



Fim

Início



Correia transportadora



Pilha e Fila estática

Prof. Sérgio Souza Costa

Sobre mim



Sérgio Souza Costa

Professor - UFMA

Doutor em Computação Aplicada (INPE)



prof.sergio.costa@gmail.com



<https://sites.google.com/site/profsergiocosta/home>



<http://www.slideshare.net/skosta/presentations?order=popular>



<https://twitter.com/profsergiocosta>

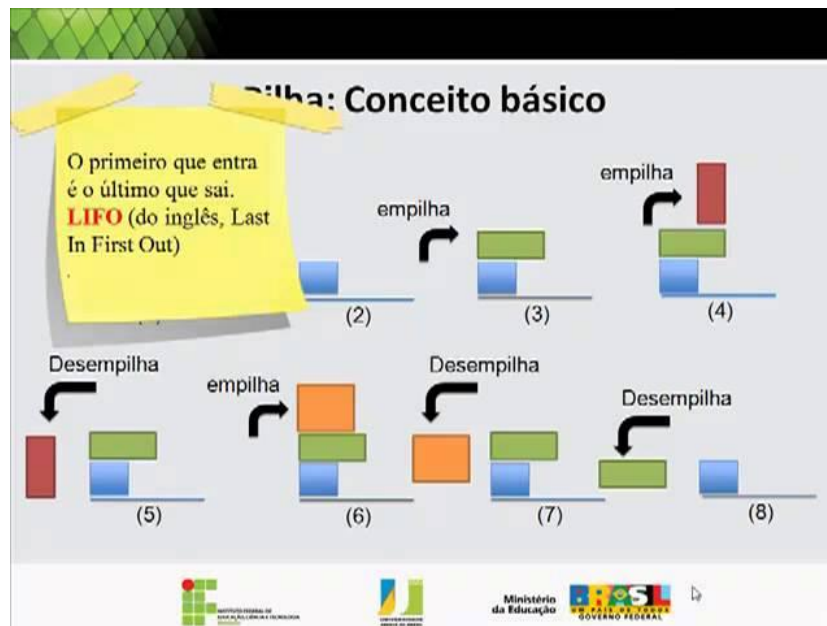


<http://gplus.to/sergiosouzacosta>

Pilhas

Um dos conceitos mais úteis na ciência da computação é o de pilha. Nesta aula, examinaremos essa simples estrutura de dados e verificaremos por que ela desempenha esse proeminente papel nas áreas de programação e de linguagens de programação.

Esta aula pode ser vista também no vídeo abaixo:





© Can Stock Photo - csp6591766

Pilhas



Quais operações você pode fazer com pilhas ?



Quais operações você pode fazer
com pilhas ?

Empilhar



Quais operações você pode fazer
com pilhas ?

Empilhar

No topo



Quais operações você pode fazer
com pilhas ?

Empilhar

No topo

Desempilhar



Quais operações você pode fazer
com pilhas ?

Empilhar

No topo

Desempilhar

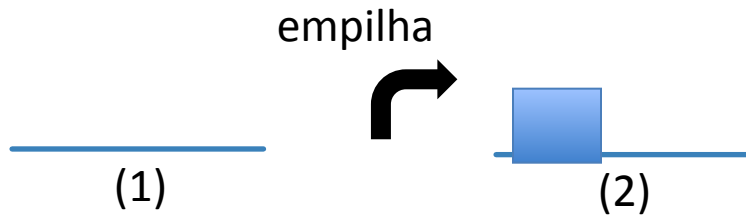
Do topo



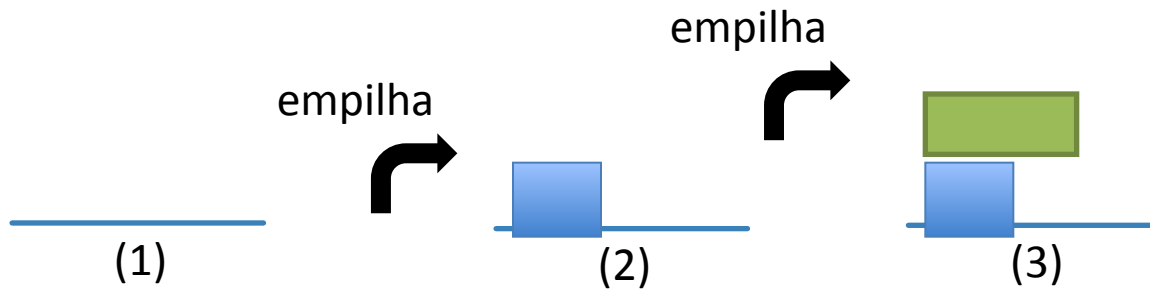
Pilha: Conceito básico

(1)

