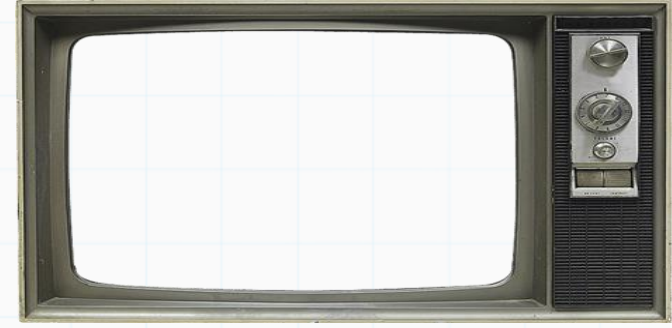


```
int pop_pilha(pilha * P)
{
    int el;

    if (P->topo >= 0)
    {
        el = P->v[P->topo];
        P->topo--;
    }
    else
        printf("pilha vazia\n");

    return el;
}
```

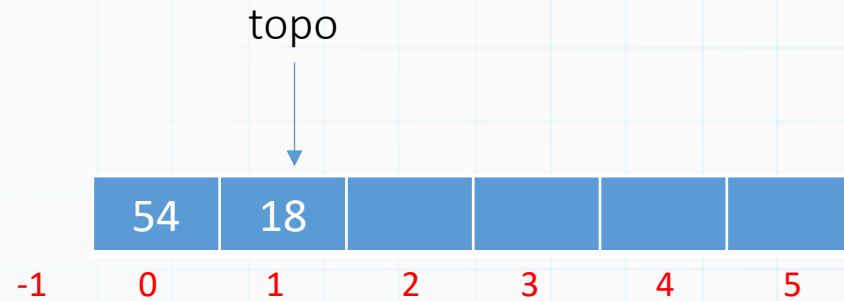
Pilhas



- Pilhas em vetores:
 - Usaremos vetor com o tamanho máximo da pilha (ex: 6) e um índice para apontar para o topo
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)

REPRESENTAÇÃO GRÁFICA DE PILHA

VETOR



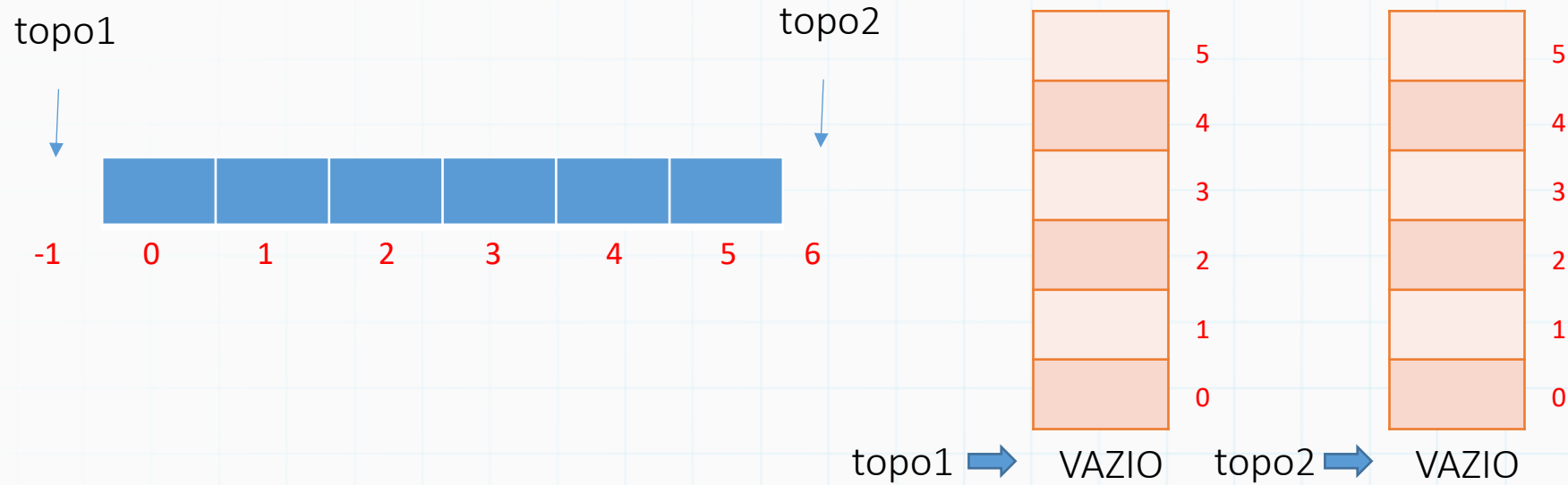
```
int top_pilha(pilha * P)
{
    int el;

    if (P->topo >= 0)
        el = P->v[P->topo];
    else
        printf("pilha vazia\n");

    return el;
}
```

Pilhas

Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



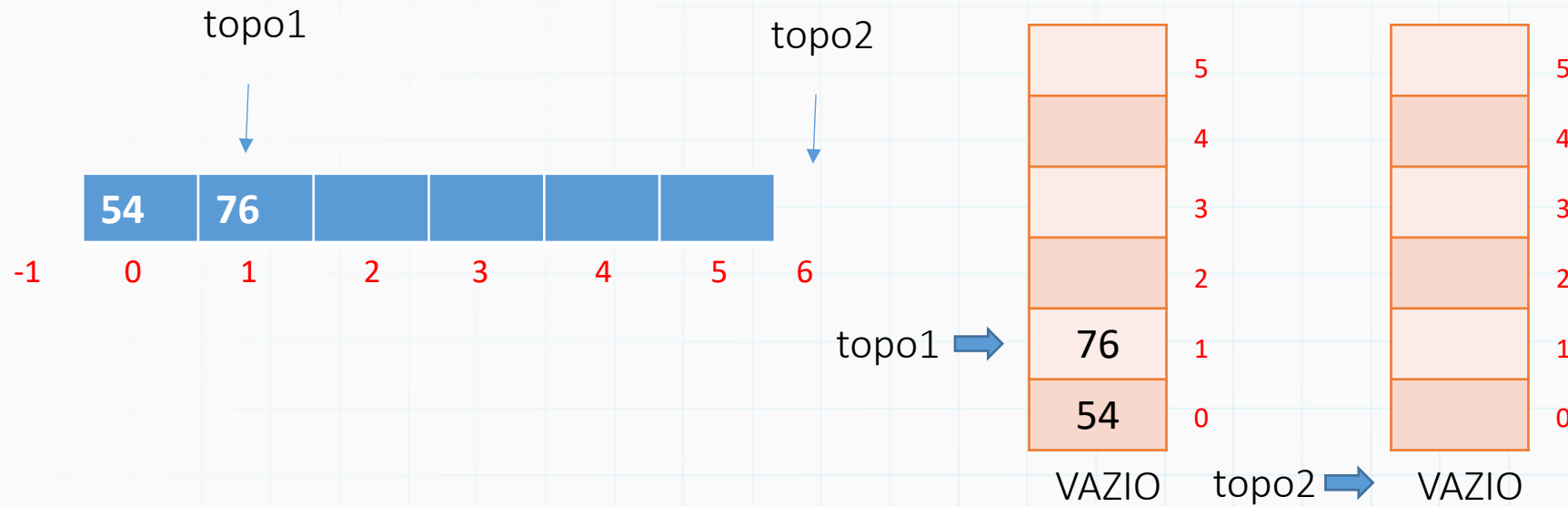
- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6

```
struct PILHAS
{
    int tam;
    int topo1;
    int topo2;
    int *v;
};
typedef struct PILHAS pilhas;
```

```
void cria_pilhas(pilhas * P, int tam)
{
    P->topo1 = -1;
    P->topo2 = tam;
    P->tam = tam;
    P->v = (int*) malloc( P->tam * sizeof(int));
}
```

Pilhas

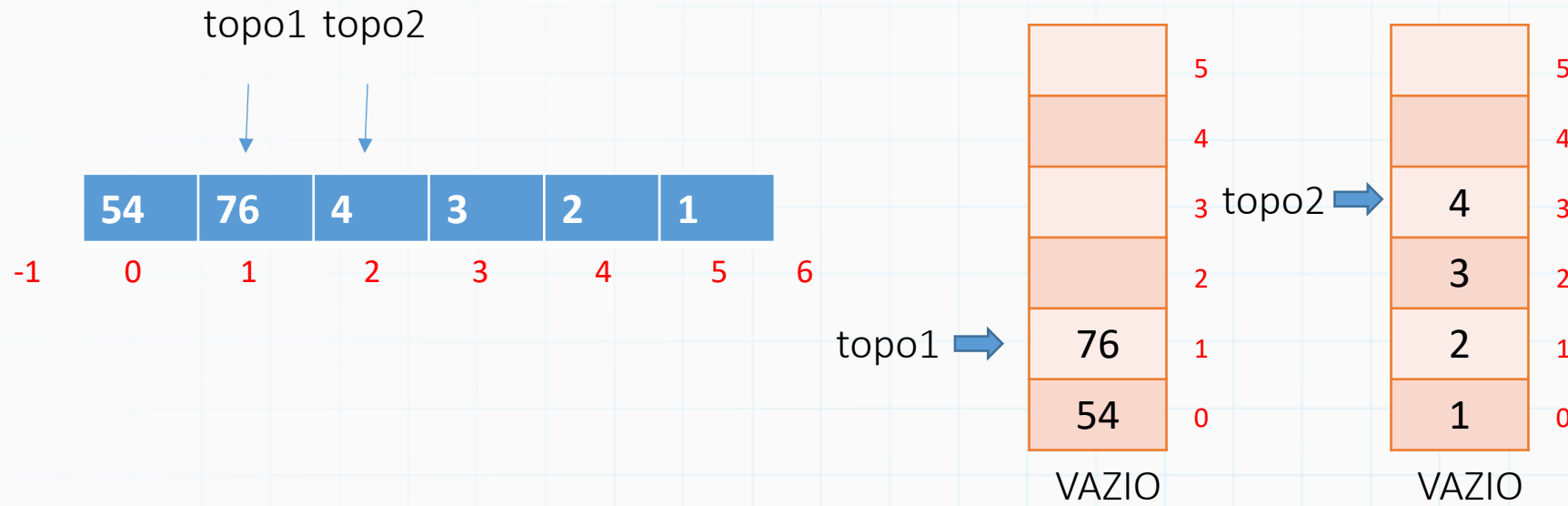
Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6
- vamos 54 e 76 na pilha 1 -> `PUSH(P, 1, 54)` e `PUSH(P, 1, 76)`

