UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Pilha com alocação estática

Prof. Rafael Liberato

liberato@utfpr.edu.br

Objetivos

- Entender o funcionamento de uma Pilha utilizando alocação Estática
- Ser capaz de implementar as operações definidas no TAD Pilha manipulando uma estrutura estática de armazenamento.

Roteiro

- TAD Pilha
- Pilha Estática
- Simulação
- Implementação

TAD Pilha

TAD Pilha

```
#define ItemType int

typedef struct{

}Stack;

Stack* createStack();
void initializeStack(Stack* stack);
int push(Stack* stack, ItemType e);
int pop(Stack* stack, ItemType* e);
int top(Stack* stack, ItemType* e);
void printStack(Stack* stack);
int containsStack(Stack* stack, ItemType *e);
int sizeStack(Stack* stack);
int isEmptyStack(Stack* stack);
```

Vamos identificar os atributos que representarão a Pilha estática

Estrutura utilizada para armazenar os dados

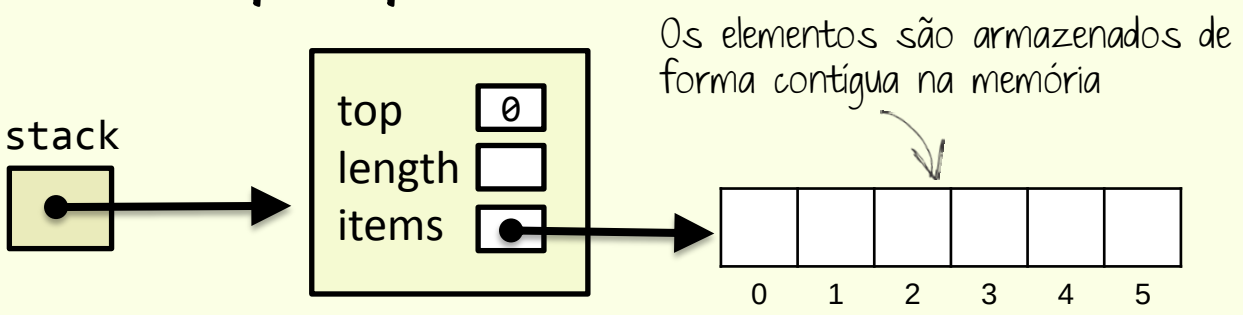
Pilha Estática

Pilha Estática

- A Pilha Estática utiliza uma estrutura de alocação estática de memória para o armazenamento dos dados
- A linguagem nos fornece essa estrutura por meio dos arranjos unidimensionais (vetores)
  - ➔ Os elementos da Pilha são armazenados em um vetor
- Para manter a eficiência da remoção na Pilha Estática vamos representar a base da pilha com a posição zero do array.

Pilha Estática

- **Atributos**
  - O atributo `items` armazena o endereço do array
  - O atributo `length` armazena a quantidade de espaços do array
  - O atributo `top` armazena a posição do primeira posição vazia da Fila. (top-1 é a posição do elemento que está no topo da Pilha). O atributo `top` também representa a quantidade de elementos que a pilha possui.




```
typedef struct{
    int top;
    int length;
    ItemType *items;
}Stack;
```

Simulação

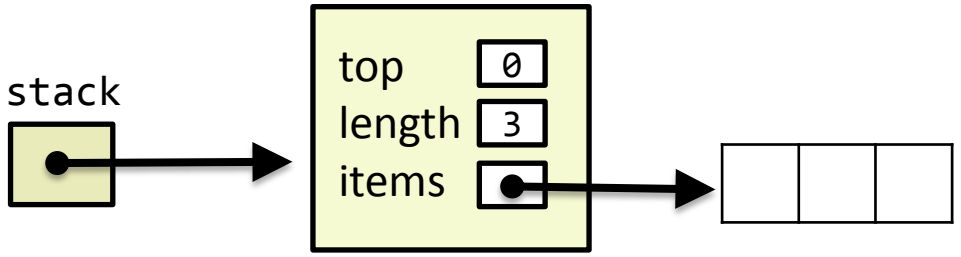
Simulação

- **Utilize a simulação para entender o comportamento das funções e auxiliá-lo na implementação.**

```
#define ItemType int

typedef struct{
int top;
    int length;
    ItemType *items;
}Stack;

Stack* createStack();
void initializeStack(Stack* stack);
int push(Stack* stack, ItemType e);
int pop(Stack* stack, ItemType* e);
int top(Stack* stack, ItemType* e);
void printStack(Stack* stack);
int containsStack(Stack* stack, ItemType *e);
int sizeStack(Stack* stack);
int isEmptyStack(Stack* stack);
```