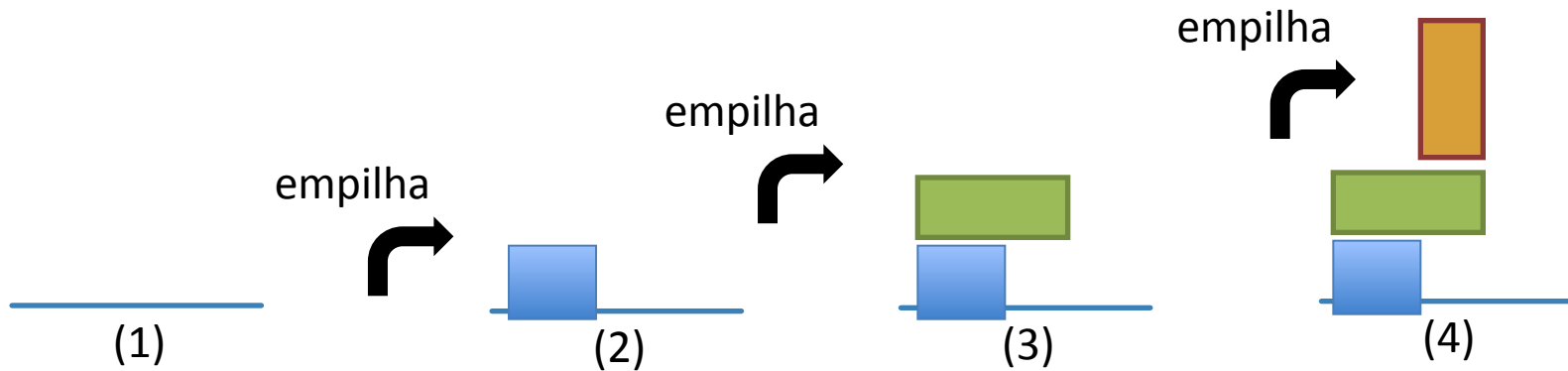
Pilha: Conceito básico



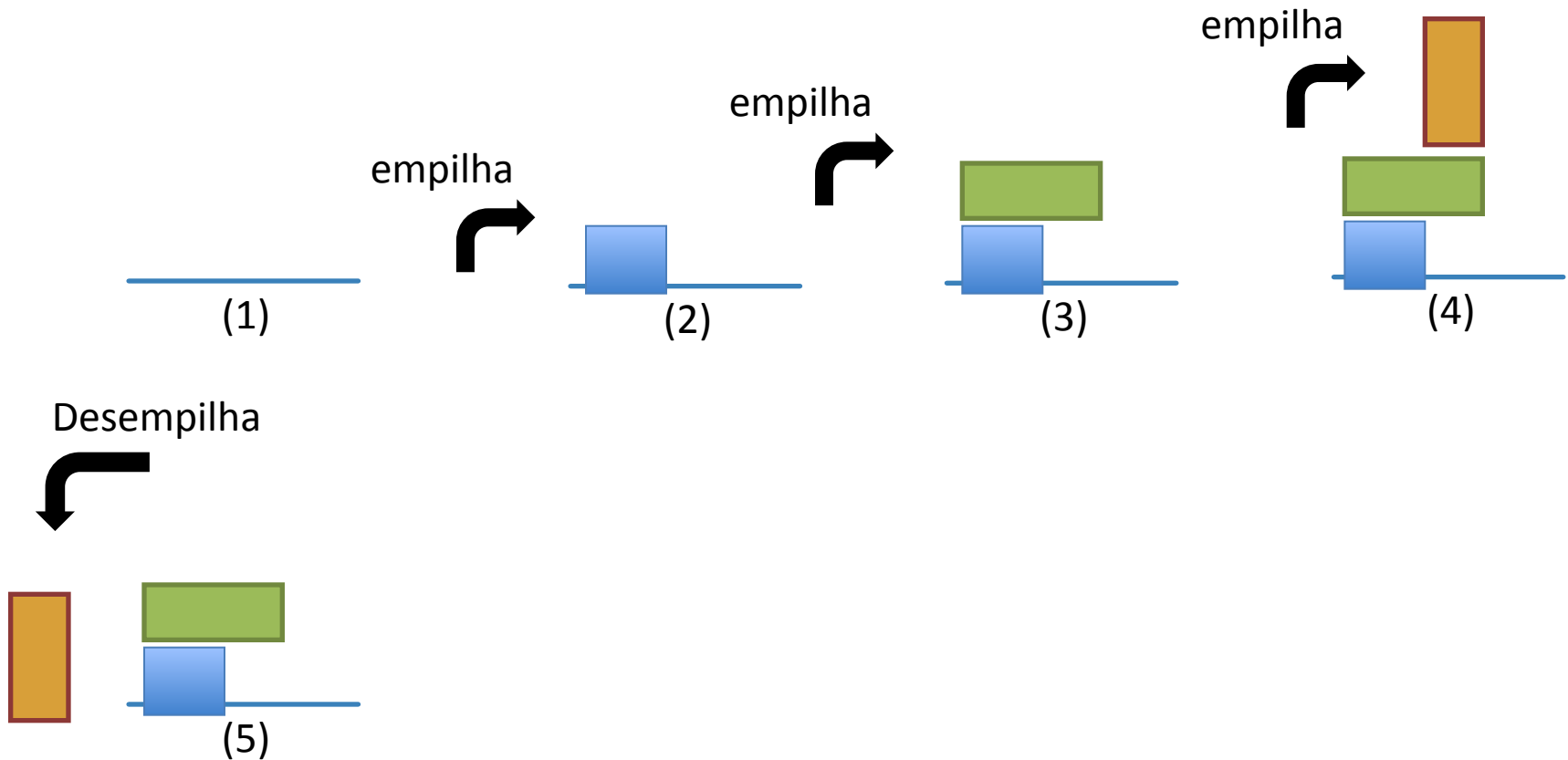
Pilha: Conceito básico



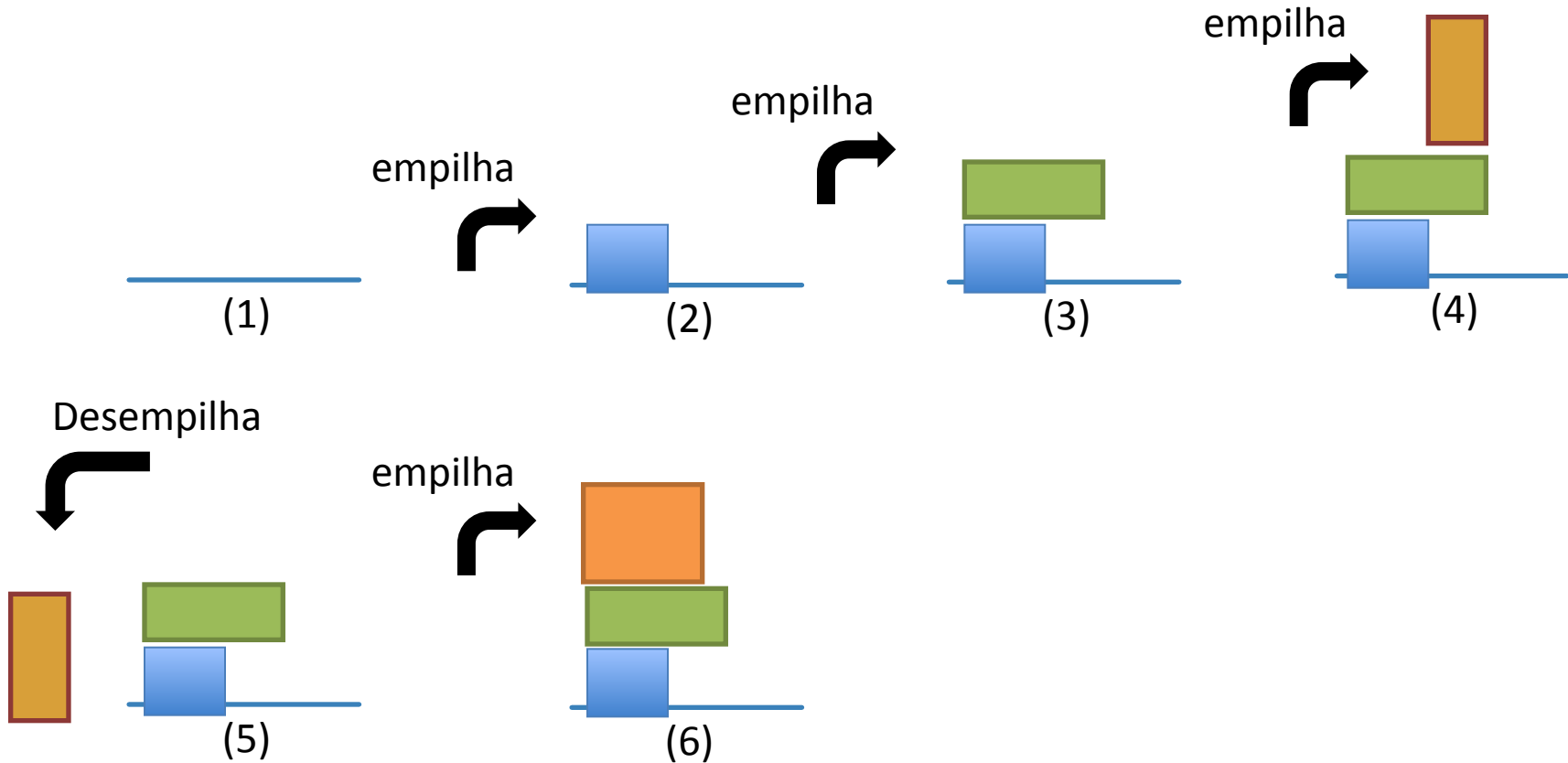
Pilha: Conceito básico



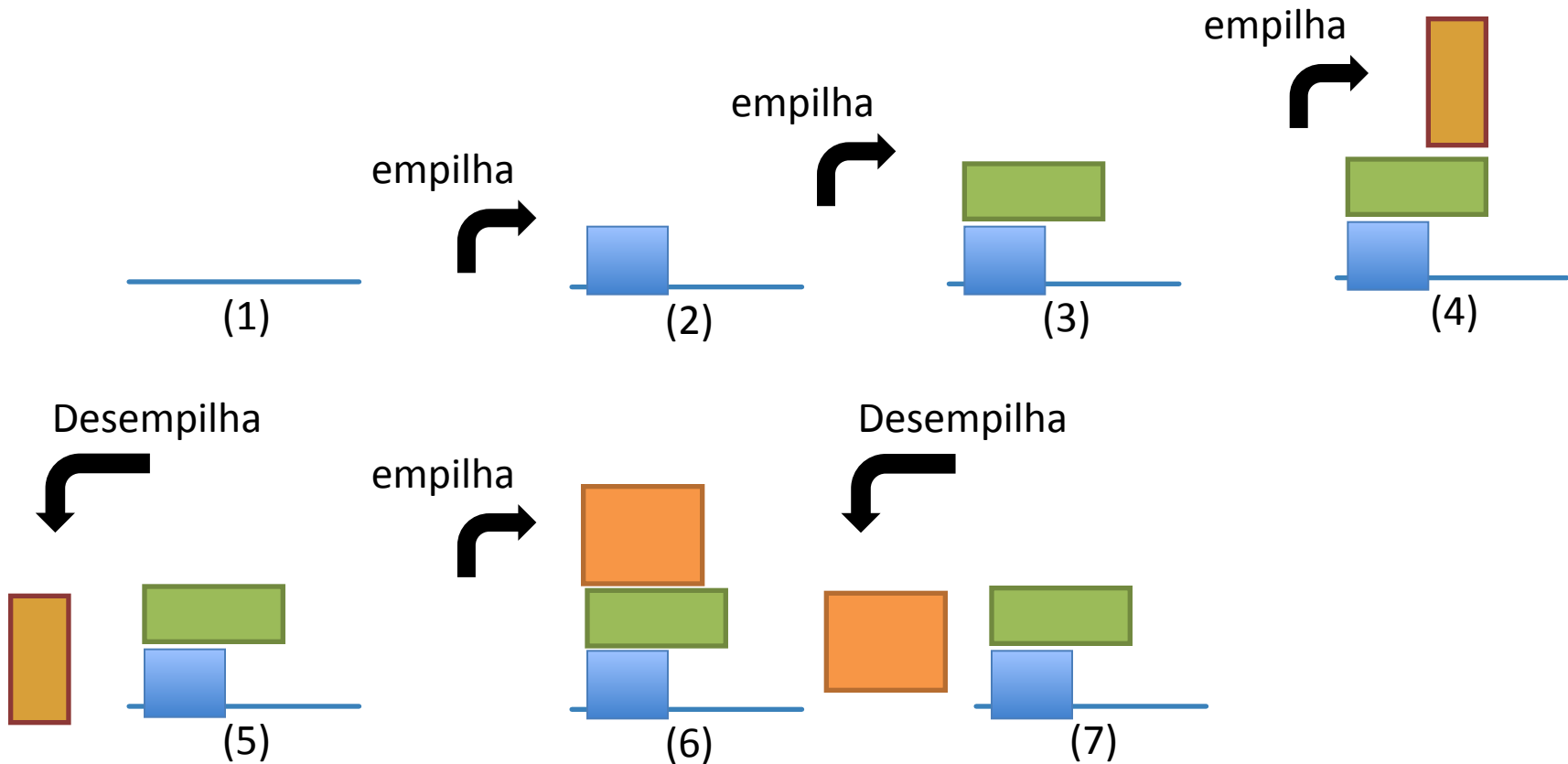
Pilha: Conceito básico



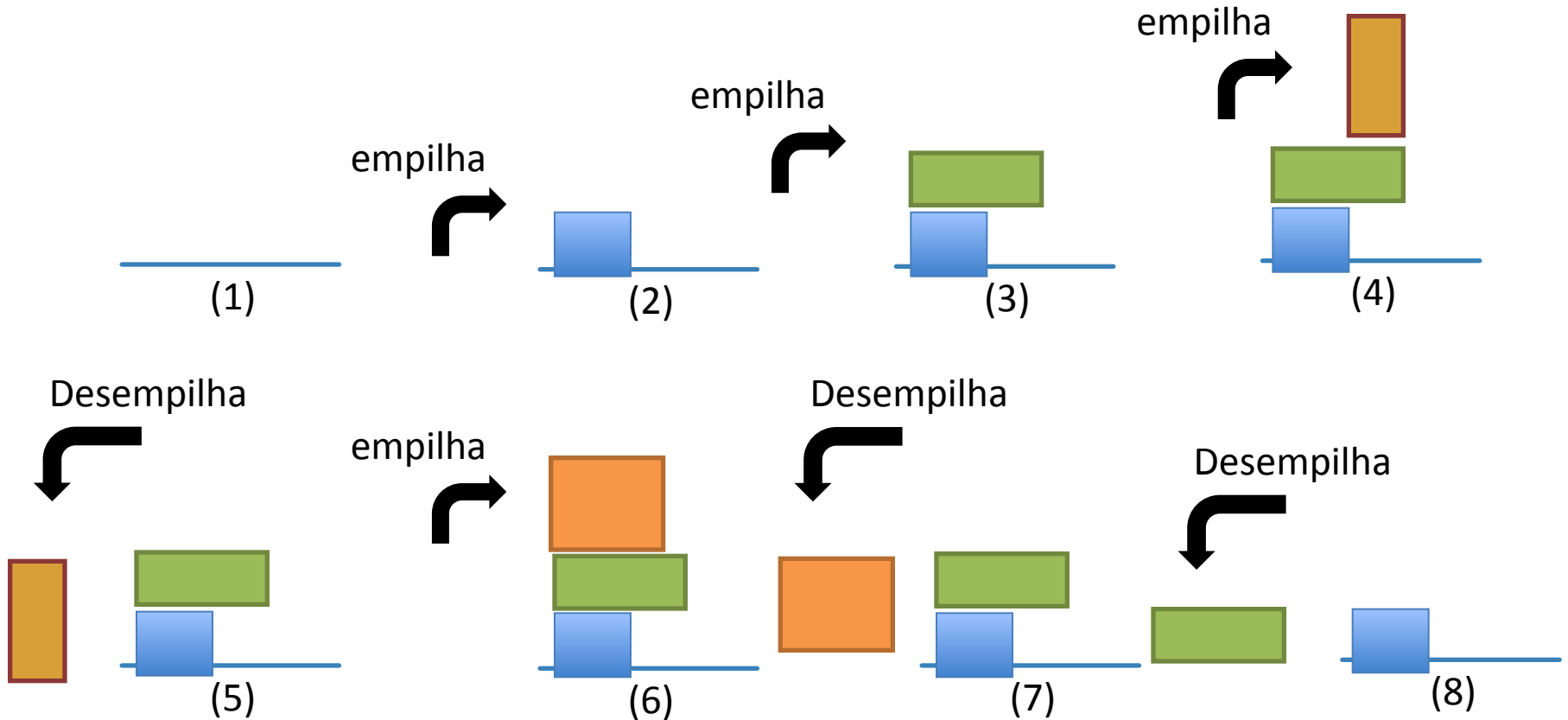
Pilha: Conceito básico



Pilha: Conceito básico

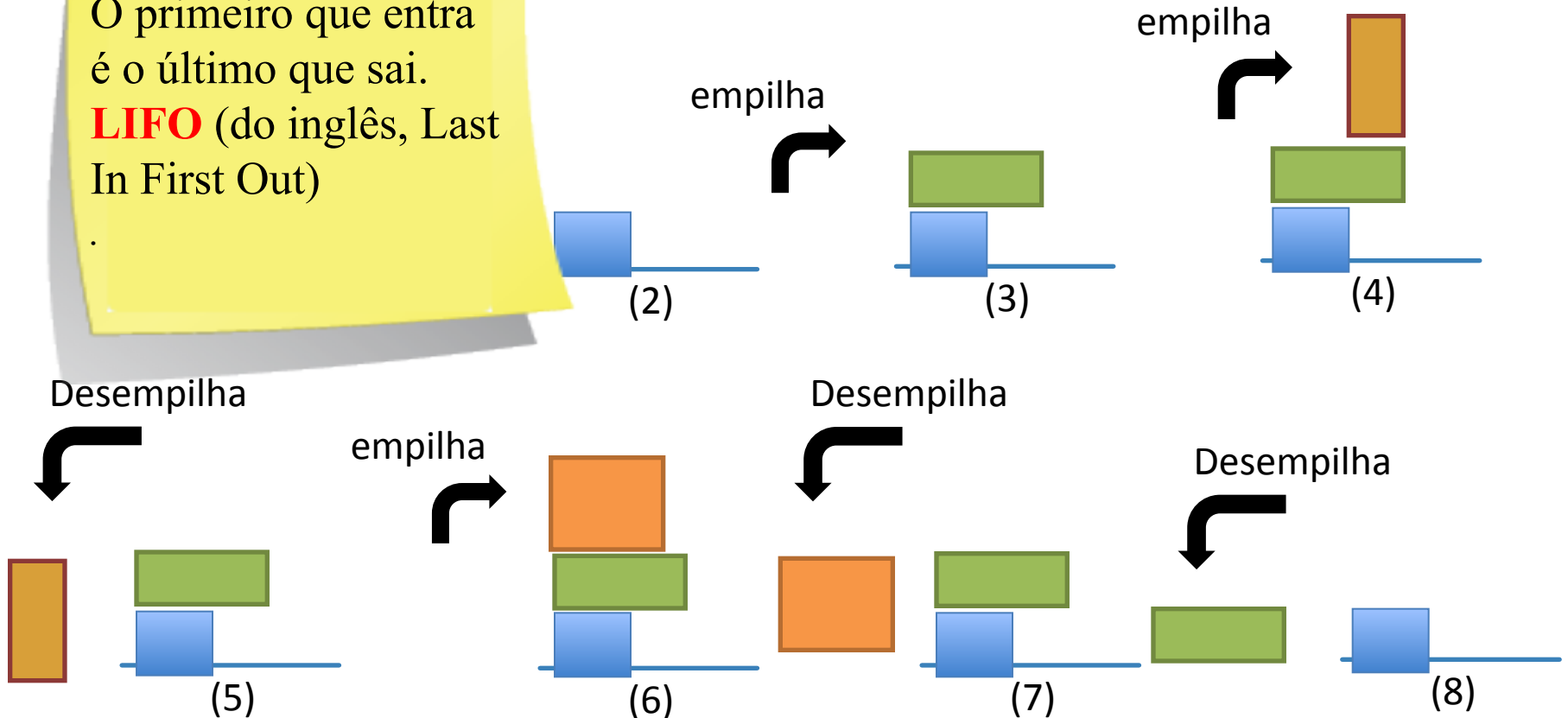


Pilha: Conceito básico



Empilha: Conceito básico

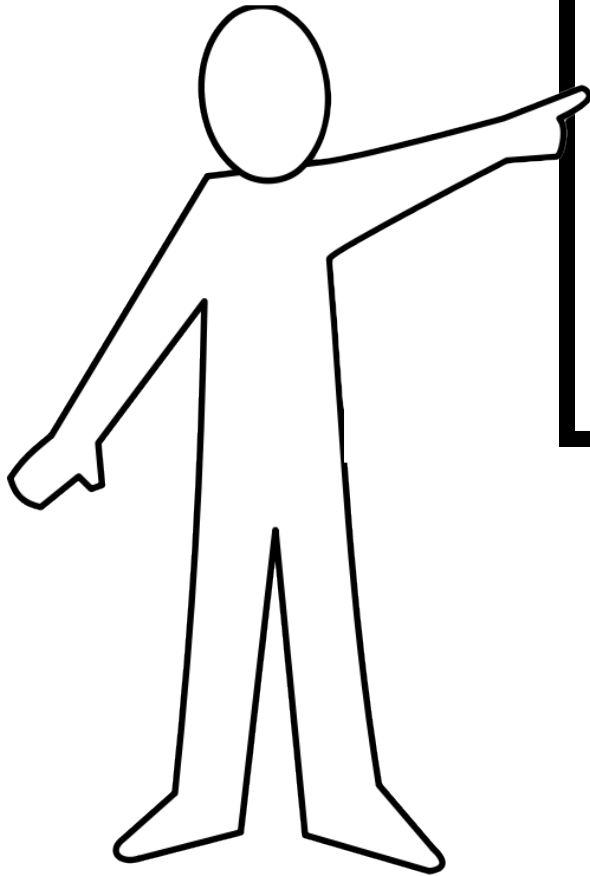
O primeiro que entra
é o último que sai.
LIFO (do inglês, Last
In First Out)



O Tipo Abstrato Pilha

O Tipo Abstrato Pilha

Ou seja, as operações que descrevem o comportamento da Pilha.



Quais seriam estas
operações, vocês são
capazes de enumerá-las ?

1. Empilhar

1. Empilhar

2. Desempilhar

1. Empilhar

2. Desempilhar

3. Verificar o valor do
topo

1. Empilhar

2. Desempilhar

3. Verificar o valor do topo

4. Verificar se a pilha
está vazia

1. Empilhar

2. Desempilhar

3. Verificar o valor do topo

4. Verificar se a pilha está vazia

5. Cria uma Pilha

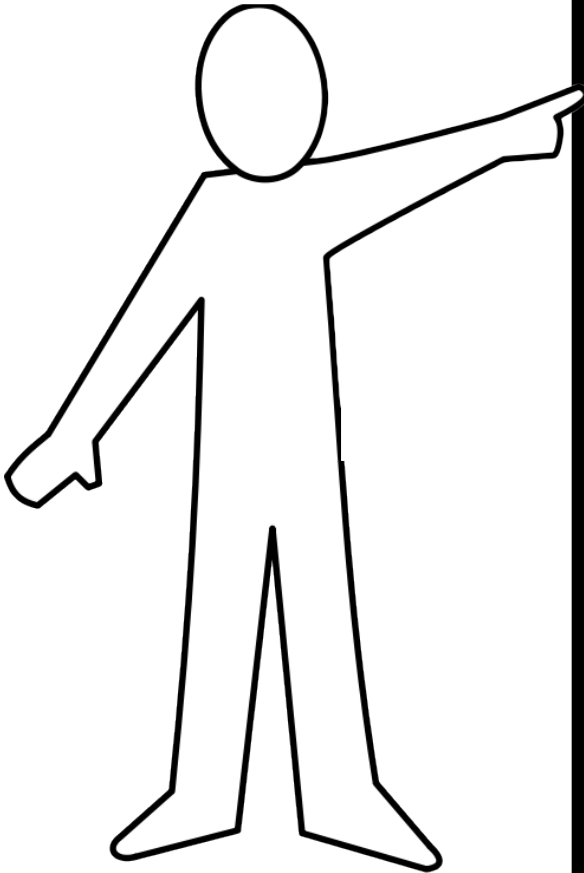
TAD: Pilha

Significado das operações (P sendo uma pilha e x um elemento)

1. *cria_pilha* () - Inicializa uma pilha, retornando uma dada pilha P
2. *empilha*(P, x) - Acrescenta o elemento x no topo da pilha P (o tamanho de P aumenta)
3. *desempilha* (P) - Remove e retorna o elemento no topo da pilha (o tamanho de P diminui)
4. *pilha_vazia* (P) - Verifica se a pilha está vazia (ou seja, não possui nenhum elemento)
5. *topo* (P) – Retorna o valor que está no topo da pilha

Qual será o estado final da pilha após estas operações ?

```
p = cria_pilha()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```

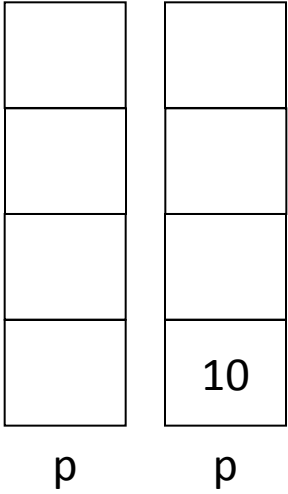




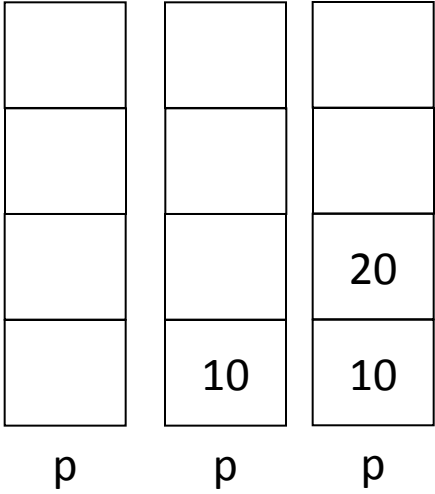
p

→