Pilhas

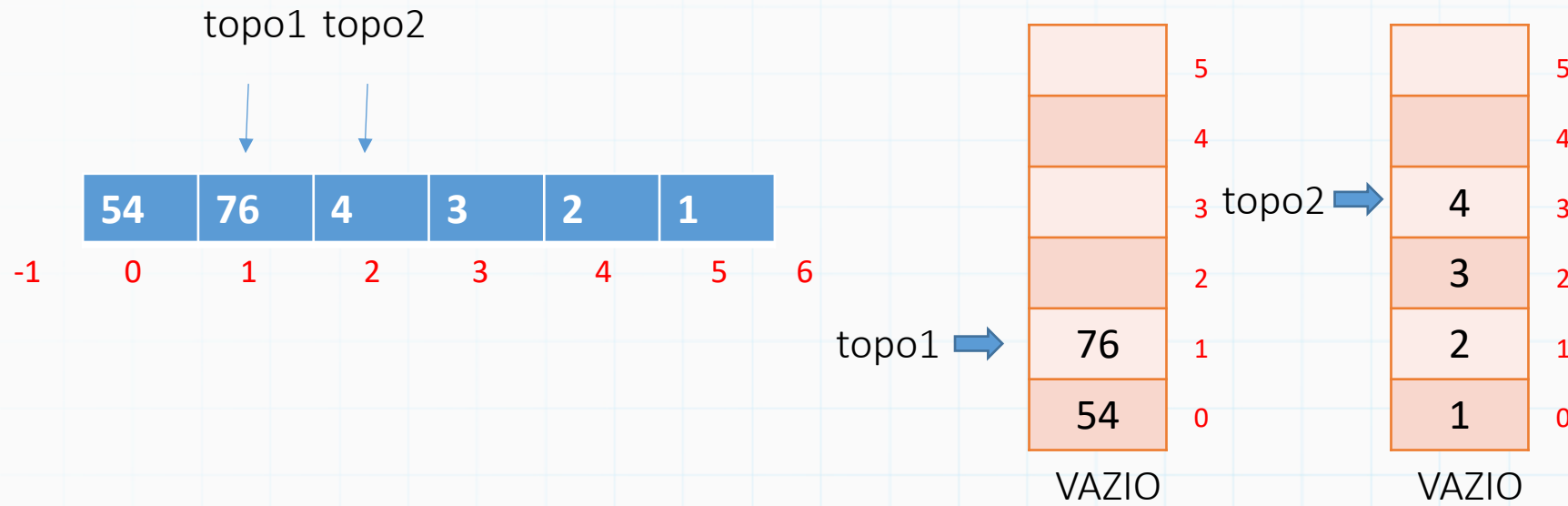
Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6
- vamos 54 e 76 na pilha 1 -> PUSH(P, 1, 54) e PUSH(P, 1, 76)
- vamos 1, 2, 3 e 4 na pilha 2 -> PUSH(P, 2, 1), PUSH(P, 2, 2), PUSH(P, 2, 3) e PUSH(P, 2, 4)

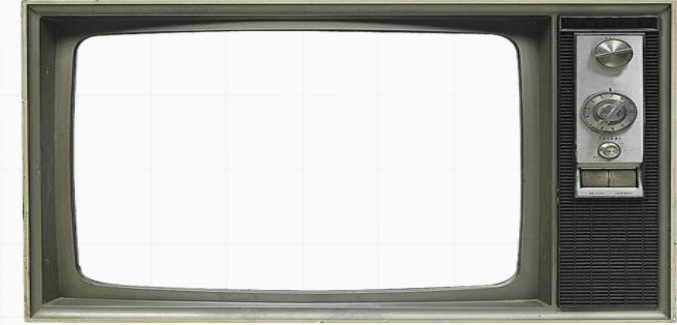
Pilhas

Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor

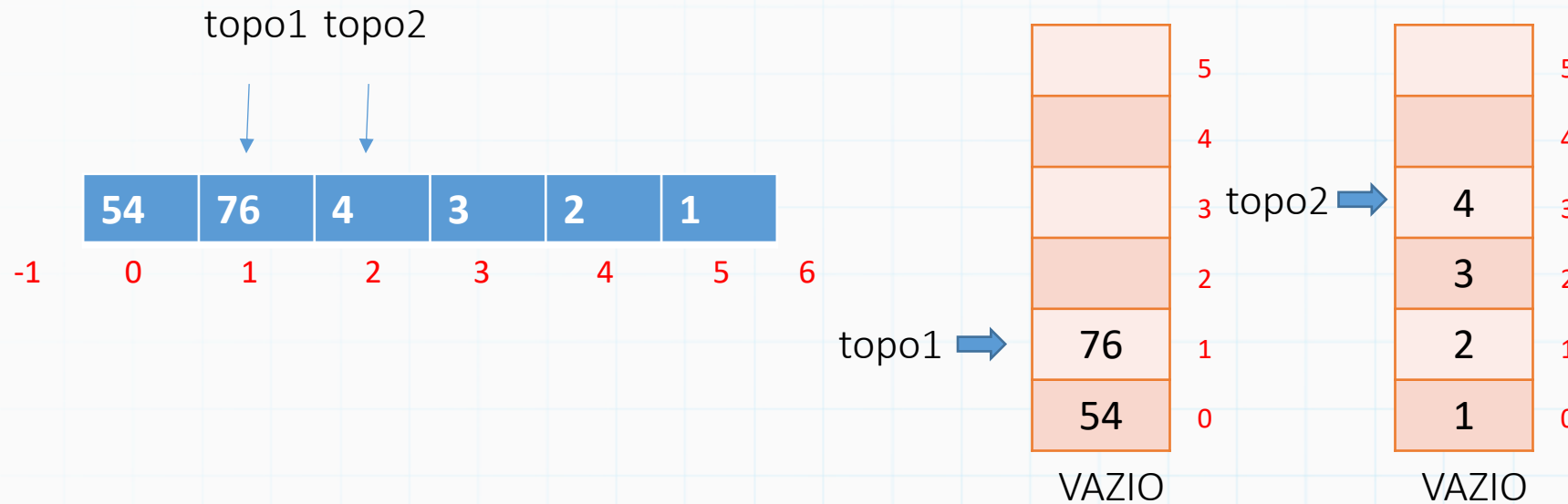


- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6
- vamos 54 e 76 na pilha 1 -> `PUSH(P, 1, 54)` e `PUSH(P, 1, 76)`
- vamos 1, 2, 3 e 4 na pilha 2 -> `PUSH(P, 2, 1)`, `PUSH(P, 2, 2)`, `PUSH(P, 2, 3)` e `PUSH(P, 2, 4)`
- vamos 130 na pilha 1 -> `PUSH(P, 1, 130)` -> **Não dá, ta cheio**

Pilhas



Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6
- vamos 54 e 76 na pilha 1 -> PUSH(P, 1, 54) e PUSH(P, 1, 76)
- vamos 1, 2, 3 e 4 na pilha 2 -> PUSH(P, 2, 1), PUSH(P, 2, 2), PUSH(P, 2, 3) e PUSH(P, 2, 4)
- vamos 130 na pilha 1 -> PUSH(P, 1, 130) -> **Não dá, ta cheio**

```
struct PILHAS
{
    int tam;
    int topo1;
    int topo2;
    int *v;
};

typedef struct PILHAS pilhas;
```

Exercício 1) Codifique a função abaixo de push em pilhas duplas, onde num=1 -> push na pilha 1, num=2 -> push na pilha 2.