Simulação

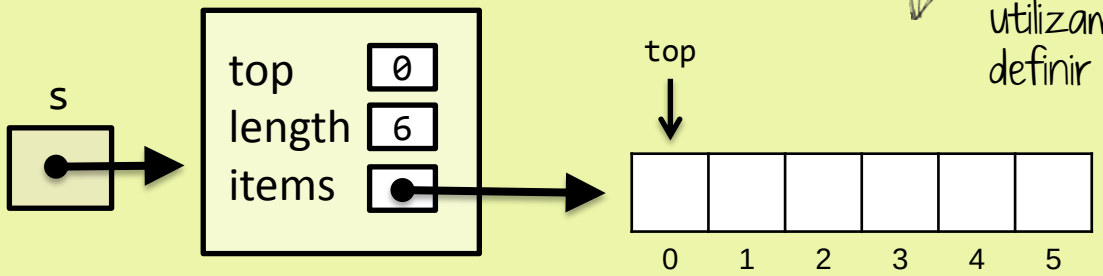
```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```

Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



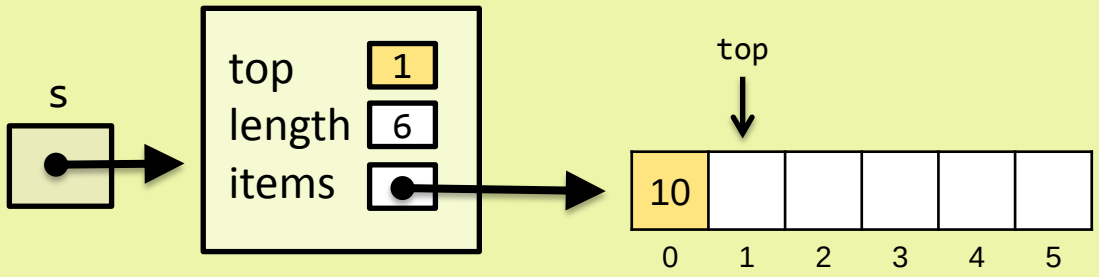
O tamanho do vetor é definido dentro da função. Normalmente utilizamos uma constante para definir o tamanho do vetor