```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```



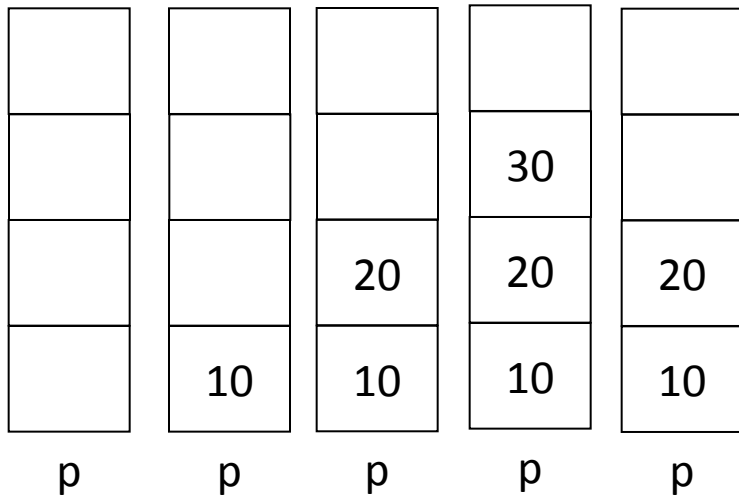
```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```



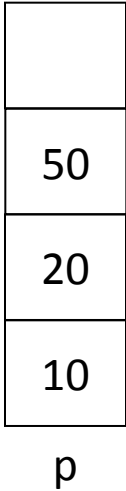
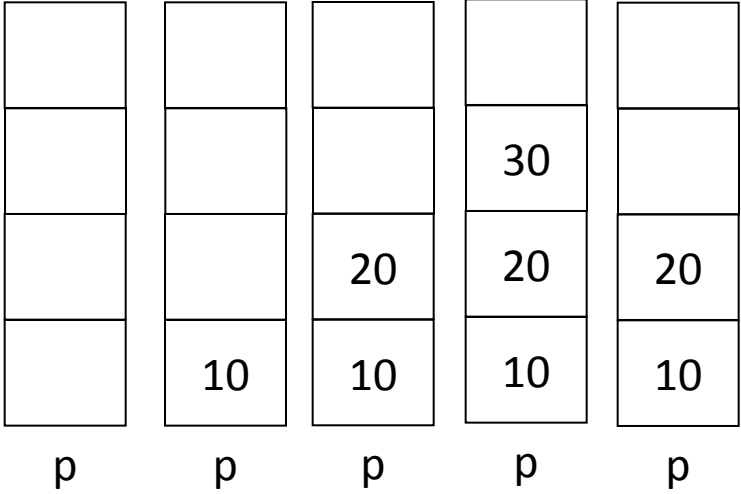
```
p = cria_pilha ()
empilha (p, 10)
empilha (p, 20)
empilha (p, 30)
desempilha (p)
empilha (p, 50)
desempilha (p)
desempilha (p)
empilha (p, 90)
```



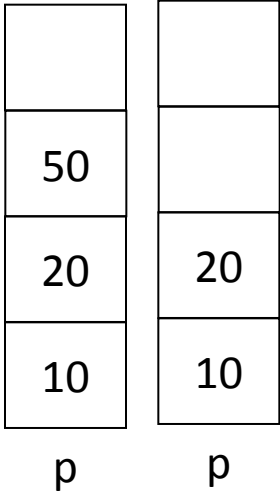
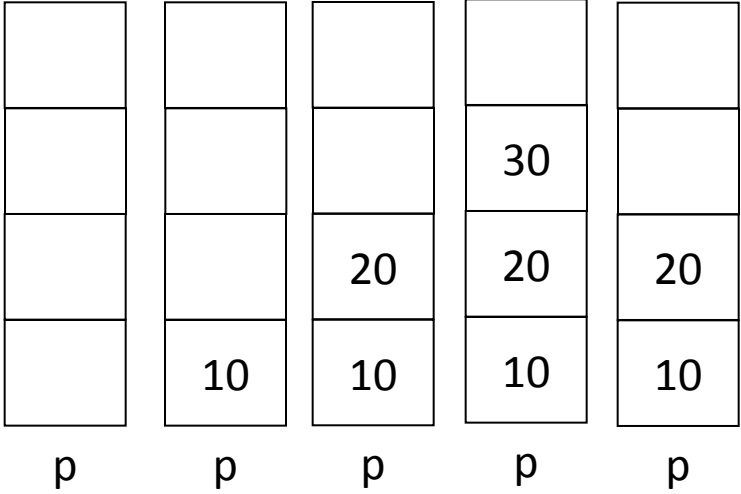
```
p = cria_pilha ()
empilha (p, 10)
empilha (p, 20)
empilha (p, 30)
desempilha (p)
empilha (p, 50)
desempilha (p)
desempilha (p)
empilha (p, 90)
```

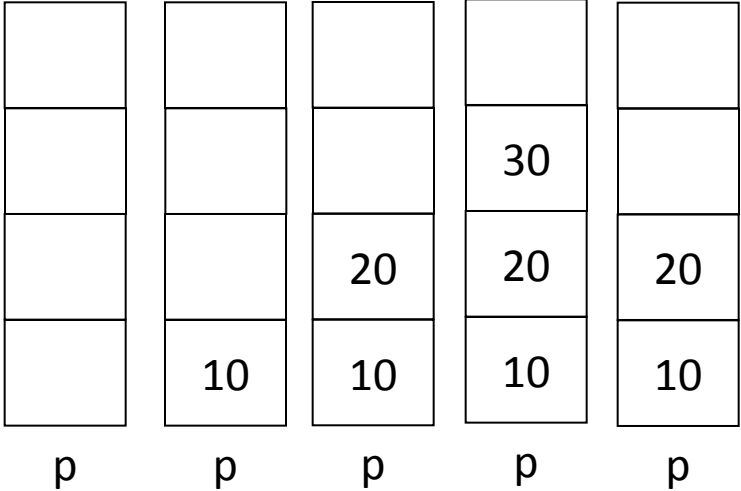
```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```



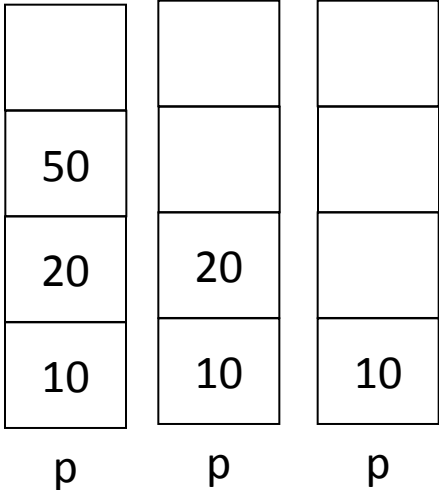
```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```

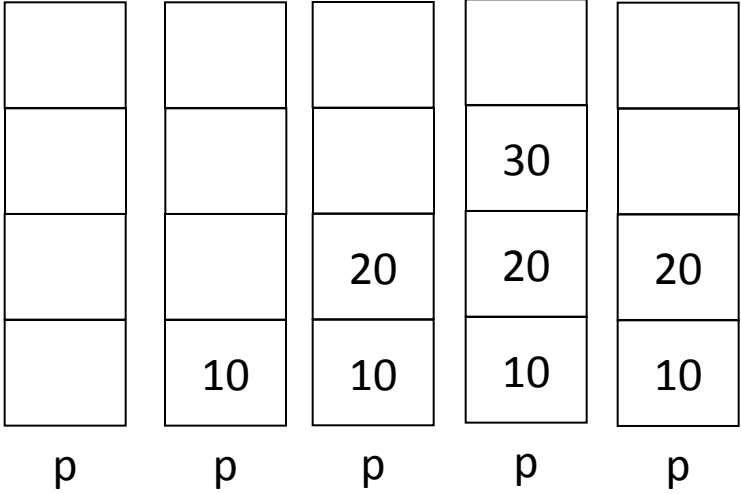


```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
→ empilha (p, 90)
```

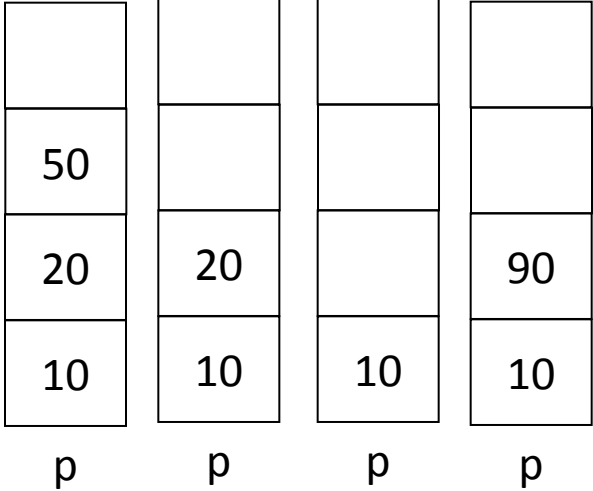


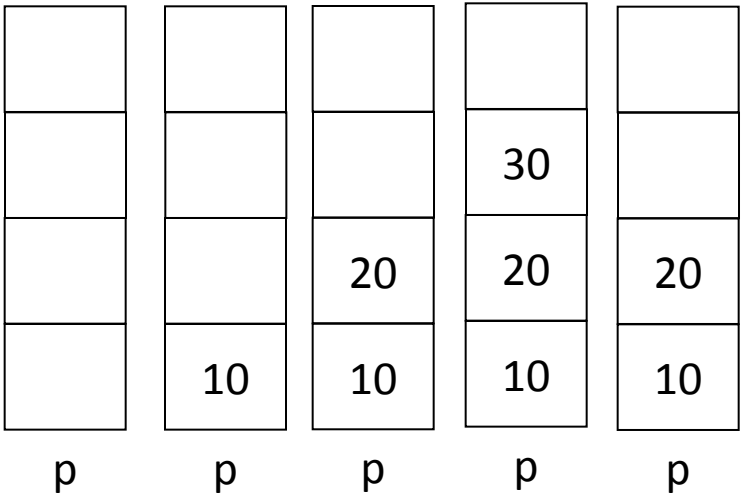
```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```



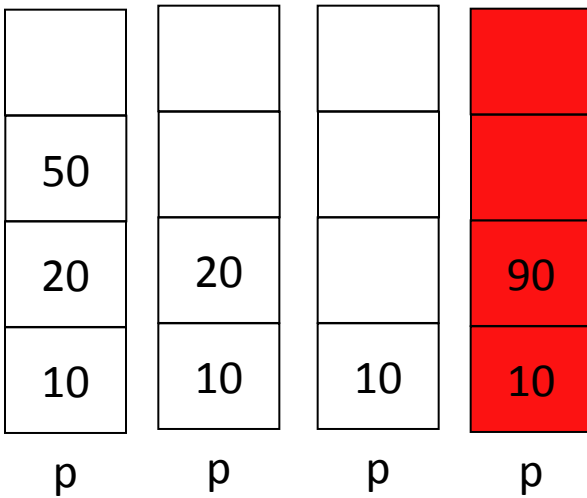


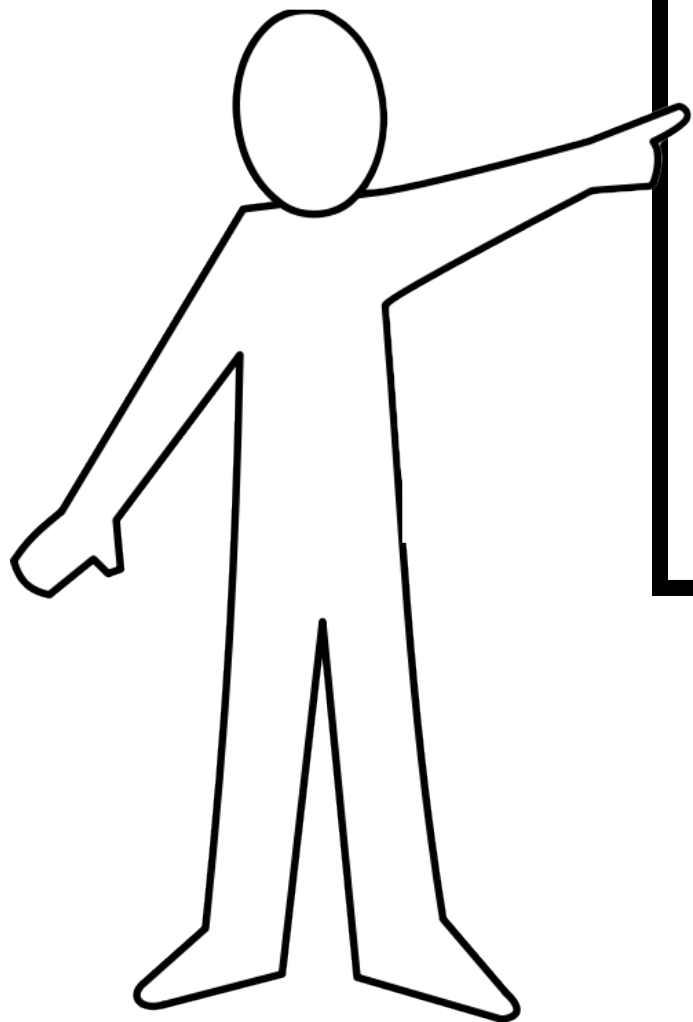
```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```



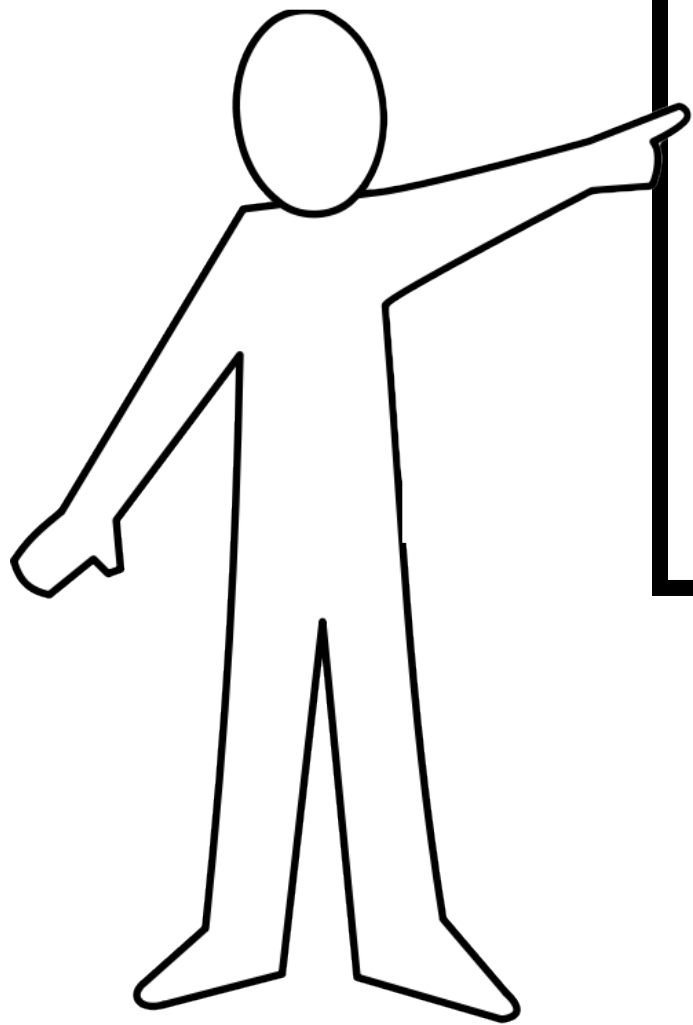