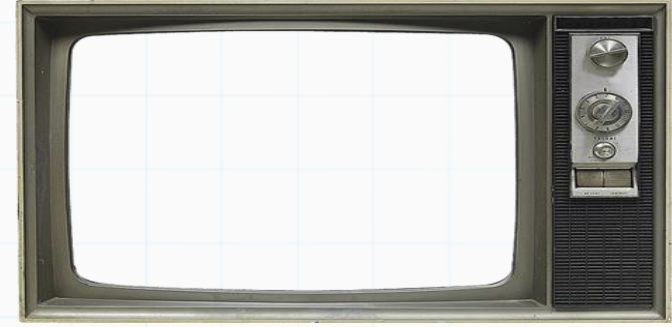
Cuidado com a pilha cheia

```
void push_pilhas(pilhas * P, int num, int el)
```


Pilhas

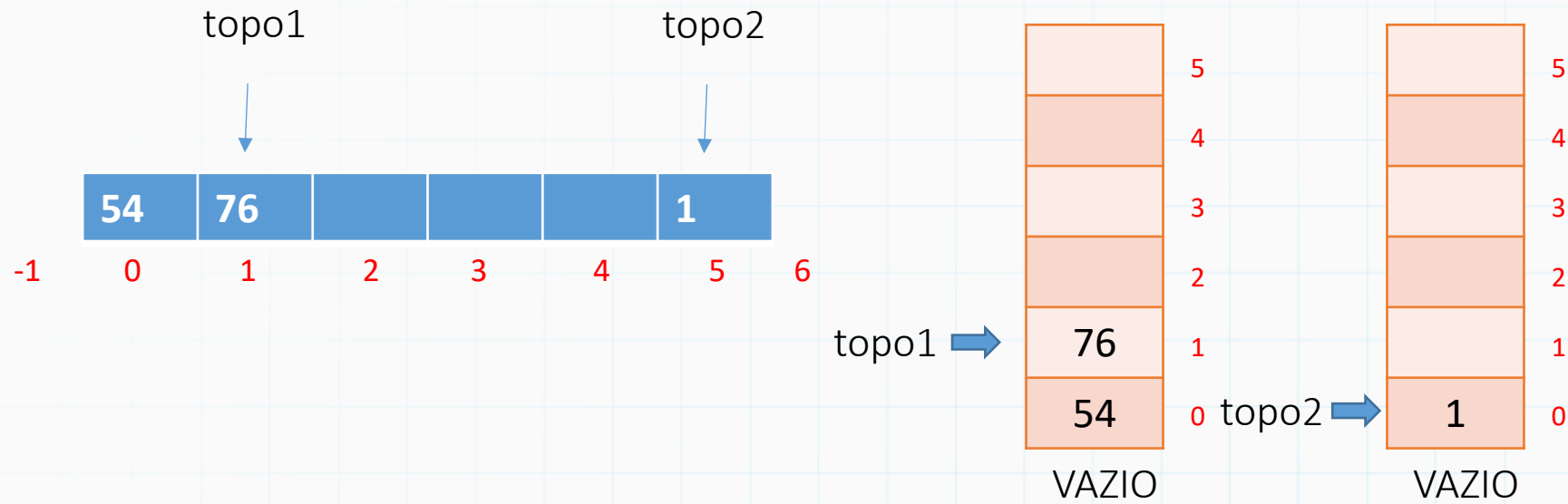
Exercício 1) Codifique a função abaixo de push em pilhas duplas, onde num=1 -> push na pilha 1, num=2 -> push na pilha 2 .

```
void push_pilhas(pilhas * P, int num, int el)
{
    if (P->topo1 == P->topo2-1)
        printf("pilhas cheias\n");
    else
    {
        if (num == 1)
        {
            P->topo1++;
            P->v[P->topo1] = el;
        }
        else
        {
            P->topo2--;
            P->v[P->topo2] = el;
        }
    }
}
```



Pilhas

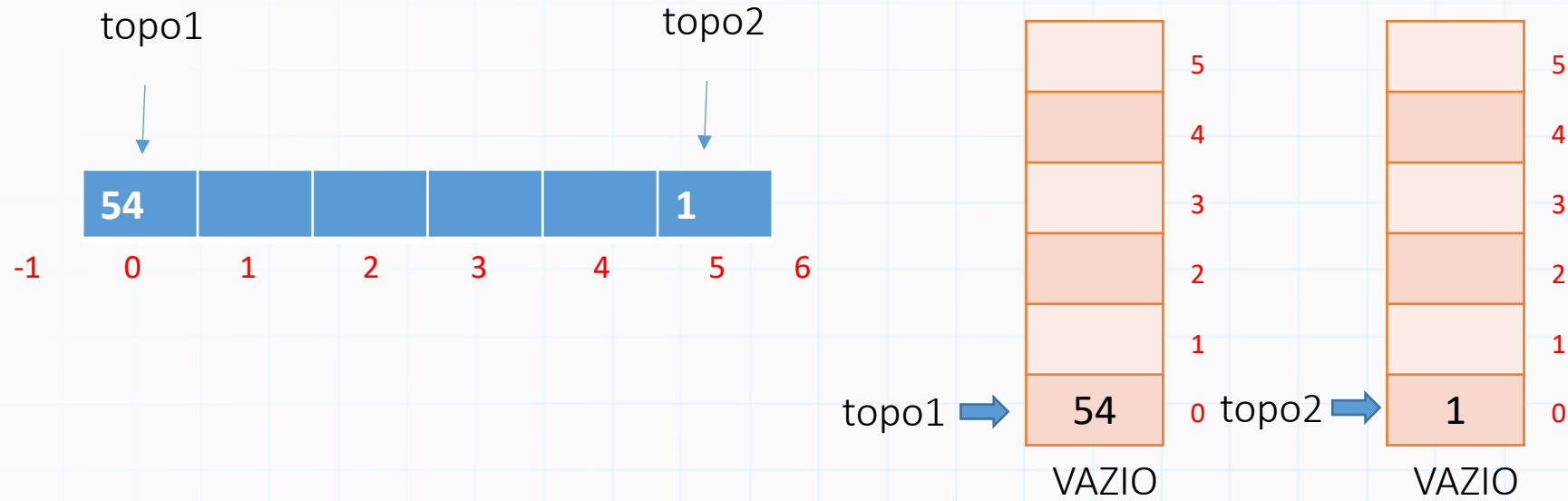
Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6, vamos ver o processo pop

Pilhas

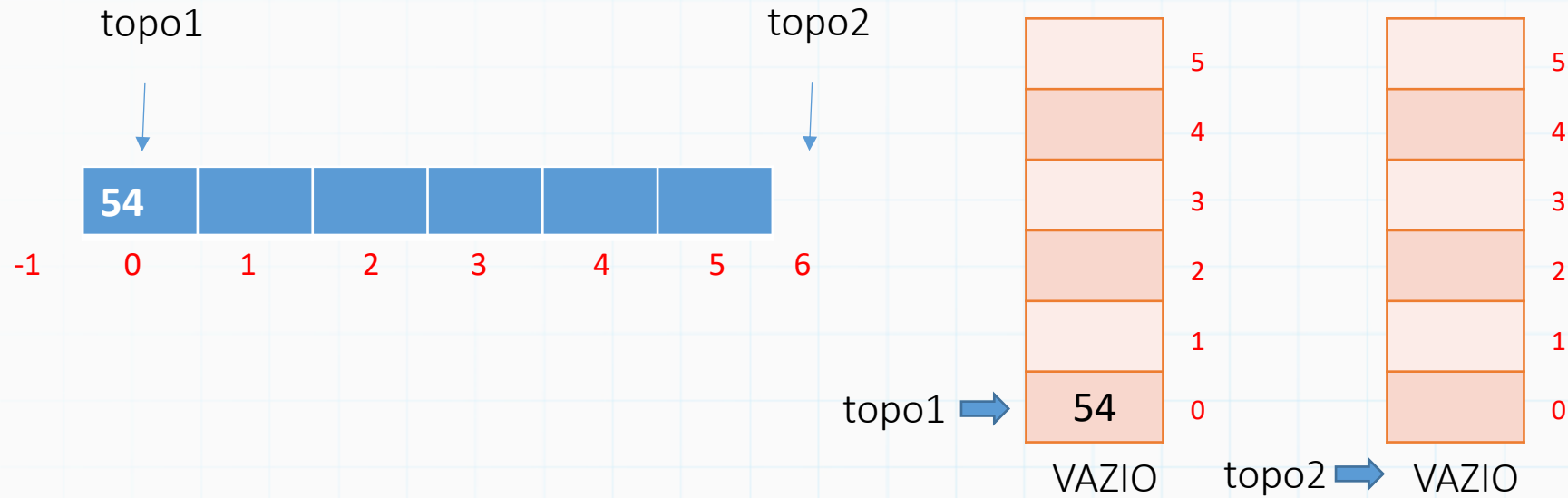
Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6, vamos ver o processo pop
- vamos fazer pop na pilha 1 -> POP(P, 1)

Pilhas

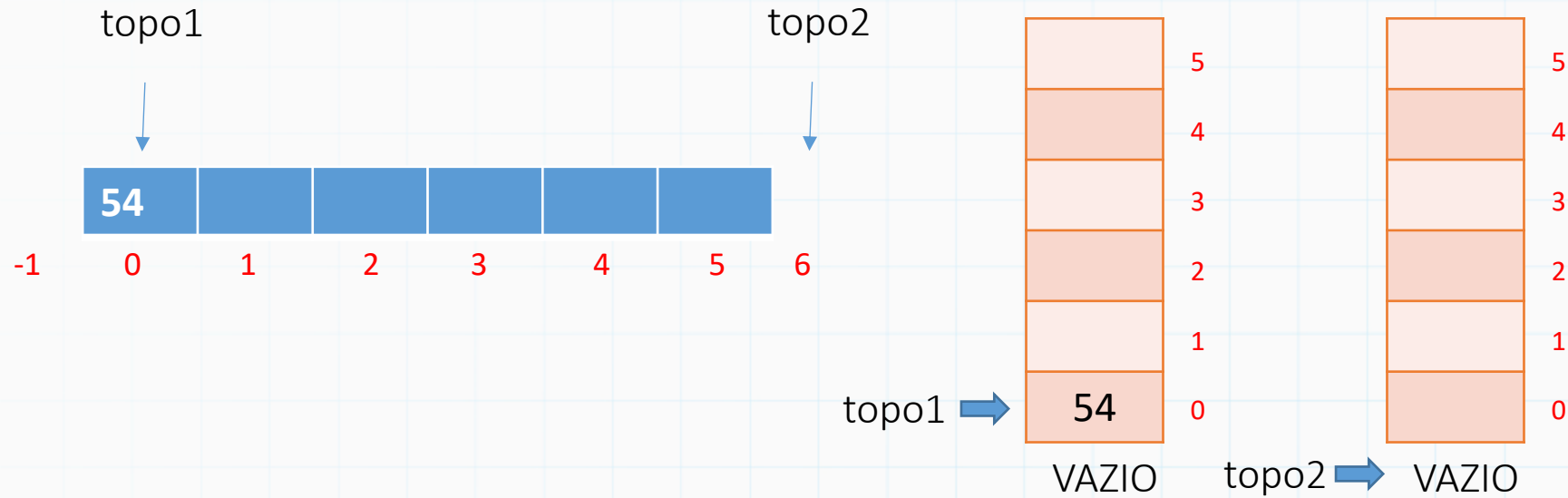
Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6, vamos ver o processo pop
- vamos fazer pop na pilha 1 -> $\text{POP}(P, 1)$
- vamos fazer pop na pilha 2 -> $\text{POP}(P, 2)$