PILHA VAZIA

Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

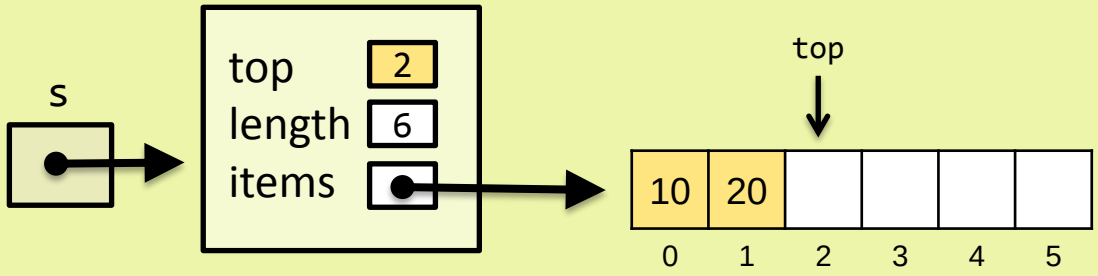
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

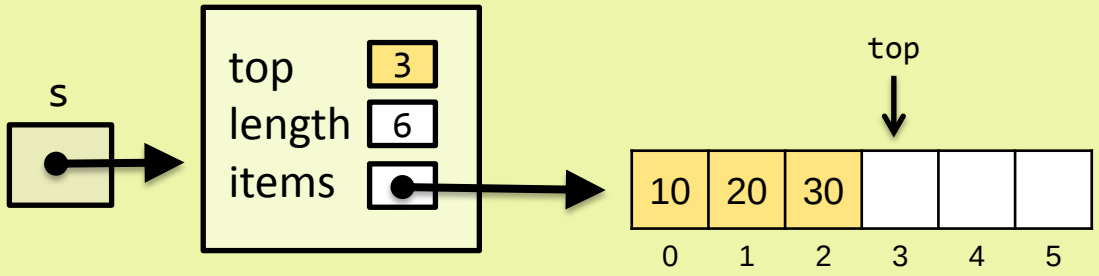
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