```
p = cria_pilha ()  
empilha (p, 10)  
empilha (p, 20)  
empilha (p, 30)  
desempilha (p)  
empilha (p, 50)  
desempilha (p)  
desempilha (p)  
empilha (p, 90)
```

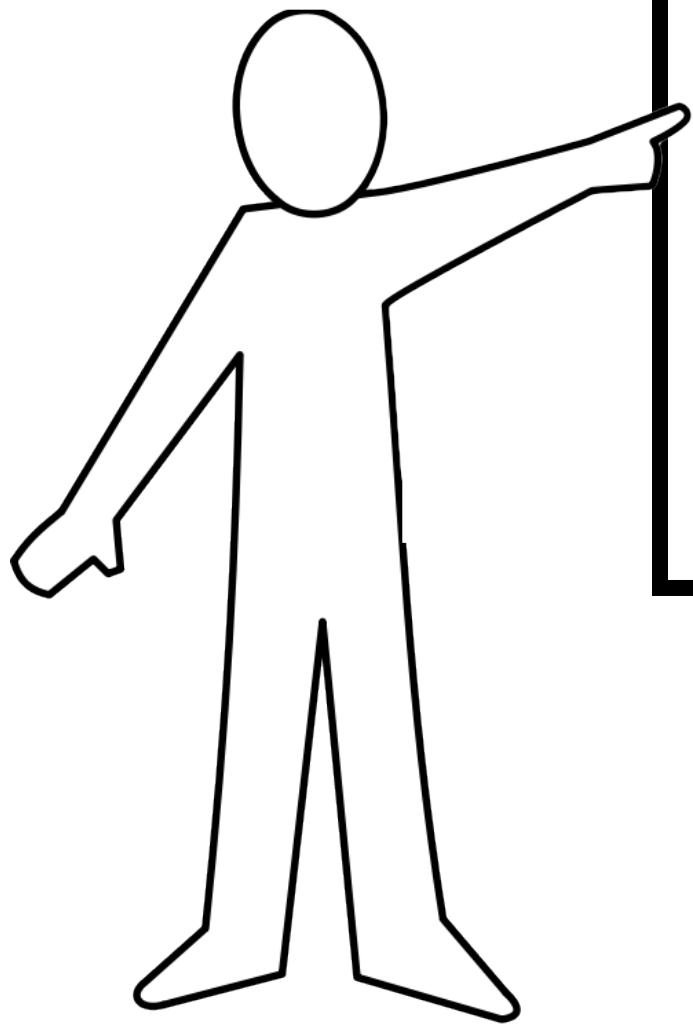




Entenderam o conceito da
pilha ?



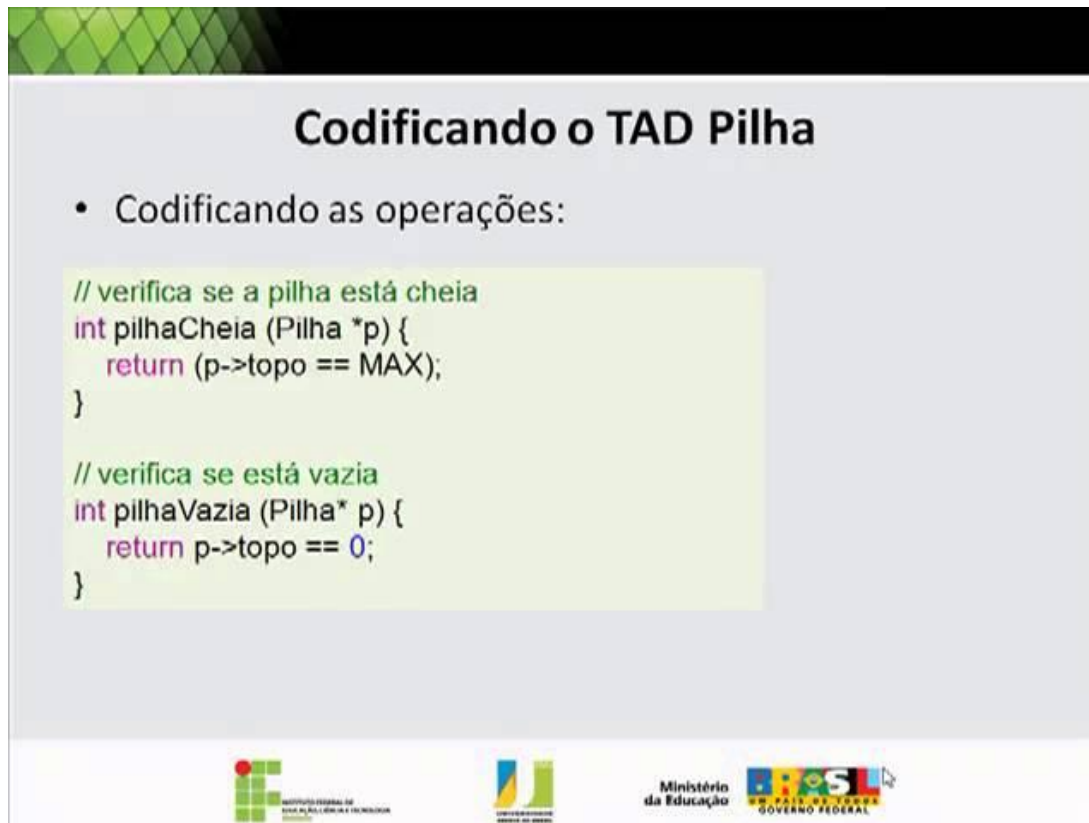
Como podemos codificar
esta estrutura ?



Que elementos podemos
usar ?

Vídeo Aula

Esta aula pode ser vista também no vídeo abaixo:



Codificando o TAD Pilha

- Codificando as operações:

```
// verifica se a pilha está cheia
int pilhaCheia (Pilha *p) {
    return (p->topo == MAX);
}

// verifica se está vazia
int pilhaVazia (Pilha* p) {
    return p->topo == 0;
}
```

Logos at the bottom: Instituto Federal de Mato Grosso do Sul, Universidade Federal do Rio de Janeiro, Ministério da Educação, and Governo Federal.

TAD: Pilha

Uma possibilidade é usar vetores, dados que estes permitem armazenar uma coleção de dados.

TAD: Pilha

Uma possibilidade é usar vetores, dados que estes permitem armazenar uma coleção de dados.

Uma pilha que usa vetor como estrutura básica é chamada de pilha estática.

TAD: Pilha

Uma possibilidade é usar vetores, dados que estes permitem armazenar uma coleção de dados.

Uma pilha que usa vetor como estrutura básica é chamada de pilha estática.

Na Unidade II, veremos como codificar uma pilha dinâmica usando listas encadeadas.

Codificando o TAD Pilha

- Antes de prosseguir, pause a aula e tentem rascunhar em papel como seria as operações:
 - Empilha e
 - Desempilha

Codificando o TAD Pilha

- Ao usar um vetor como estrutura para a nossa pilha precisamos:
 - Saber em qual posição do vetor o elemento deve ser empilhado.
 - Saber quando a pilha encheu, ou seja, precisaremos de uma operação que verifica se a pilha encheu.

Codificando o TAD Pilha

- O que vamos precisar para codificar a nossa pilha?

Codificando o TAD Pilha

- O que vamos precisar para codificar a nossa pilha?

Registros

Codificando o TAD Pilha

- O que vamos precisar para codificar a nossa pilha?

Registros

Funções

Codificando o TAD Pilha

- O que vamos precisar para codificar a nossa pilha?

Registros

Funções

Ponteiros

Codificando o TAD Pilha

- O que vamos precisar para codificar a nossa pilha?

Registros

Funções

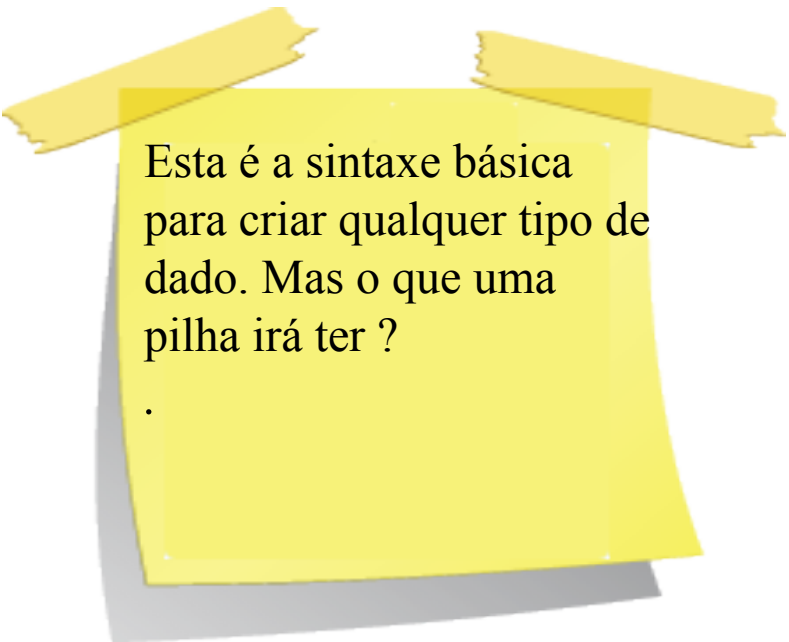
Ponteiros

Alocação Dinâmica

Codificando o TAD Pilha

Criando o tipo de dado Pilha ...

```
typedef struct {  
  
} Pilha;
```

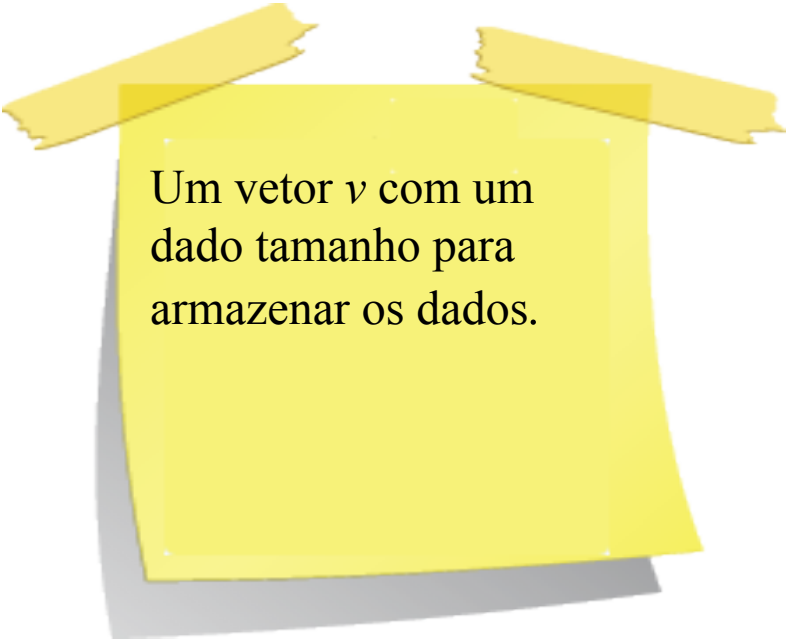
A yellow sticky note with a torn top edge, held in place by two pieces of yellow tape. It contains text in Portuguese.

Esta é a sintaxe básica
para criar qualquer tipo de
dado. Mas o que uma
pilha irá ter ?
.

Codificando o TAD Pilha

Criando o tipo de dado Pilha ...

```
typedef struct {  
    int v[10];  
  
} Pilha;
```



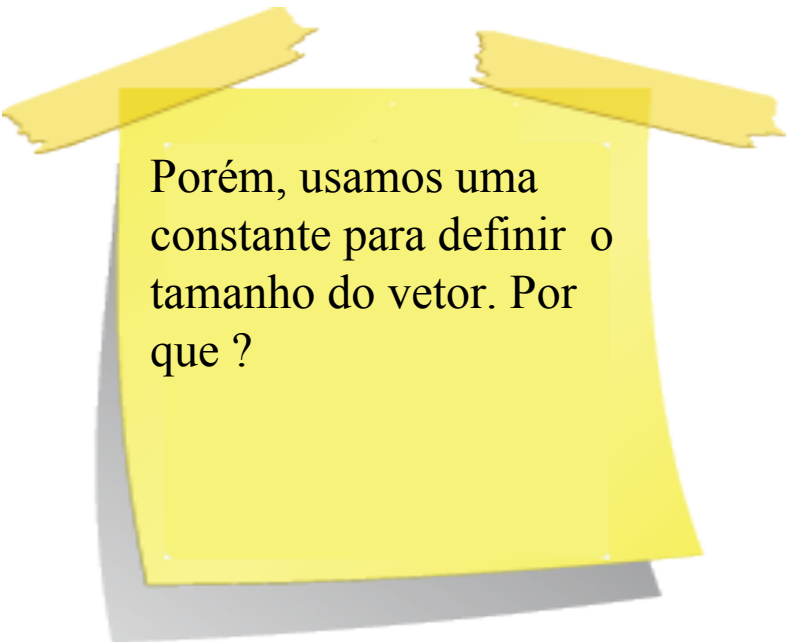
Um vetor v com um
dado tamanho para
armazenar os dados.

Codificando o TAD Pilha

Criando o tipo de dado Pilha ...

```
#define MAX 10
typedef struct {
    int v[MAX];

} Pilha;
```

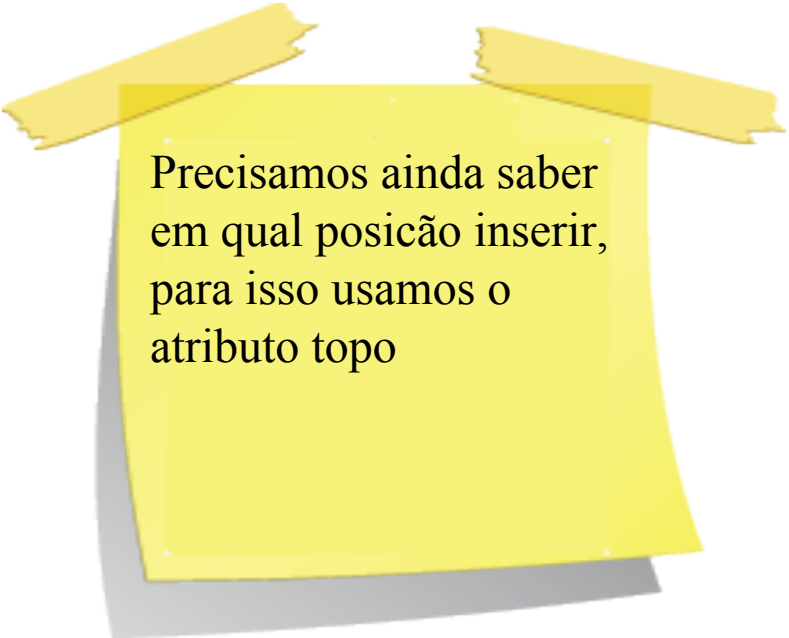


Porém, usamos uma constante para definir o tamanho do vetor. Por que ?

Codificando o TAD Pilha

Criando o tipo de dado Pilha ...