Pilhas

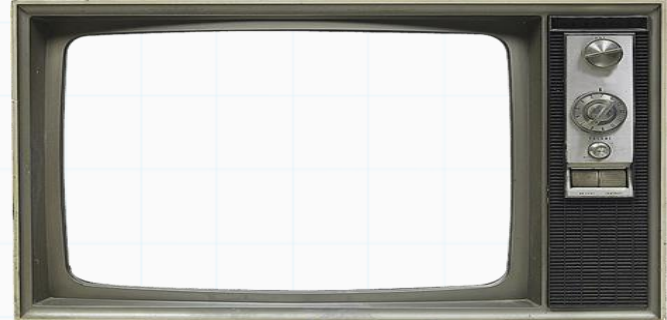
Pilhas duplas em vetores: Vamos agora implementar duas pilhas com apenas um vetor



- Veja esse exemplo que alocamos uma pilha dupla de tamanho 6, vamos ver o processo pop
- vamos fazer pop na pilha 1 -> $\text{POP}(P, 1)$
- vamos fazer pop na pilha 2 -> $\text{POP}(P, 2)$

A função pop em pilhas duplas é descrita como :

Pilhas



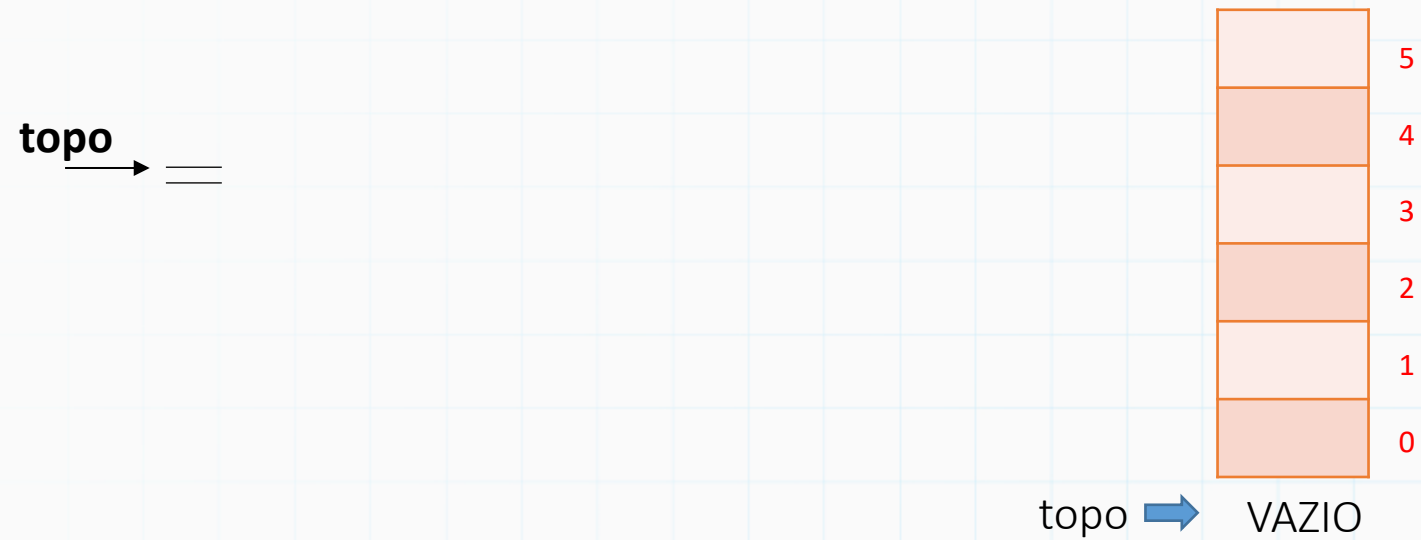
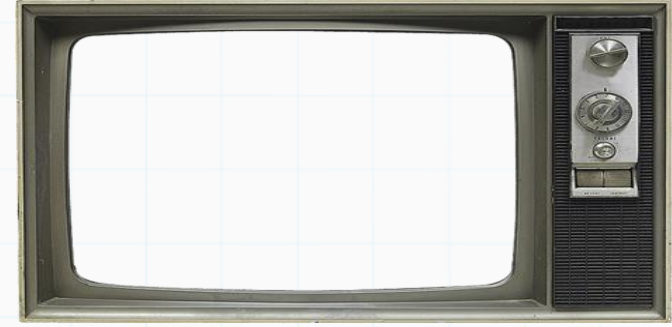
```
int pop_pilhas(pilhas * P, int num)
{
    int el;

    if (num == 1)
    {
        if (P->topo1 >= 0)
        {
            el = P->v[P->topo1];
            P->topo1--;
        }
        else
            printf("pilha 1 vazia\n");
    }
    else
    {
        if (P->topo2 < P->tam)
        {
            el = P->v[P->topo2];
            P->topo2++;
        }
        else
            printf("pilha 2 vazia\n");
    }
    return el;
}
```



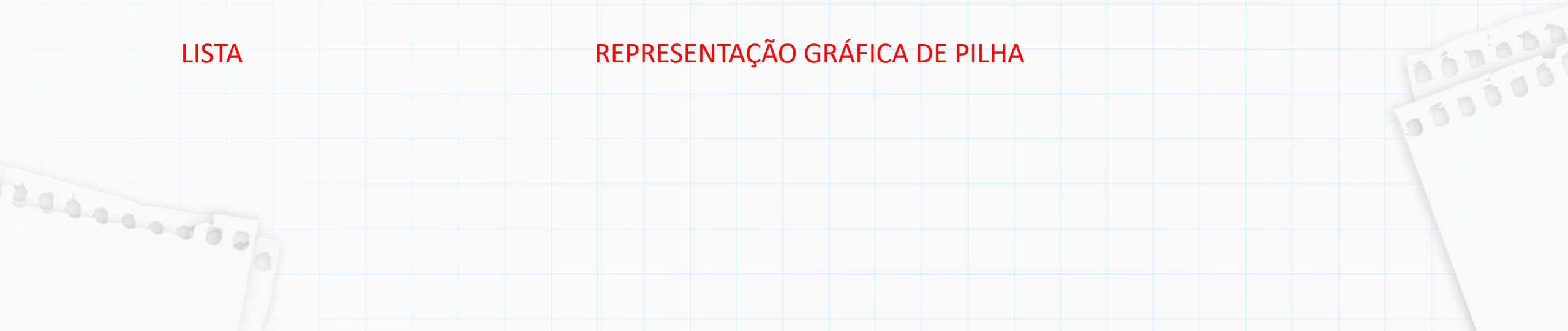
Pilhas

- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada



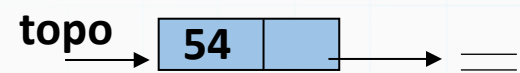
LISTA

REPRESENTAÇÃO GRÁFICA DE PILHA



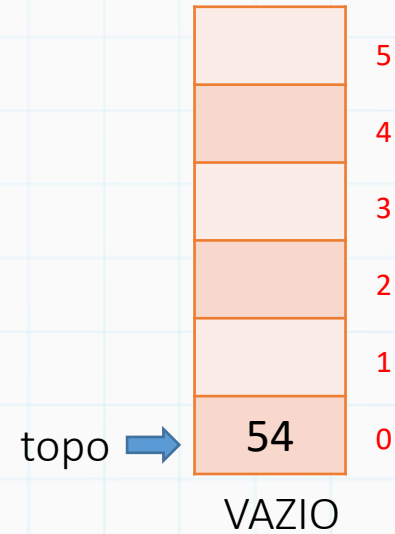
Pilhas

- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54 -> PUSH(P,54)