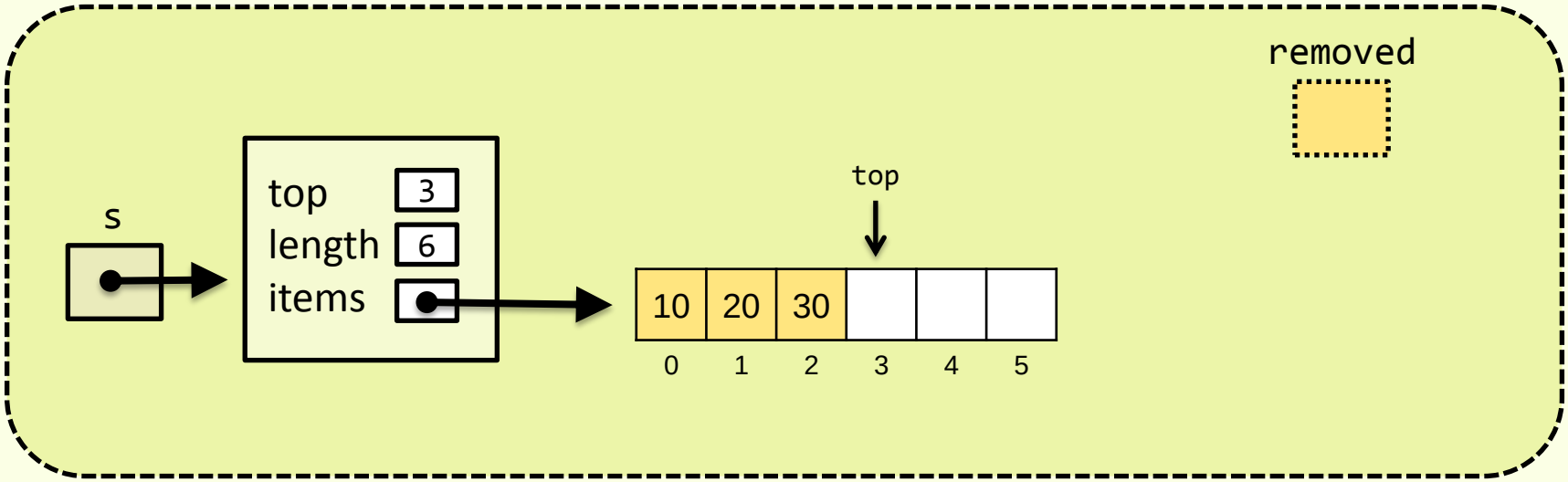
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

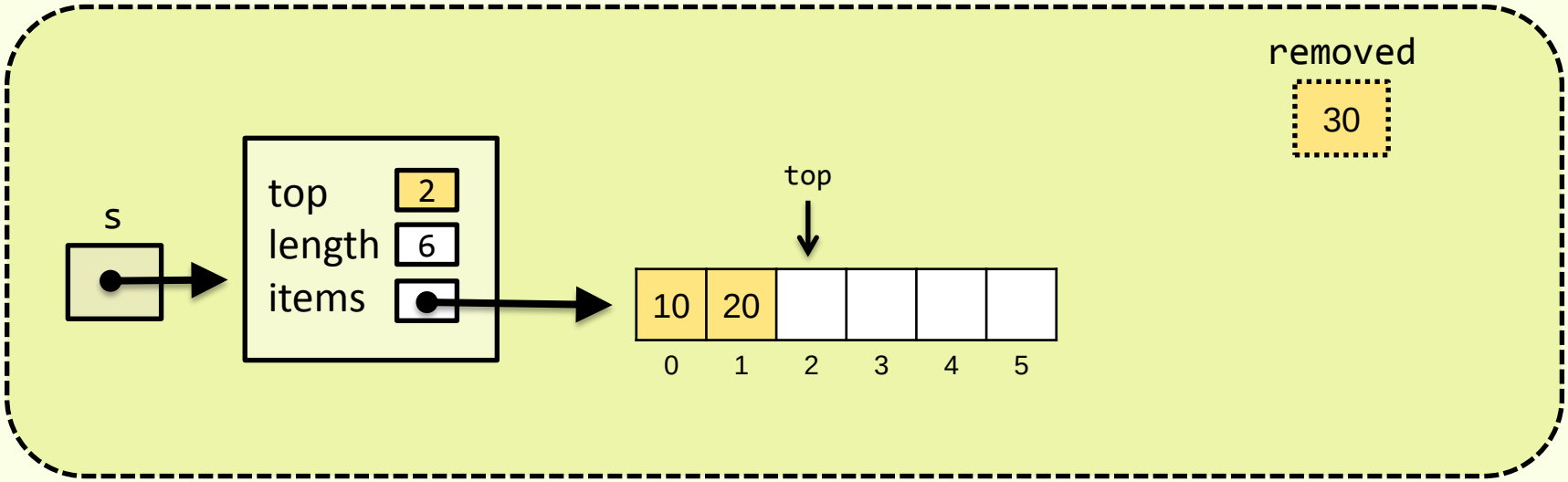
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

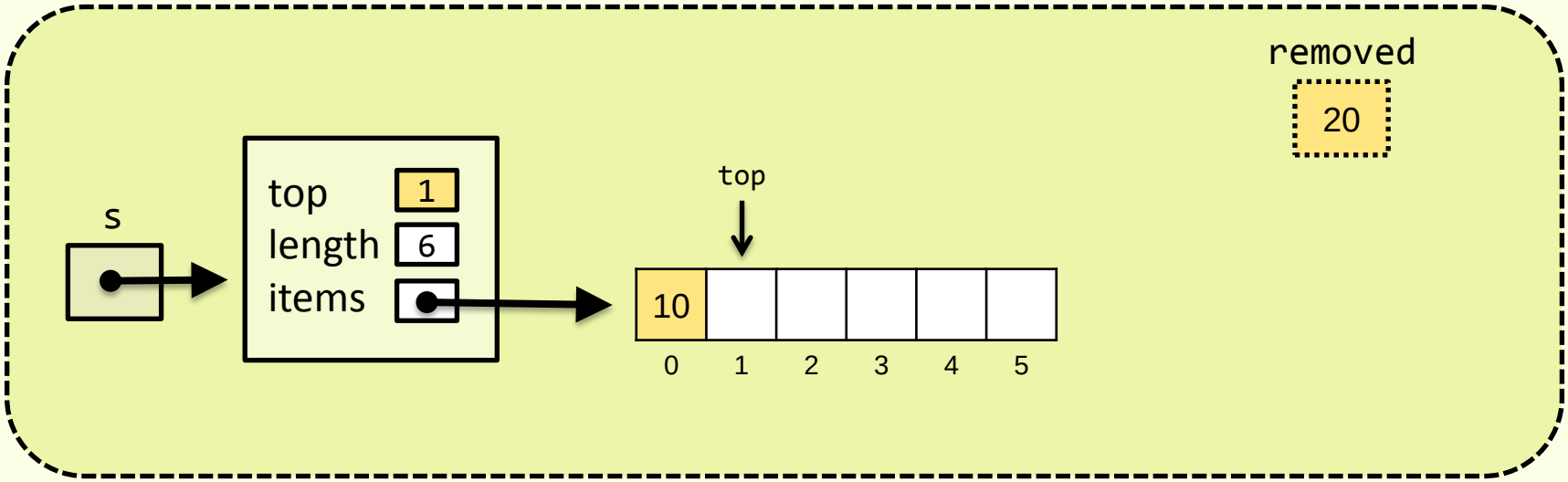
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