```
#define MAX 10
typedef struct {
    int v[MAX];
    int topo;
} Pilha;
```



Precisamos ainda saber em qual posição inserir, para isso usamos o atributo topo

Codificando o TAD Pilha

Codificando as operações:

```
Pilha* criaPilha (){  
    Pilha* p = (Pilha*)  
                malloc (sizeof(Pilha));  
    p->topo = 0;  
    return p;  
}
```

Codificando o TAD Pilha

Codificando as operações:

```
// verifica se a pilha está cheia
```

```
int pilhaCheia (Pilha *p) {  
    return (p->topo == MAX);  
}
```

```
// verifica se está vazia
```

```
int pilhaVazia (Pilha* p) {  
    return p->topo == 0;  
}
```

Codificando o TAD Pilha

Codificando as operações:

```
void empilha (Pilha *p, int x) {  
    p->v[p->topo++] = x;  
}  
  
int desempilha (Pilha* p) {  
    return p->v[--p->topo];  
}  
  
// retorna o elemento do topo, sem desempilhar  
int topo (Pilha* p) {  
    return p->v[p->topo-1];  
}
```

Codificando o TAD Pilha


Codificando as operações:



```
void empilha (Pilha *p, int x) {  
    p->v[p->topo++] = x;  
}  
  
int desempilha (Pilha* p) {  
    return p->v[--p->topo];  
}  
// retorna o elemento do topo, sem desempilhar  
int topo (Pilha* p) {  
    return p->v[p->topo-1];  
}
```

Codificando o TAD Pilha

Codificando as operações:



```
void empilha (Pilha *p, int x) {  
    p->v[p->topo] = x;  
    p->topo++;  
}
```

```
int desempilha (Pilha *p) {  
    return p->v[--p->topo];  
}
```

// retorna o elemento do topo, sem desempilhar

```
int topo (Pilha* p) {  
    return p->v[p->topo-1];  
}
```

Codificando o TAD Pilha

Codificando as operações:

```
void empilha (Pilha *p, int x) {  
    p->v[p->topo++] = x;  
}
```



```
int desempilha (Pilha* p) {  
    return p->v[--p->topo];  
}
```

// retorna o elemento do topo, sem desempilhar

```
int topo (Pilha* p) {  
    return p->v[p->topo-1];  
}
```

Codificando o TAD Pilha

Codificando as operações:

```
void empilha (Pilha *p, int x) {  
    p->v[p->topo++] = x;  
}  
  
int desempilha (Pilha *p) {  
    p->topo--;  
    return p->v[p->topo];  
}  
  
// retorna o elemento do topo, sem desempilhar  
int topo (Pilha* p) {  
    return p->v[p->topo-1];  
}
```



Codificando o TAD Pilha

Codificando as operações:

```
void empilha (Pilha *p, int x) {  
    p->v[p->topo++] = x;  
}  
  
int desempilha (Pilha* p) {  
    return p->v[--p->topo];  
}  
// retorna o elemento do topo, sem desempilhar  
int topo (Pilha* p) {  
    return p->v[p->topo-1];  
}
```



Codificando o TAD Pilha

- Aplicação Cliente: Testando nossa pilha:
 - O que será impresso ?

```
int main () {  
    Pilha* p = criaPilha ();  
    empilha (p, 10);  
    empilha (p, 30);  
    empilha (p, 40);  
    printf ("%d\n", desempilha (p));  
    printf ("%d\n", desempilha (p));  
    printf ("%d\n", desempilha (p));  
}
```

APLICAÇÕES

Outros exemplos

- Pilhas podem ser utilizadas para verificar se os parênteses em uma expressão está balanceado.

Exemplo:

$((3+4 + (4*9)$ **ERRO: Faltam dois parênteses fechando!**

Como?

Para cada carácter da expressão faça

1. Se encontrou um "(" empilha
2. Se encontrou um ")" então
 1. Se a pilha estiver vazia: expressão invalida
 2. Caso contrario desempilhe
3. Se no final pilha estiver vazia, expressão válida.

Aplicações

- Como o uso das pilhas é possível simplificar diversos algoritmos.
 - O próprio sistema operacional utiliza pilha para tratar as chamadas a funções.
 - Uma outra aplicação comum é na avaliação de expressões e de parênteses.

Identificando palíndromos

- Palíndromos são palavras ou frases que são iguais quando lidas de frente para trás
- Exemplos:
 - ANA
 - ARARA
 - ROTOR
 - SOCORRAM ME SUBI NO ONIBUS EM MARROCOS (não considerando os espaços em branco)

Identificando palíndromos

- Algoritmo

1. Empilhe $n/2$ letras
2. Descarta letra central se houver
3. Repita
 1. Compare o topo da pilha com a proxima letra
 2. Se são iguais entao Desempilhe
 3. Senao Não é um palíndromo
4. Se no final a pilha está vazia entao a palavra é um palíndromo

Identificando palíndromos

ARARA

Topo =0

Base =1

1	2	3	4	5	6	7	8

Identificando palíndromos

A R A R A

Topo = 1

Base = 1

A							
1	2	3	4	5	6	7	8

Empilha letra 1

Identificando palíndromos

ARARA

Topo = 2

Base = 1

A	R						
1	2	3	4	5	6	7	8

Empilha letra 2

Identificando palíndromos

ARA RA

Topo = 2

Base = 1

A	R						
1	2	3	4	5	6	7	8

Descarta letra central

Identificando palíndromos

ARA**RA**

Topo = 2

Base = 1

A	R						
1	2	3	4	5	6	7	8

Comparar topo com próxima letra

Identificando palíndromos

ARARA

Topo =1

Base =1

A							
1	2	3	4	5	6	7	8

Se são iguais então desempilhe

Identificando palíndromos

ARARA

Topo =1

Base =1

A							
1	2	3	4	5	6	7	8

Comparar topo com próxima letra

Identificando palíndromos

ARARA

Topo =1

Base =1

1	2	3	4	5	6	7	8

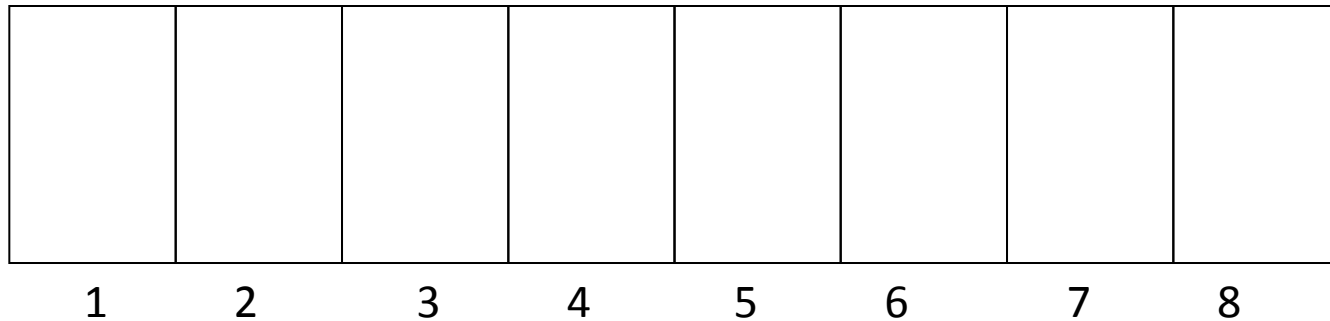
Se são iguais então desempilhe

Identificando palíndromos

ARARA

Topo =1

Base =1



Se no final a pilha está vazia então é um palíndromo

Outros exemplos

- Pilhas podem ser utilizadas para verificar se os parênteses em uma expressão está balanceado.

Exemplo:

$((3+4 + (4*9)$ **ERRO: Faltam dois parênteses fechando!**

Como?

Outros exemplos

- Pilhas podem ser utilizadas para verificar se os parênteses em uma expressão está balanceado.

Exemplo:

$((3+4 + (4*9)$ **ERRO: Faltam dois parênteses fechando!**

Como?

Para cada carácter da expressão faça

- Se encontrou um "(" empilha
- Se encontrou um ")" então
 - Se a pilha estiver vazia: expressão invalida
 - Caso contrario desempilhe
- Se no final pilha estiver vazia, expressão válida.

Notação Infixa - Avaliação

Operadores entre operandos.

$$4 + 6 * 8$$

Como decidir quais operadores são avaliados primeiros ?

- Abordagem usual é usar as mesmas **regras** da matemática.
- Caso a **precedência** seja a mesma, avalia-se da esquerda para direita
- Parênteses tem **precedência** sobre todos operadores.

Notação Polonesa

É fácil observar que devido a estas regras, o algoritmo para avaliar expressões na notação infixa não é muito trivial.

O matemático polonês Jan Łukasiewicz criou uma notação em torno de 1920 que elimina a necessidade de regras.

Ficou sendo conhecida como notação polonesa.

Não é muito usado na matemática convencional, mas muito usado nas ciências da computação devido sua simplicidade.

Avaliação de expressões - Notações

Infixa: $A + (B * C) / D$

Posfixa (polonesa reversa): $A B C * + D /$

Préfixa (polonesa): $+ A / * B C D$

Avaliação de expressões - Notações

Infixa: $A + (B * C) / D$

Posfixa (polonesa reversa): $A B C * + D$

Préfixa (polonesa): $+ A / * B C D$

Exemplos de uso da notação polonesa