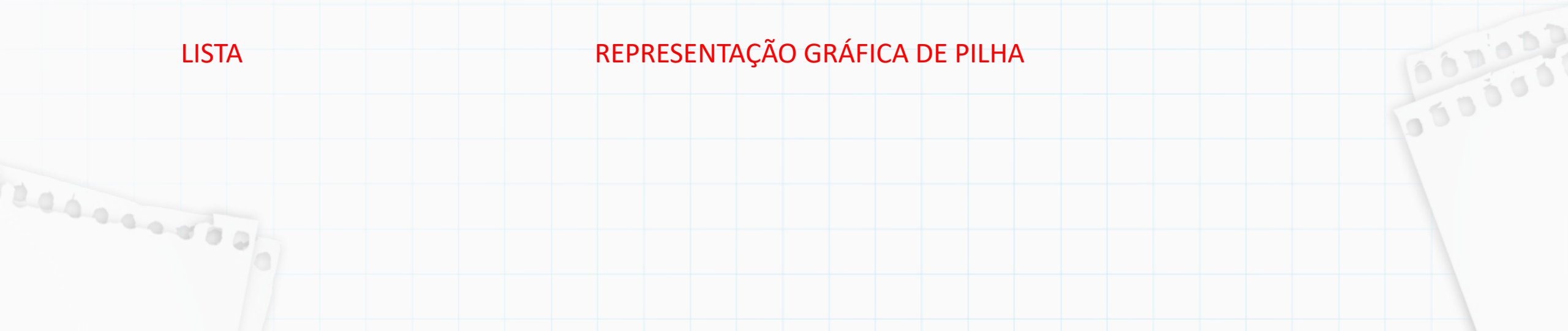
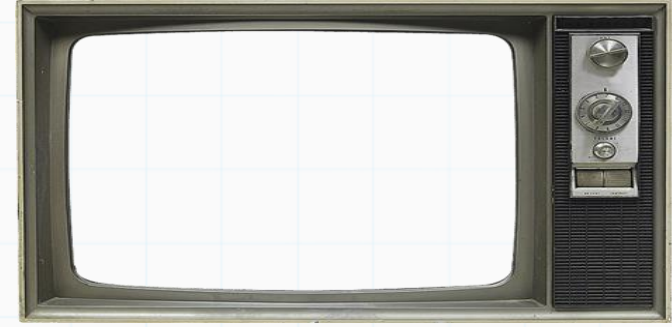


inserções e deleções no
começo da lista

LISTA

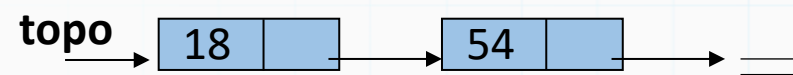


REPRESENTAÇÃO GRÁFICA DE PILHA



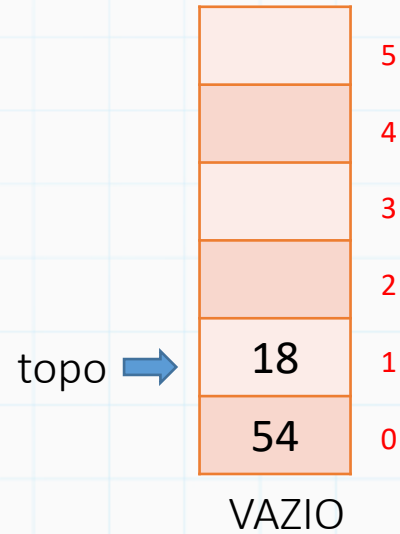
Pilhas

- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 -> PUSH(P,54), PUSH(P,18)

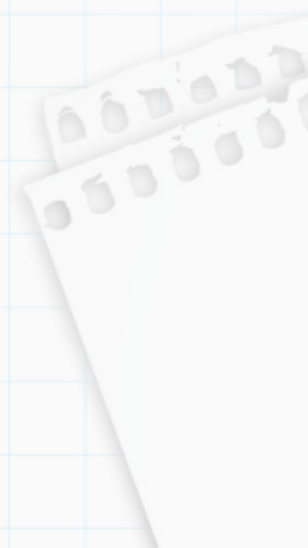
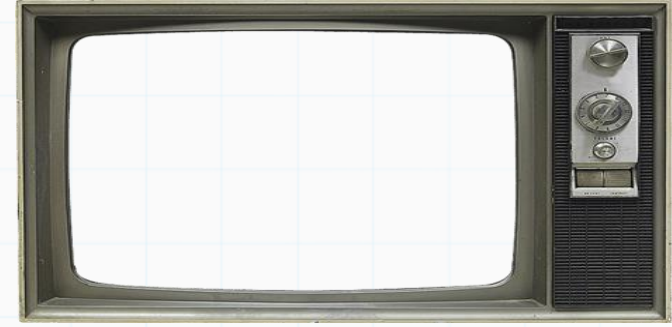


inserções e deleções no
começo da lista

LISTA

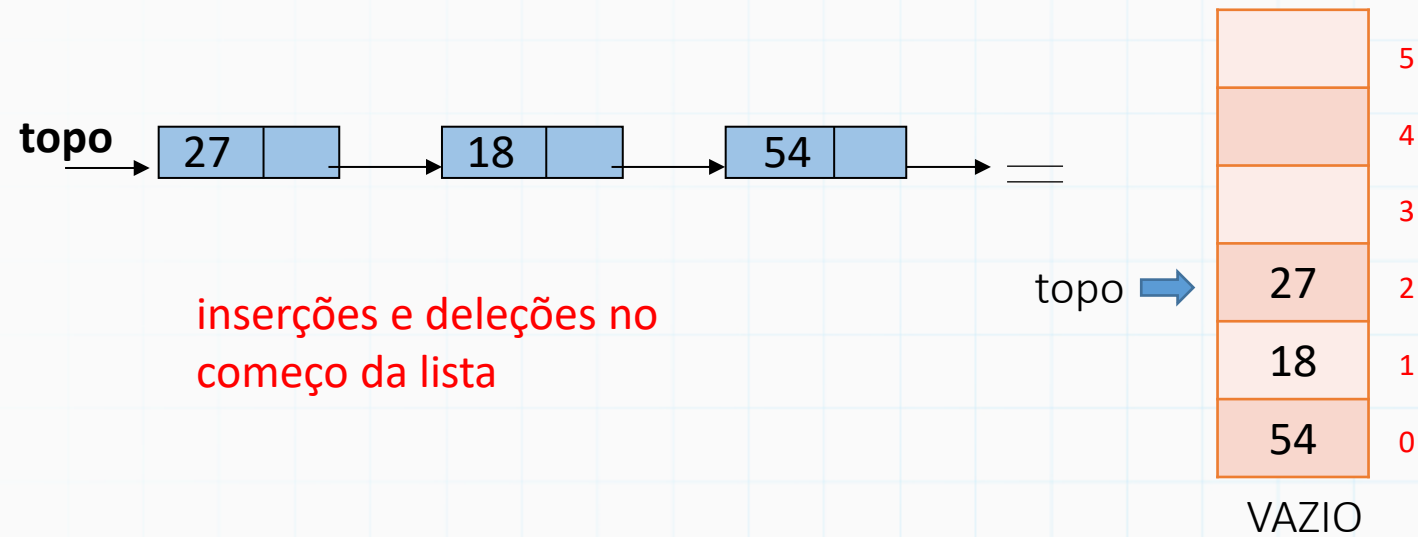


REPRESENTAÇÃO GRÁFICA DE PILHA



Pilhas

- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)

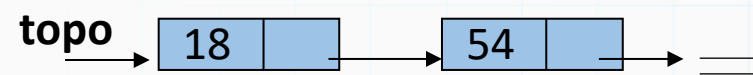


LISTA

REPRESENTAÇÃO GRÁFICA DE PILHA

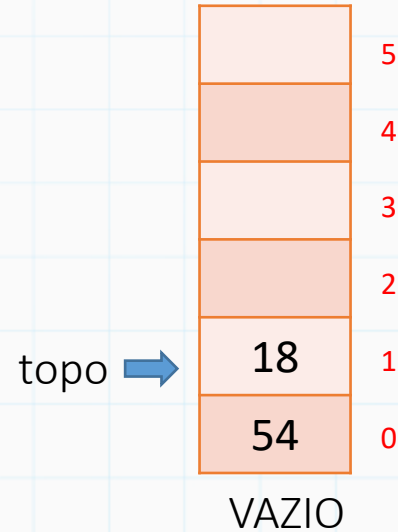
Pilhas

- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)

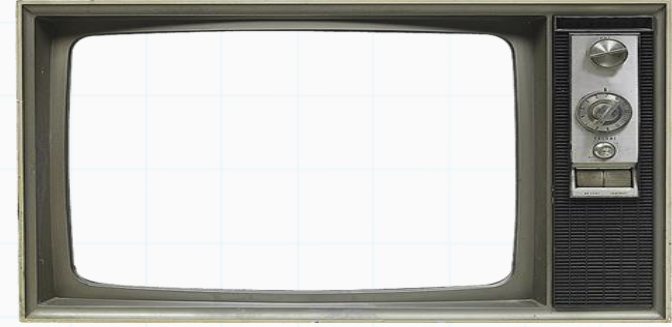


inserções e deleções no
começo da lista

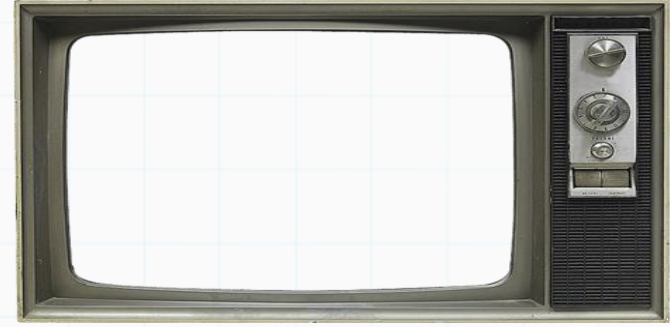
LISTA



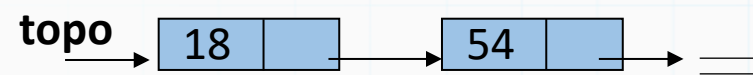
REPRESENTAÇÃO GRÁFICA DE PILHA



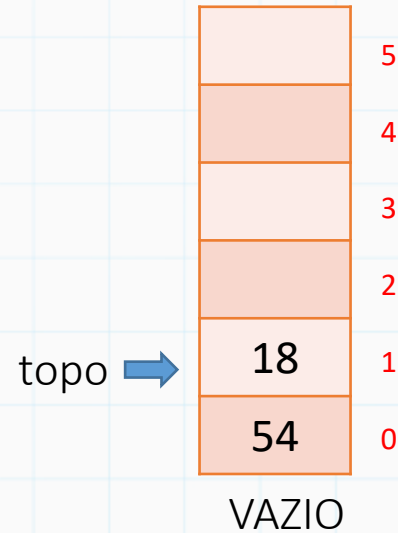
Pilhas



- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)



inserções e deleções no
começo da lista



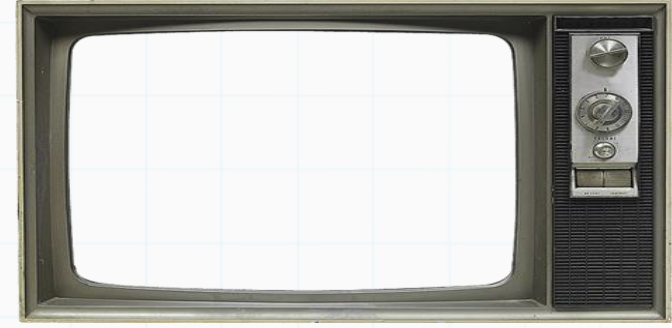
```
pilha * aloca_no(void)
{
    pilha *aux;
    aux      = (pilha *) malloc (sizeof(pilha));
    aux->prox = NULL;
    return aux;
}
```

```
struct NO
{
    int info;
    struct NO *prox;
};
typedef struct NO pilha;
```