```
RAD XYZ HEX R= 'X'  
> Eng. Mecânica> 10 53 AUG:01  
5:  
4:  
3:  
2:  
1:  
STACK MEM BRCH TEST TYPE LIST
```

Linguagens:

Scheme e Lisp

Algoritmo de avaliação de expressão na notação posfixa.

Notação Posfixa

A avaliação de expressões nesta notação é a mais simples. Percorrendo uma expressão da esquerda para direita, sabemos que ele deve operar os dois últimos valores encontrados (considerando que os operadores são binários, por exemplo os aritméticos).

Notação posfixa

Considerem a seguinte expressão $7\ 3\ +\ 5\ *$

Expressão	Elemento	Ação	Pilha
73+5*			P:[]
3+5*	7	Empilhar	P:[7]
+5*	3	Empilhar	P:[3,7]
5*	+	Desempilha 3 Desempilha 7 Empilha 3+7	P:[7] P:[] P:[10]
*	5	Empilhar	P:[5,10]
	*	Desempilha 5 Desempilha 10 Empilha 5*10	P:[10] P:[] P:[50]

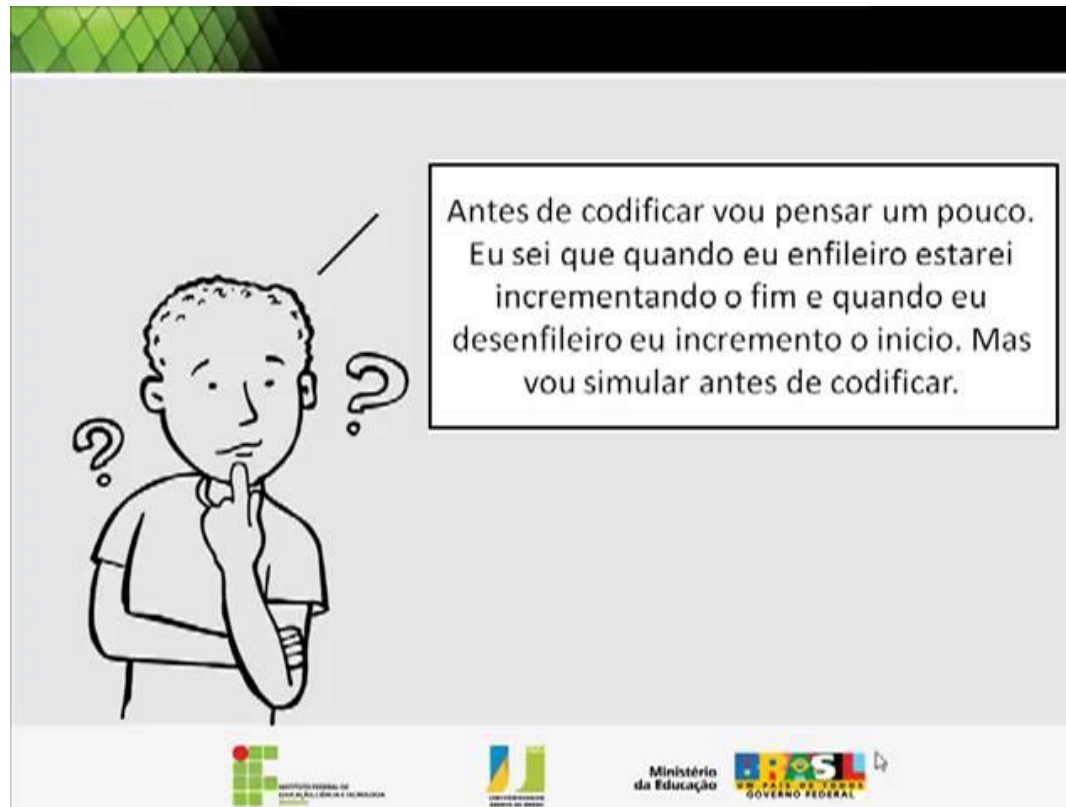
Se a pilha tiver mais de um elemento, erro, caso contrário retorna o único valor da pilha.

Referências Bibliográficas

- AARON, Tanenbaun. **Estruturas de Dados usando C.** São Paulo: Makron Books, 1995.
- PEREIRA, Silvio. **Estruturas de Dados Fundamentais.** São Paulo: Editora Erica, 1996.
- VELOSO, Paulo et al. **Estruturas de Dados.** Rio de Janeiro: Editora Campus, 1983.

Vídeo Aula

Esta aula pode ser vista também no vídeo abaixo:



Fila: Conceito

Conceito base: O primeiro a entrar é o primeiro a sair. Em Inglês (Fifo): **F**irst **I**n **F**ist **O**ut

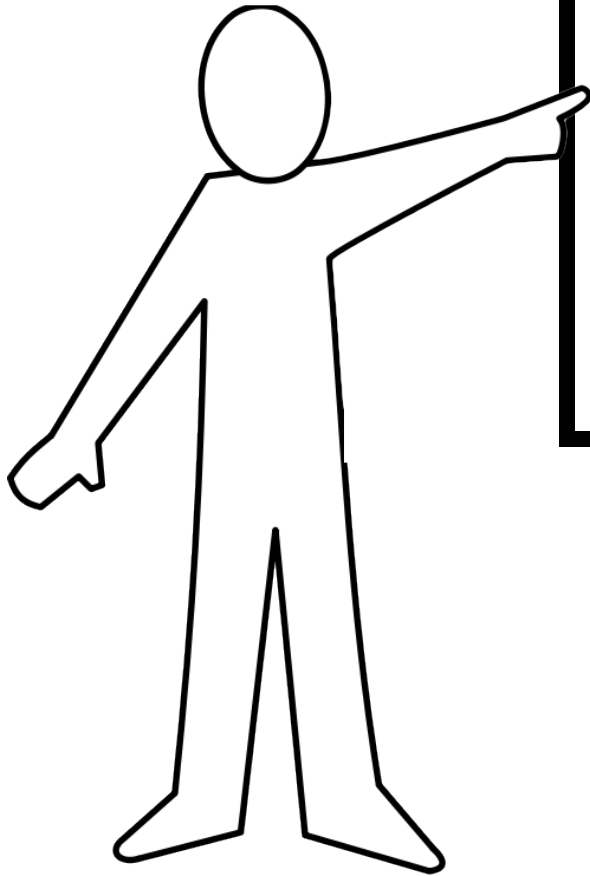


- Fila de banco
- Fila de pacientes
- Fila do supermercado

O Tipo Abstrato Fila

O Tipo Abstrato Fila

Ou seja, as operações que descrevem o comportamento da Fila.



Quais seriam estas
operações, são capazes de
enumerá-las ?

1. criarFila

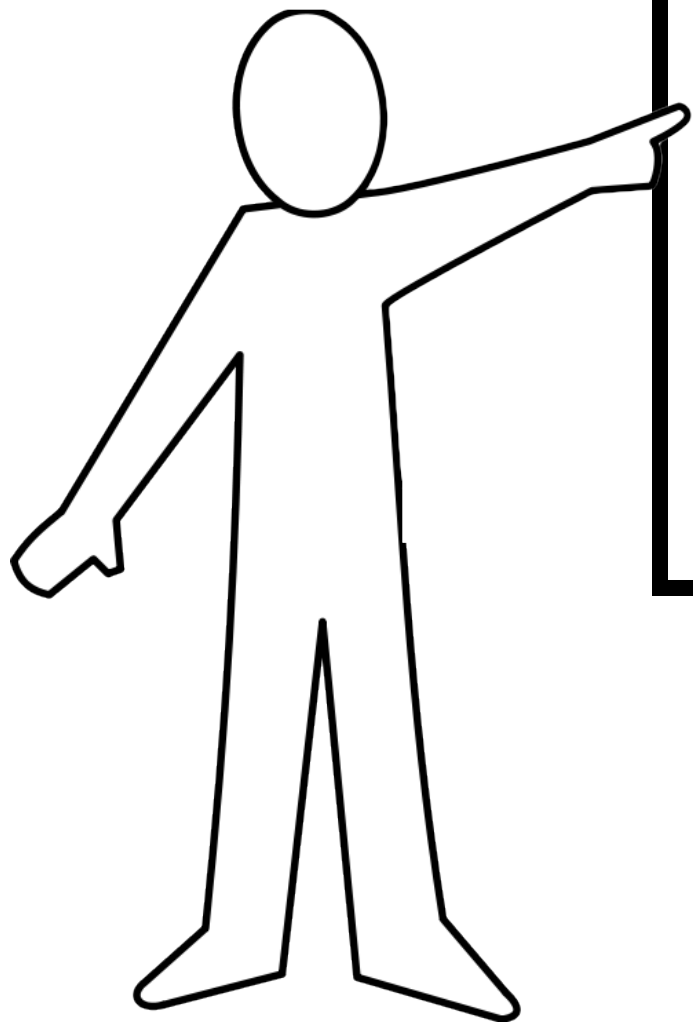
1. criarFila

2. enfileirar

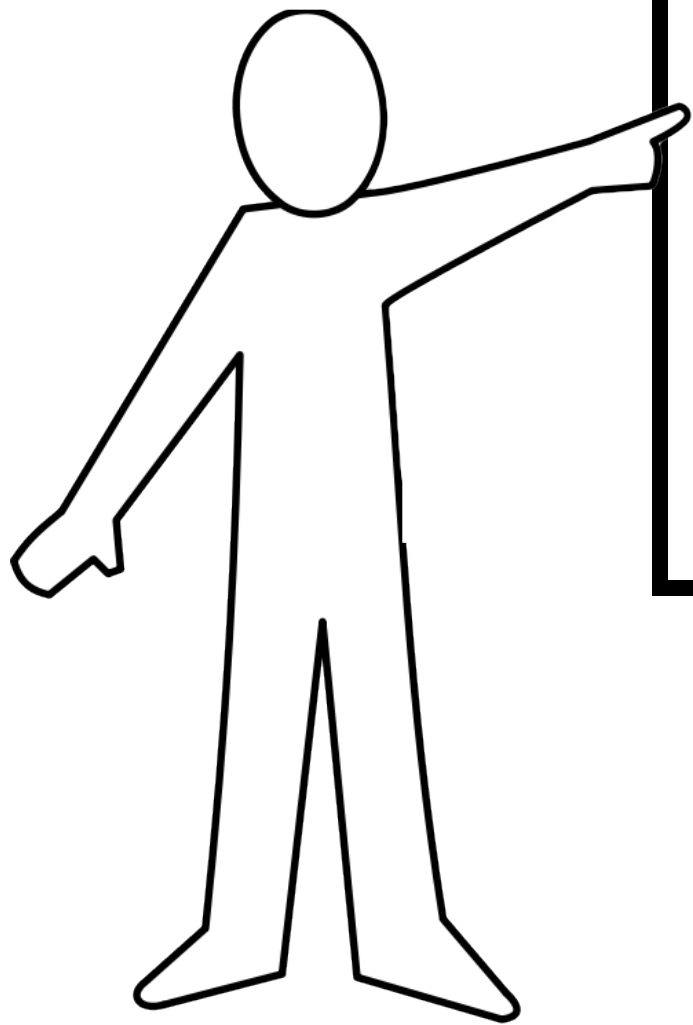
1. criarFila

2. enfileirar

3. desenfileirar



Como podemos codificar
esta estrutura ?



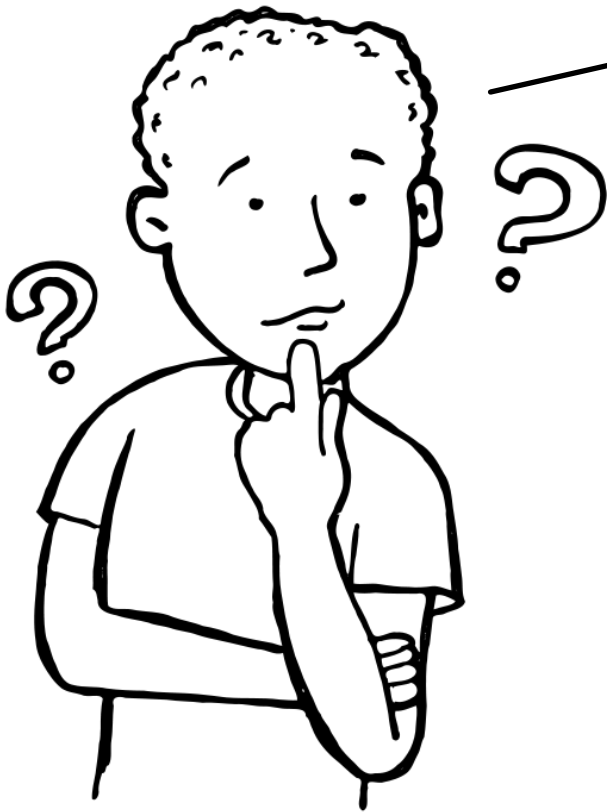
Quais elementos podemos
usar ?

TAD: Fila

Novamente, uma possibilidade é usar vetores, dado que estes permitem armazenar uma coleção de dados.

Uma fila que usa vetor como estrutura básica é chamada de fila estática.

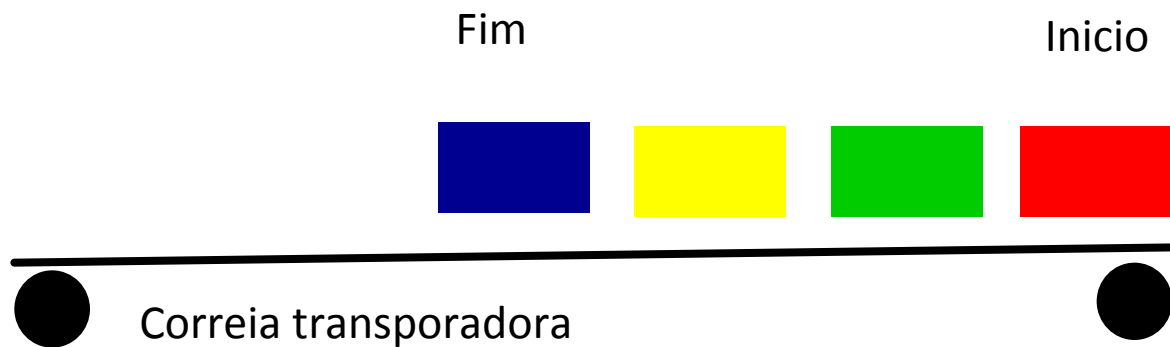
Na Unidade III, veremos como codificar uma fila dinâmica usando listas encadeadas.



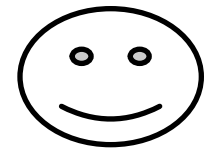
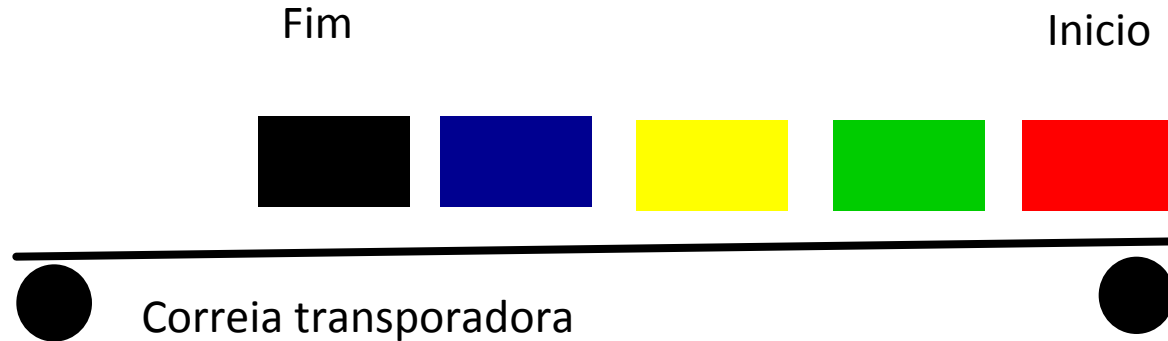
Estava pensando, e observei que na pilha a entrada e saída é pelo mesmo “lado”.
Porém na fila é diferente, quem chega vai para fim da fila e sai quem está no início da fila.



Imaginando, um trabalho em série, onde um conjunto de peças percorre uma correia até ela ir para empacotamento teríamos algo assim:

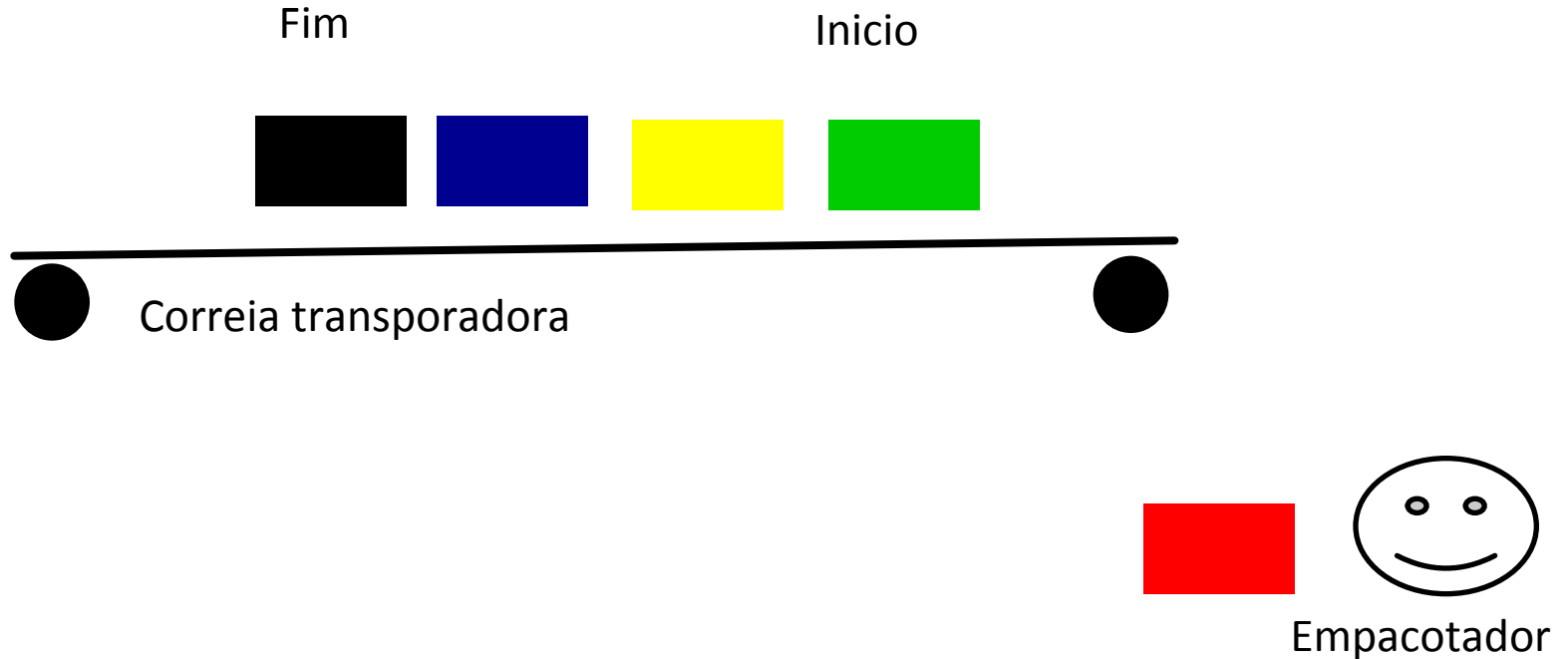


Quando chega um novo produto, ele é colocado no fim da fila.

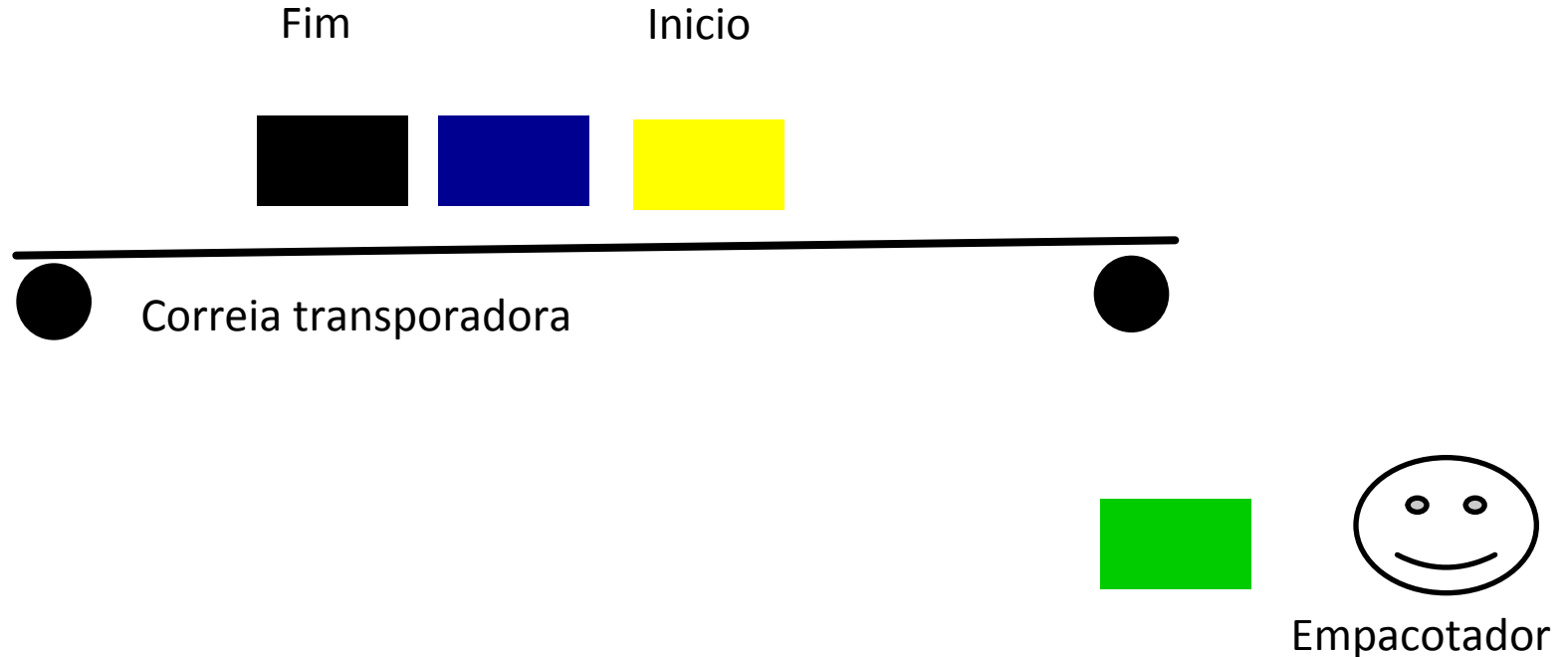


Empacotador

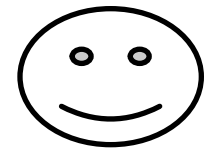
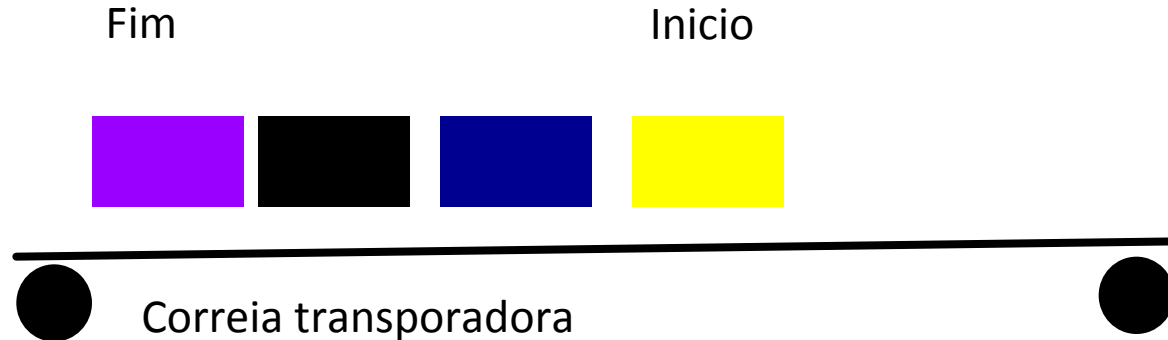
Quando um produto chega ao fim do processo,
a peça anterior passa a ser o início da fila.



Quando um produto chega ao fim do processo,
a peça anterior passa a ser o início da fila.



Quando chega um novo produto, ele é colocado no fim da fila.



Empacotador

CODIFICANDO ...

TAD: Fila

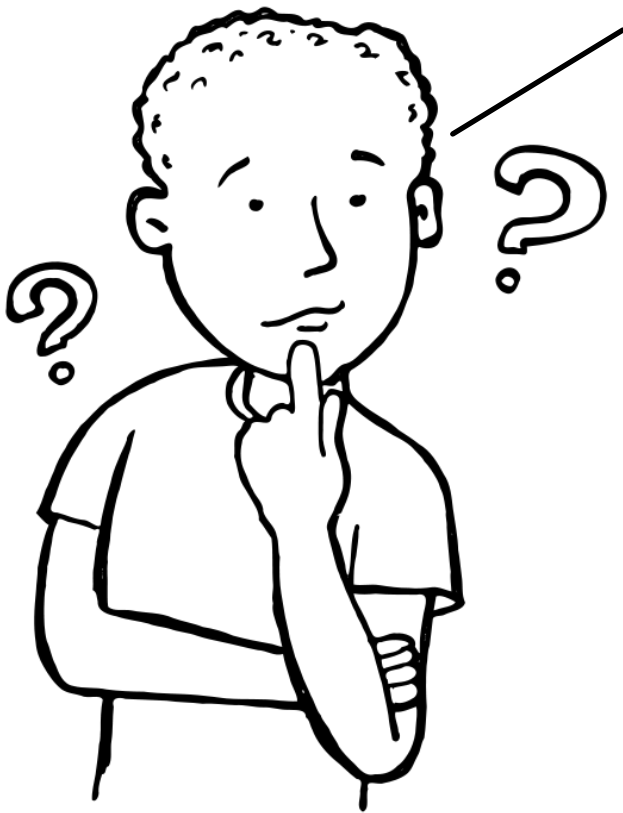
Definindo o tipo de dados Fila. Observe, que tivemos que incluir os dois atributos, que marcam o inicio e o fim da fila.