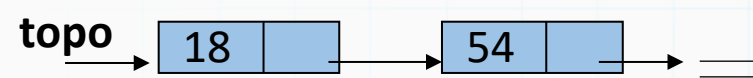
```
int main()
{
    pilha *p = NULL;

    return 0;
}
```

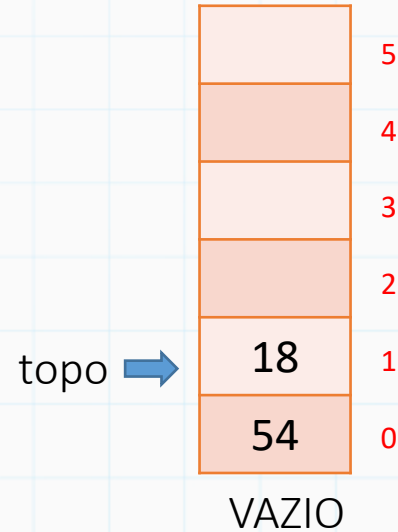
Pilhas



- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)



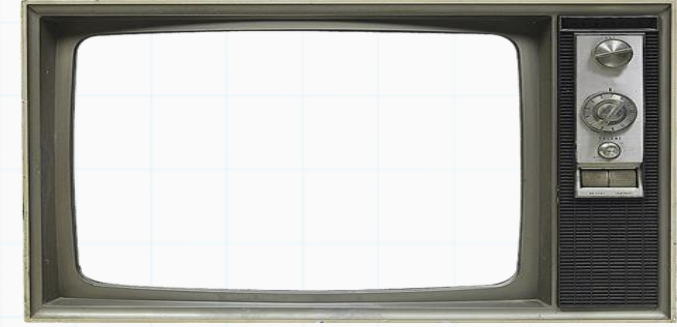
inserções e deleções no
começo da lista



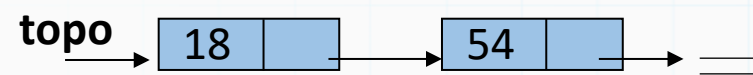
```
int vazia_pilha(pilha* p)
{
    if (p == NULL)
        return 1;
    else
        return 0;
}
```

```
int cheia_pilha(pilha* p)
{
    printf("tu eh maneh, pilha encadeada não fica cheia\n");
}
```

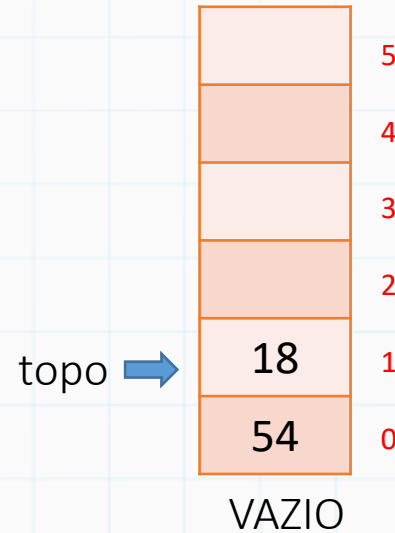
Pilhas



- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)



inserções e deleções no
começo da lista

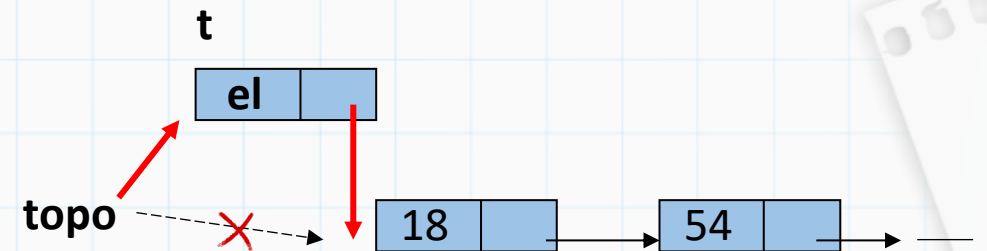


```
ilha* push_pilha(ilha* topo, int el)
{
    ilha *t;
    t = aloca_no();

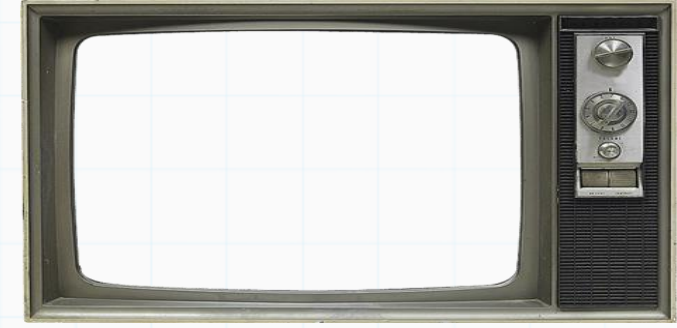
    t->info = el;
    t->prox = topo;
    topo    = t;

    return topo;
}
```

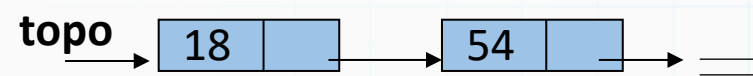
```
int main()
{
    ilha *topo = NULL;
    topo = push_pilha(topo, 28);
    ...
}
```



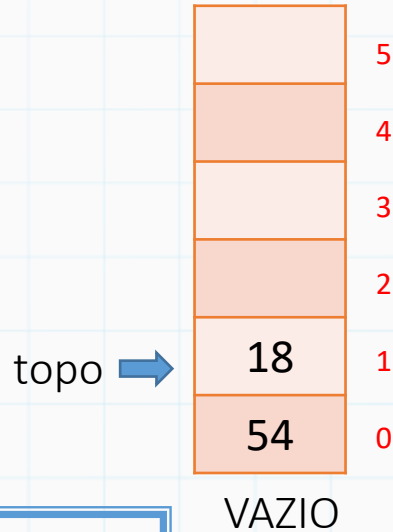
Pilhas



- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)



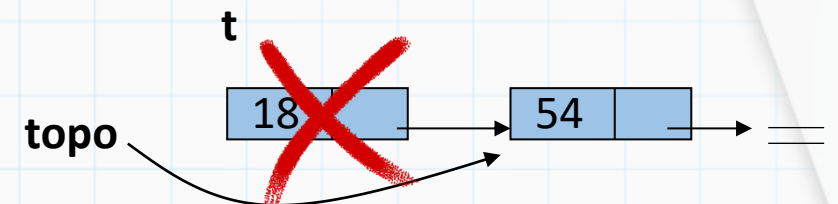
inserções e deleções no
começo da lista



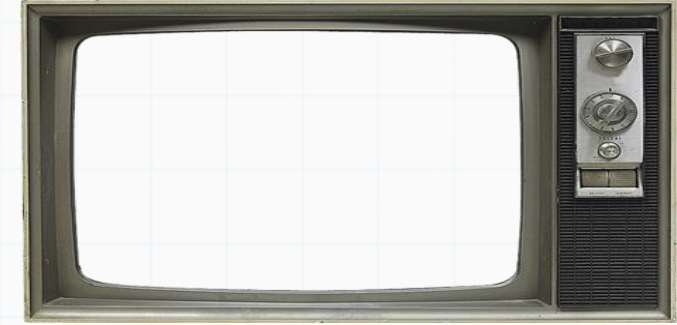
```
pilha *topo = NULL;  
topo = push_pilha(topo, 28);  
topo = push_pilha(topo, 45);  
topo = push_pilha(topo, 9);
```

```
int el;  
topo = pop_pilha(topo, &el);  
printf("topo = %d\n", topo);  
...
```

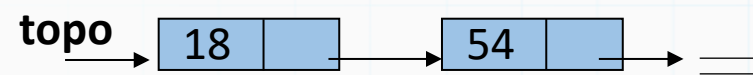
```
pilha* pop_pilha(pilha* topo, int *el)  
{  
    pilha *t;  
  
    if(topo == NULL)  
        printf("pilha vazia\n");  
    else  
    {  
        t = topo;  
        *el = topo->info;  
        topo = topo->prox;  
        free(t);  
    }  
    return topo;  
}
```



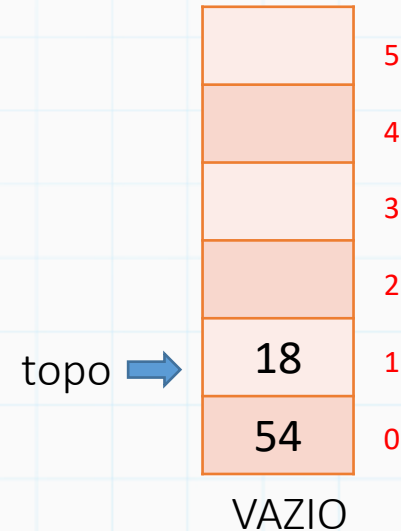
Pilhas



- Pilhas em listas encadeadas:
 - Usaremos uma lista simplesmente encadeada
 - Vamos empilhar elemento 54, 18 e 27 -> PUSH(P,54), PUSH(P,18) e PUSH(P,27)
 - Vamos retirar o topo da pilha -> POP(P)



inserções e deleções no
começo da lista

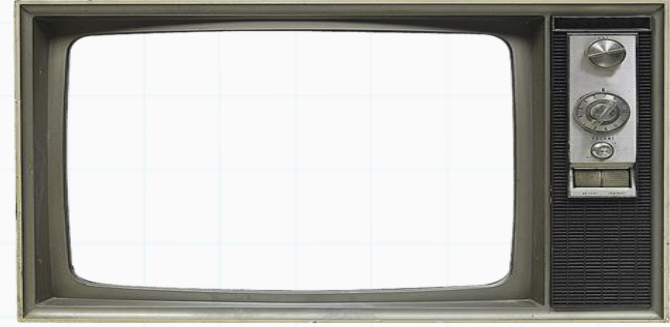


```
int top_pilha(pilha* topo)
{
    if(topo == NULL)
        return -1;
    else
        return topo->info;
}
```

```
pilha *topo = NULL;
topo = push_pilha(topo, 28);
topo = push_pilha(topo, 45);
topo = push_pilha(topo, 9);
printf("topo = %d\n", top_pilha(topo));
...
```

Pilhas

- Exercício 3) Um usuário inseriu numa pilha os números 1, 2, 3, 4 e 5 nessa ordem usando a operação **push**, porem ele pode (ou não) ter realizado operações **pop** enquanto estava empilhando os números (**sempre que uma operação pop é realizada, o número é impresso**):

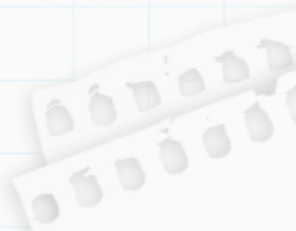
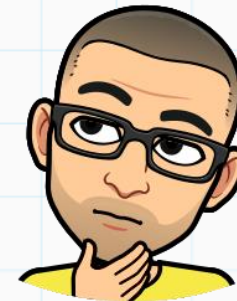


Ex: **push(1), push(2), push(3), pop(), push(4), push(5), pop() pop()**

Nessa caso seria impresso 3, 5 e 4

Quais das seguintes impressões podem ter sido geradas com a sequencia de push 1, 2, 3, 4 e 5 (alternando ou não com pop), e quais seriam as sequencias de push e pop:

- a) 3, 4, 5, 1, e 2
- b) 3, 4, 5, 2, e 1
- c) 1, 5, 2, 3 e 4
- d) 5, 4, 3, 1 e 2
- e) 5, 4, 3, 2 e 1



Pilhas

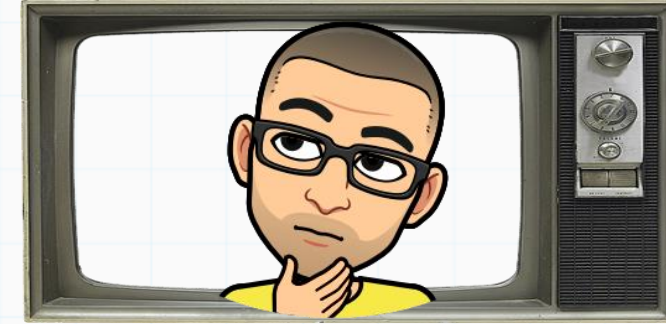
Exercício 4) Dado uma pilha p, escreva uma função **recursiva** chamada **push_bottom** que empilha um determinado elemento **el** e empurra ele direto para o fundo da pilha.

```
void push_bottom_pilha(pilha * P, int el)
```

A função deve apenas interagir com a pilha com as funções de pilha (**push**, **pop** e **top**)

```
void push_pilha(pilha * P, int el)
int pop_pilha(pilha * P)
int top_pilha(pilha * P)
int vazia_pilha(pilha * P)
```

Dica: a cada chamada recursiva, a pilha retira um elemento e entra na recursão, mas não esqueça de recolocar o elemento quando voltar da recursão.



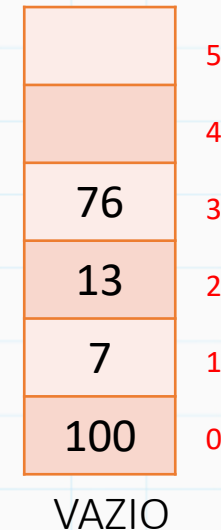
Não podemos acessar/alterar elementos que não seja através dessas funções

```
struct PILHA
{
    int tam;
    int topo;
    int *v;
};
typedef struct PILHA pilha;
```

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

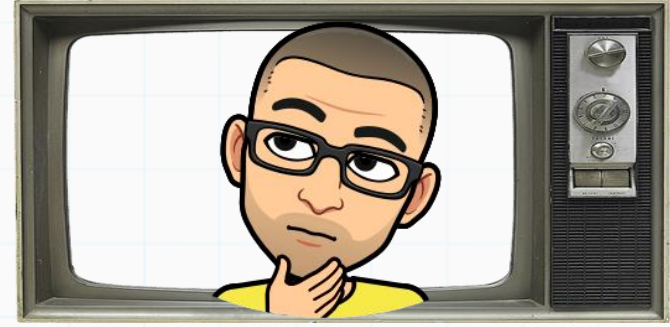
    push_pilha(&P, 7);
    push_pilha(&P, 13);
    push_pilha(&P, 76);
    push_bottom_pilha(&P, 100);
    imprime_pilha(&P);
}
```

topo →



Pilhas

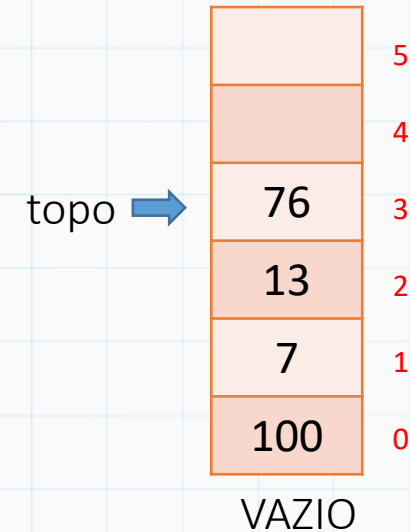
Exercício 4) Dado uma pilha p, escreva uma função **recursiva** chamada **push_bottom** que empilha um determinado elemento **el** e empurra ele direto para o fundo da pilha.



```
void push_bottom_pilha(pilha * P, int el)
{
    if (vazia_pilha(P) == 1)
        push_pilha(P, el);
    else
    {
        int temp = pop_pilha(P);

        push_bottom_pilha(P, el);

        push_pilha(P, temp);
    }
}
```



Pilhas

Exercício 5) Dado uma pilha p, escreva uma função **recursiva** chamada **inverte_pilha** que inverte os elementos de uma pilha.

```
void inverte_pilha(pilha* P)
```

A função deve apenas interagir com a pilha com as funções de pilha (**push**, **pop** e **top**) e a função nova **push_bottom**

```
void push_pilha(pilha * P, int el)
```

```
int pop_pilha(pilha * P)
```

```
int top_pilha(pilha * P)
```

```
int vazia_pilha(pilha * P)
```

```
void push_bottom_pilha(pilha * P, int el)
```

Vamos usar

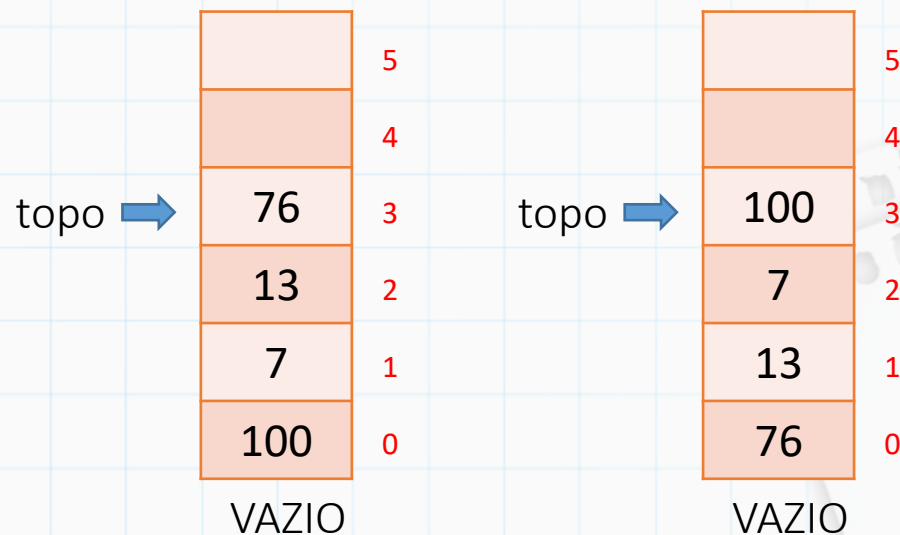
Dica: 1) Retira, 2) Recursão,
3) Recoloca no Fundo.