```
typedef struct {  
    int v[MAX];  
    int inicio, fim;  
}Fila;
```

TAD: Fila

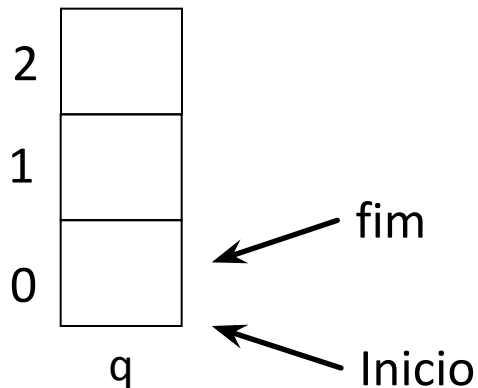
Codificando as operações:

```
Fila* criaFila () {  
    Fila* q = (Fila*)  
        malloc (sizeof(Fila));  
    q->inicio = 0;  
    q->fim = 0;  
    return q;  
}
```



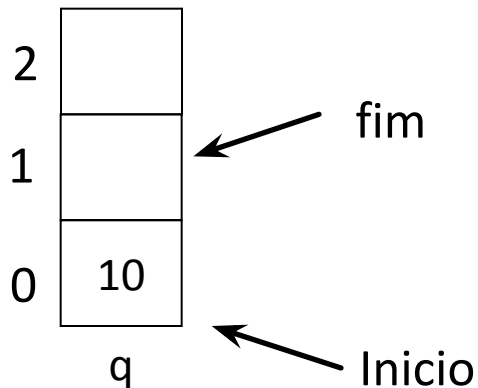
Antes de codificar vou pensar um pouco.
Eu sei que quando eu enfileiro estarei
incrementando o fim e quando eu
desenfileiro eu incremento o inicio. Mas
vou simular antes de codificar.

Considerando uma fila de no máximo 3 elementos:



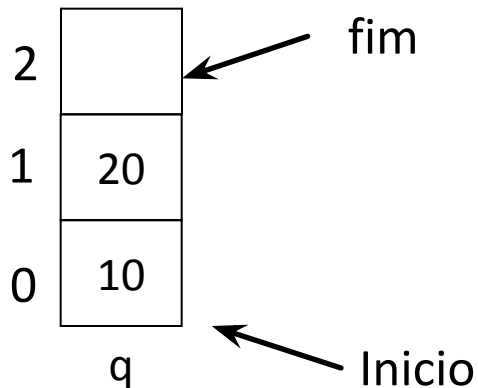
```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```


Considerando uma fila de no máximo 3 elementos:



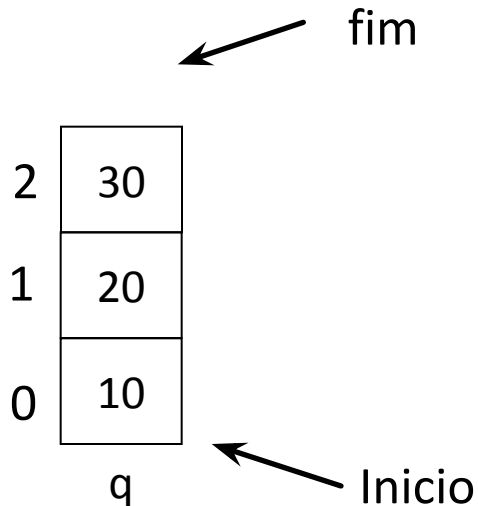
```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

Considerando uma fila de no máximo 3 elementos:



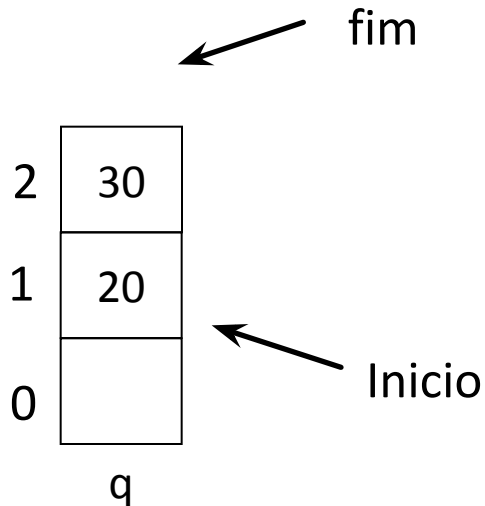
```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

Considerando uma fila de no máximo 3 elementos:



```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

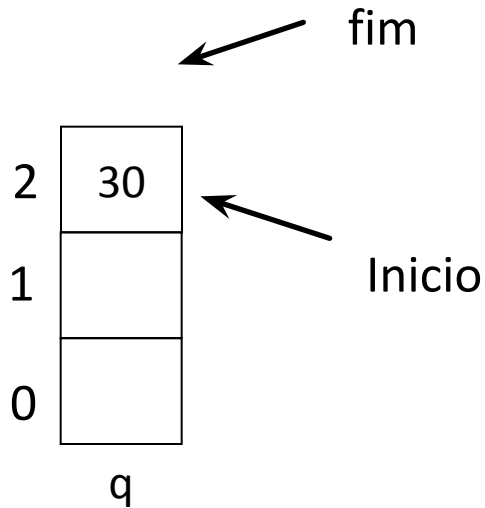
Considerando uma fila de no máximo 3 elementos:



10

```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

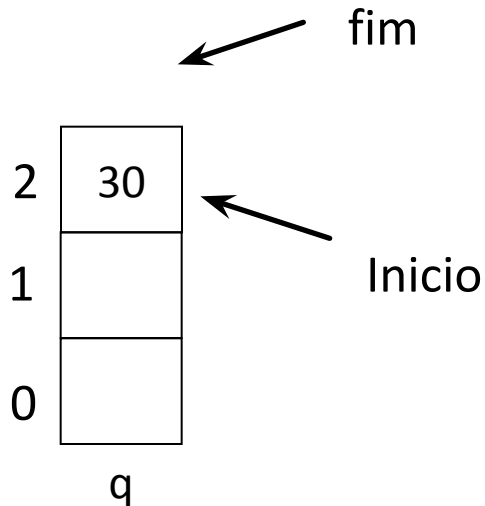
Considerando uma fila de no máximo 3 elementos:



10
20

```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

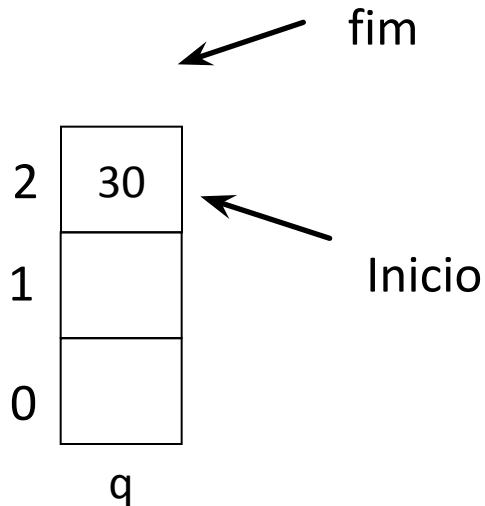
Considerando uma fila de no máximo 3 elementos:



10
20

```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

Considerando uma fila de no máximo 3 elementos:

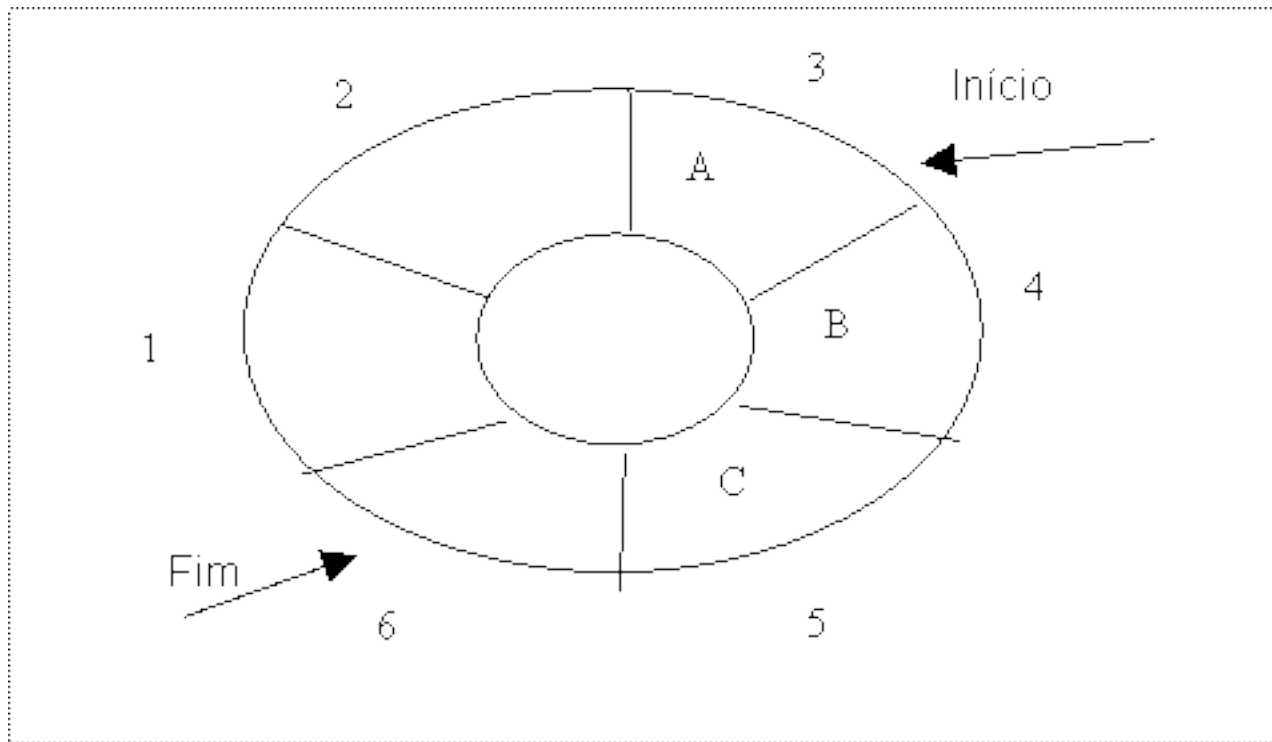


Erro: Mesmo tendo áreas vagas, eu não posso mais adicionar elemento.

```
Fila *q = criaFila ();
enqueue (q, 10);
enqueue (q, 20);
enqueue (q, 30);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
enqueue (q, 50);
enqueue (q, 60);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
```

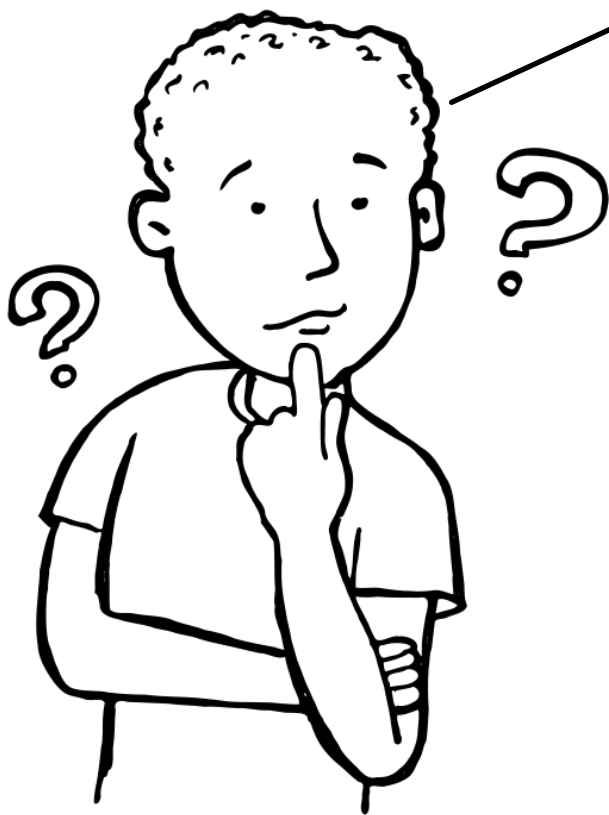
TAD: Fila

Solução, vetor circular:



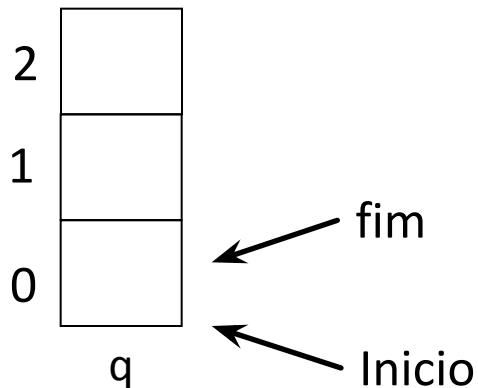
Depois do último elemento, temos o primeiro e antes do primeiro temos o último.