```
struct PILHA
{
    int tam;
    int topo;
    int *v;
};
typedef struct PILHA pilha;
```

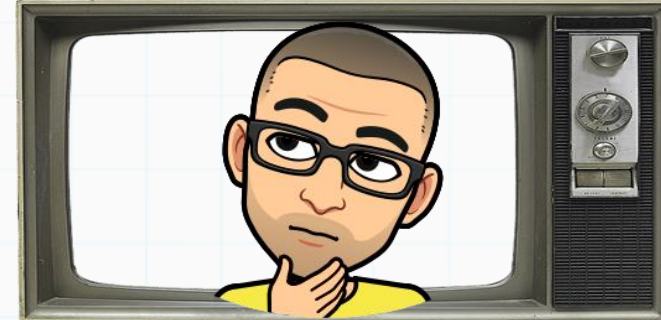
```
int main()
{
    pilha P;
    cria_pilha(&P, 10);

    push_pilha(&P, 7);
    push_pilha(&P, 13);
    push_pilha(&P, 76);
    push_bottom_pilha(&P, 100);
    imprime_pilha(&P);

    inverte_pilha(&P);
    imprime_pilha(&P);
}
```

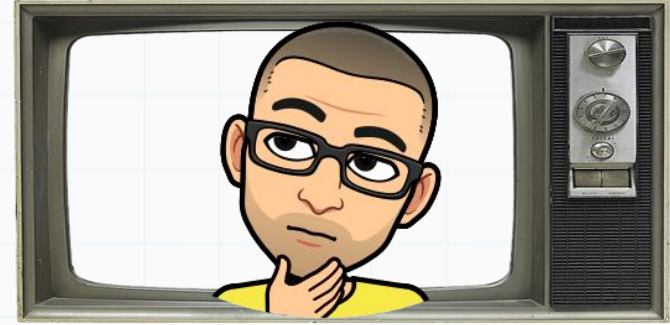


Não podemos acessar/alterar elementos que não seja através dessas funções



Pilhas

Exercício 5) Dado uma pilha p, escreva uma função **recursiva** chamada **inverte_pilha** que inverte os elementos de uma pilha.

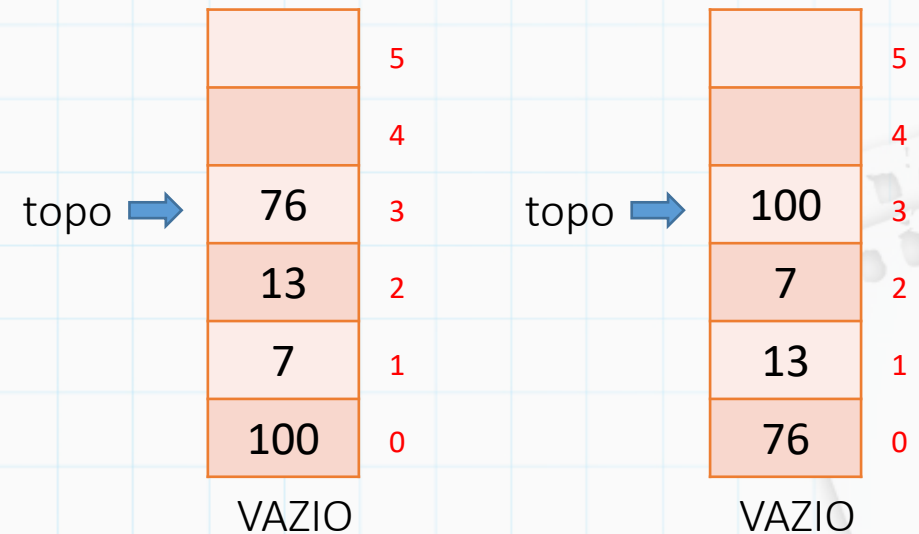


```
void inverte_pilha(pilha* P)
{
    int temp;

    if (vazia_pilha(P) == 1)
        return;
    else
    {
        int temp = pop_pilha(P);

        inverte_pilha(P);

        push_bottom_pilha(P, temp);
    }
}
```



Pilhas

Exercício 6) Dado uma pilha p, escreva uma função `remove_el_pilha` que dado um elemento el, remove da pilha esse elemento.

```
void remove_el_pilha(pilha* P, int el)
```

A função deve apenas interagir com a pilha com as funções de pilha (`push`, `pop` e `top`)

```
void push_pilha(pilha * P, int el)
int pop_pilha(pilha * P)
int top_pilha(pilha * P)
int vazia_pilha(pilha * P)
```

MAS PODEMOS CRIAR
E USAR UMA PILHA
AUXILIAR DE DENTRO
DA FUNÇÃO.

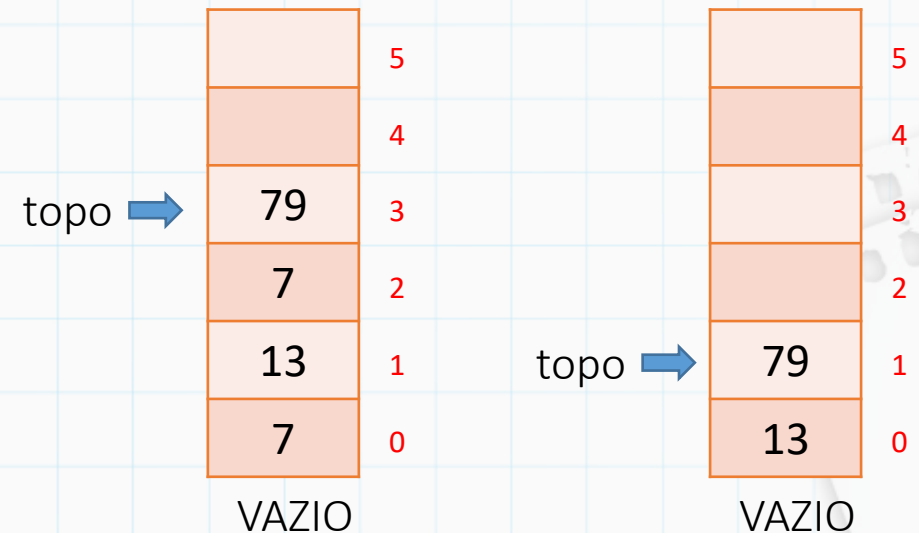
DICA: Jogue elementos
para a pilha auxiliar,
depois jogue de volta.

```
struct PILHA
{
    int tam;
    int topo;
    int *v;
};
typedef struct PILHA pilha;
```

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

    push_pilha(&P, 7);
    push_pilha(&P, 13);
    push_pilha(&P, 7);
    push_pilha(&P, 79);
    imprime_pilha(&P);

    remove_el_pilha(&P, 7);
    imprime_pilha(&P);
}
```



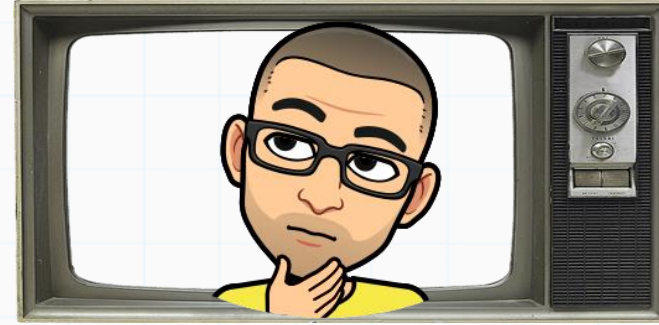
Pilhas

Exercício 6) Dado uma pilha p, escreva uma função `remove_el_pilha` que dado um elemento el, remove da pilha esse elemento

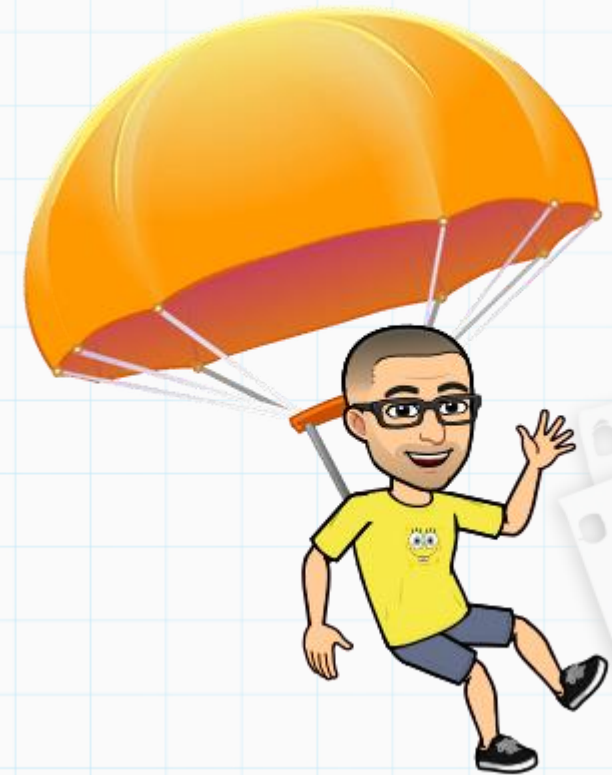
```
void remove_el_pilha(pilha* P, int el)
{
    pilha P_aux;
    cria_pilha(&P_aux, P->topo+1);

    while (vazia_pilha(P) == 0)
    {
        int item = pop_pilha(P);
        if (item != el)
            push_pilha(&P_aux, item);
    }

    while (vazia_pilha(&P_aux) == 0)
    {
        int item = pop_pilha(&P_aux);
        push_pilha(P, item);
    }
}
```



Até a próxima



Slides baseados no curso de Aline Nascimento