Fonte: <http://www2.dc.ufscar.br/~bsi/materiais/ed/u5.html>



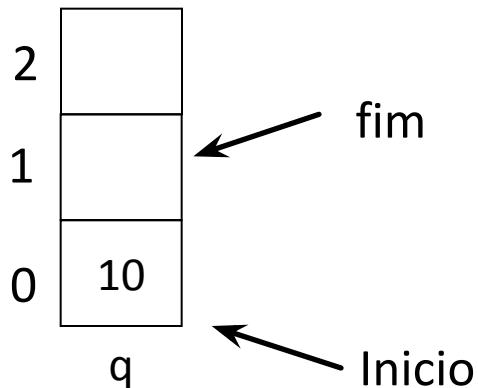
Agora vou simular novamente, mas considerando meu vetor circular.

Considerando uma fila de no máximo 3 elementos:



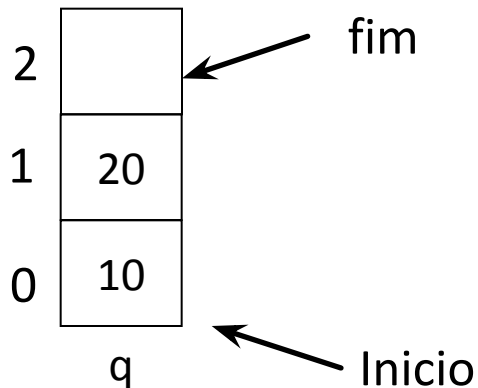
```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

Considerando uma fila de no máximo 3 elementos:



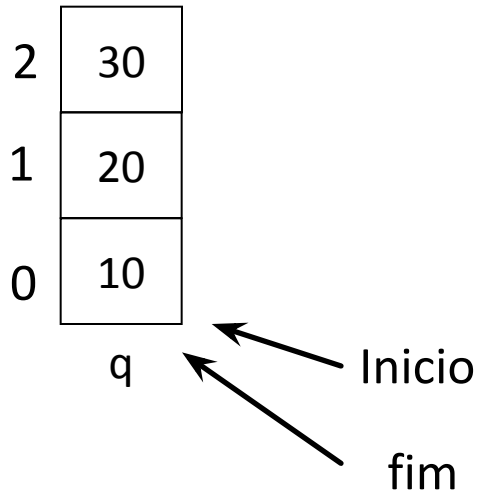
```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

Considerando uma fila de no máximo 3 elementos:



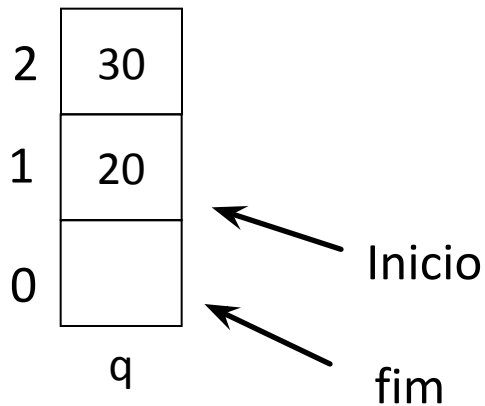
```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

Considerando uma fila de no máximo 3 elementos:



```
Fila *q = criaFila ();
enqueue (q, 10);
enqueue (q, 20);
enqueue (q, 30);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
enqueue (q, 50);
enqueue (q, 60);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
```

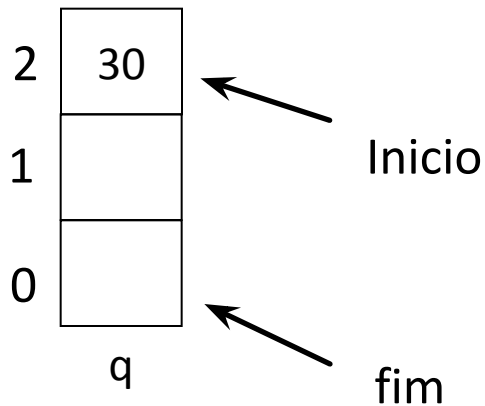
Considerando uma fila de no máximo 3 elementos:



10

```
Fila *q = criaFila ();
enqueue (q, 10);
enqueue (q, 20);
enqueue (q, 30);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
enqueue (q, 50);
enqueue (q, 60);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
```

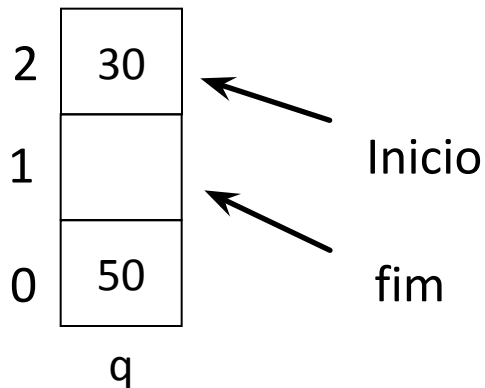
Considerando uma fila de no máximo 3 elementos:



10
20

```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

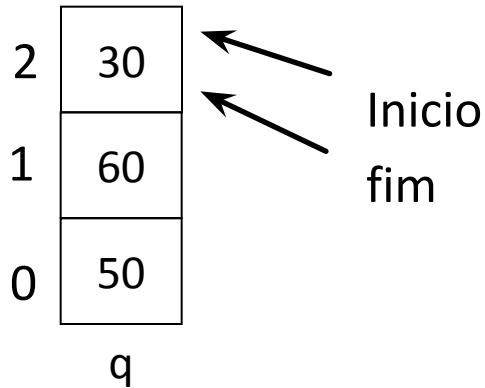
Considerando uma fila de no máximo 3 elementos:



10
20

```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

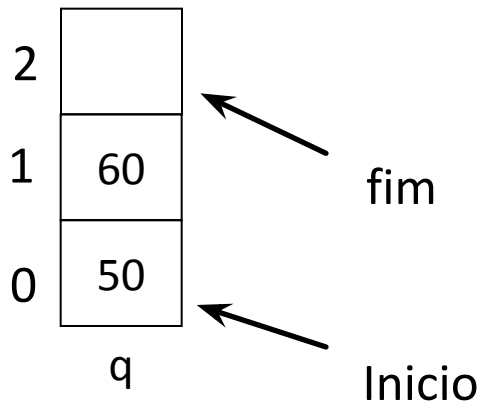

Considerando uma fila de no máximo 3 elementos:



10
20

```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

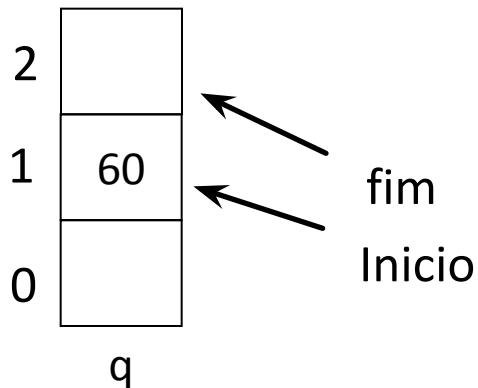
Considerando uma fila de no máximo 3 elementos:



10
20
30

```
Fila *q = criaFila ();
enqueue (q, 10);
enqueue (q, 20);
enqueue (q, 30);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
enqueue (q, 50);
enqueue (q, 60);
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
printf ("%d\n", dequeue(q));
```

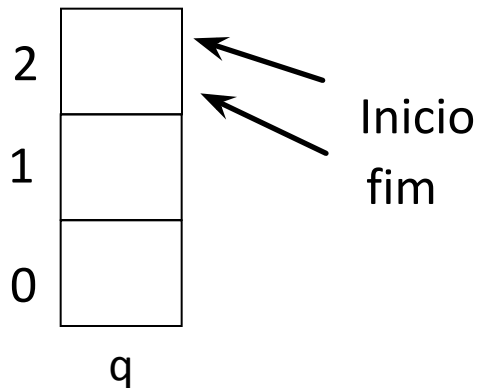
Considerando uma fila de no máximo 3 elementos:



10
20
30
50

```
Fila *q = criaFila ();  
enfileira (q, 10);  
enfileira (q, 20);  
enfileira (q, 30);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
enfileira (q, 50);  
enfileira (q, 60);  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));  
printf ("%d\n", desenfileira(q));
```

Considerando uma fila de no máximo 3 elementos:



10
20
30
50
60

```
Fila *q = criaFila ();  
enqueue (q, 10);  
enqueue (q, 20);  
enqueue (q, 30);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
enqueue (q, 50);  
enqueue (q, 60);  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));  
printf ("%d\n", dequeue(q));
```

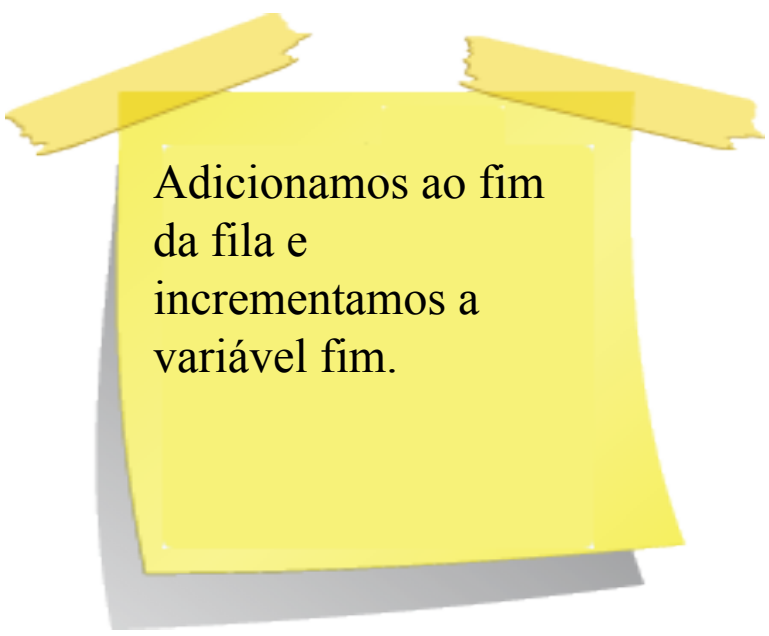
Codificando as operações:

```
void enfileira (Fila* q, int x) {  
    q->v[q->fim++] = x;  
    if (q->fim == MAX)  
        q->fim = 0;  
}
```

TAD: Fila

Codificando as operações:

```
void enfileira (Fila* q, int x) {  
    → q->v[q->fim++] = x;  
    if (q->fim == MAX)  
        q->fim = 0;  
}
```

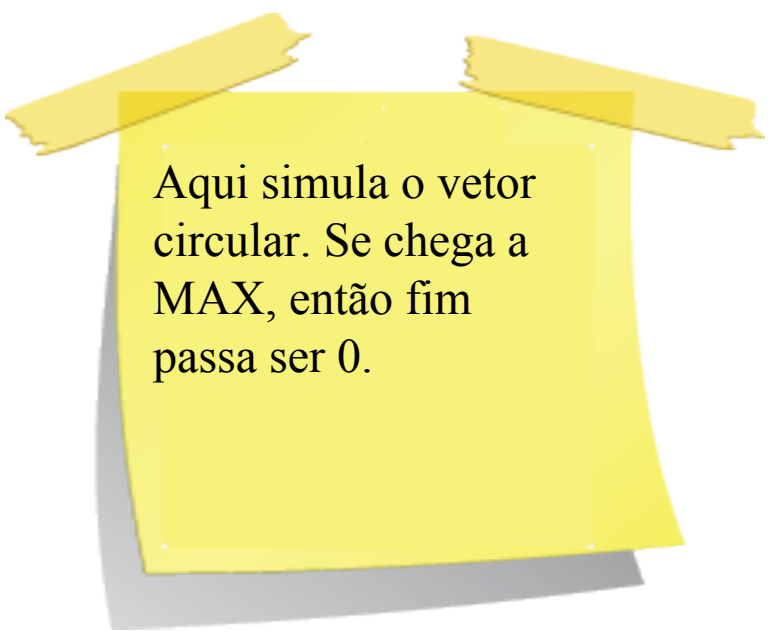


Adicionamos ao fim
da fila e
incrementamos a
variável fim.

TAD: Fila

Codificando as operações:

```
void enfileira (Fila* q, int x) {  
    q->v[q->fim++] = x;  
    → if (q->fim == MAX)  
        q->fim = 0;  
}
```

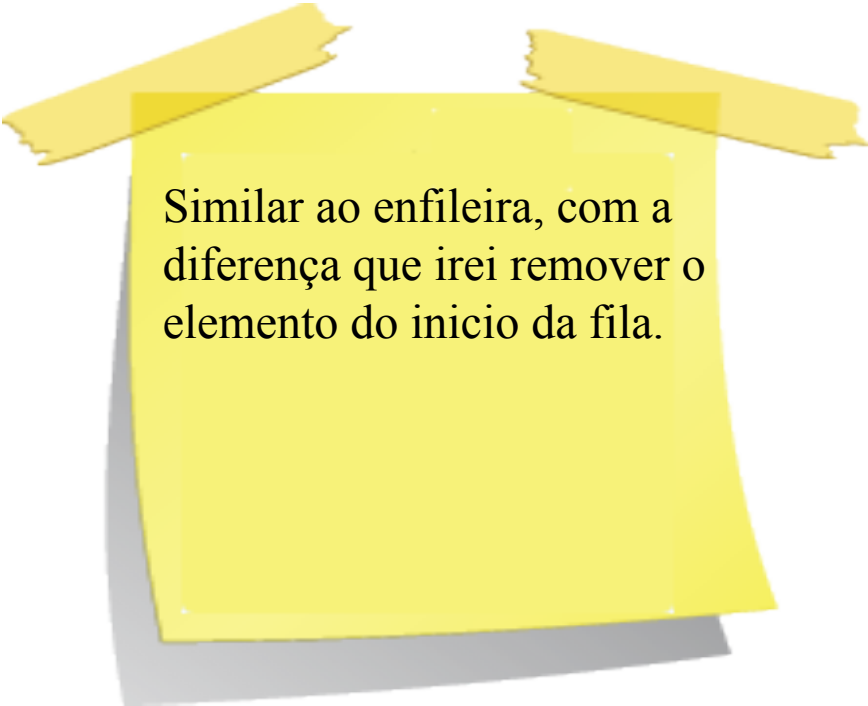


Aqui simula o vetor circular. Se chega a MAX, então fim passa ser 0.

TAD: Fila

Codificando as operações:

```
int desenfileira(Fila* q) {  
    int x = q->v[q->inicio++];  
    if (q->inicio == MAX)  
        q->inicio = 0;  
    return x;  
}
```




Similar ao enfileira, com a diferença que irei remover o elemento do início da fila.

TAD: Fila

Testando nossa codificação:

```
int main () {  
    Fila *q = criaFila ();  
    enfileira (q, 10);  
    enfileira (q, 20);  
    enfileira (q, 30);  
    printf ("%d\n", desenfileira(q));  
    printf ("%d\n", desenfileira(q));  
    enfileira (q, 40);  
    printf ("%d\n", desenfileira(q));  
    printf ("%d\n", desenfileira(q));  
    return 0;  
}
```



O que será
impresso ?

Exemplos de Aplicação

Fila: Aplicações

- Em controle de processos no sistema operacional, fila de processos.
- Em algoritmo de busca em grafo (busca em largura).
- Algoritmo de caminho mínimo (Dijkstra).
- Algoritmo de ordenação.

Referências Bibliográficas

- AARON, Tanenbaun. **Estruturas de Dados usando C.** São Paulo: Makron Books, 1995.
- PEREIRA, Silvio. **Estruturas de Dados Fundamentais.** São Paulo: Editora Erica, 1996.
- VELOSO, Paulo et al. **Estruturas de Dados.** Rio de Janeiro: Editora Campus, 1983.