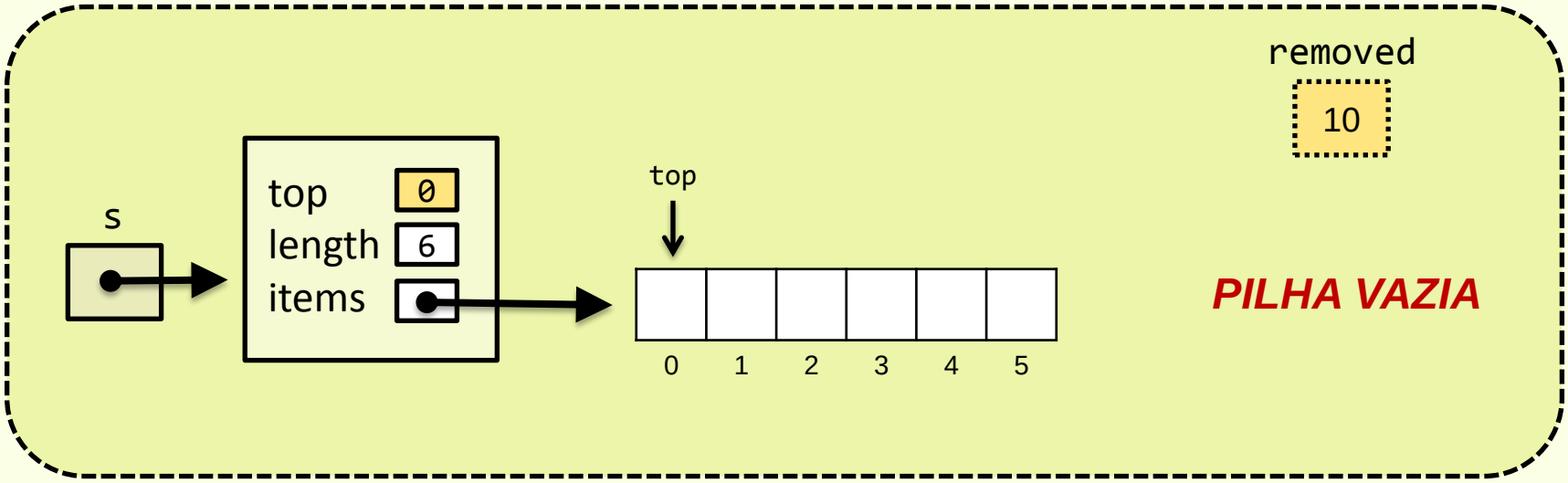
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

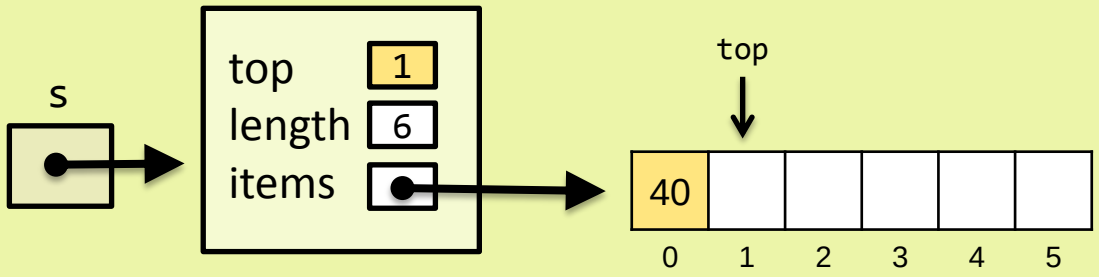
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

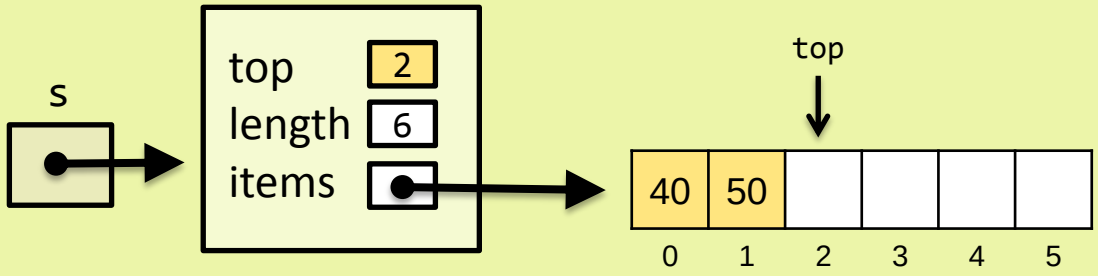
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

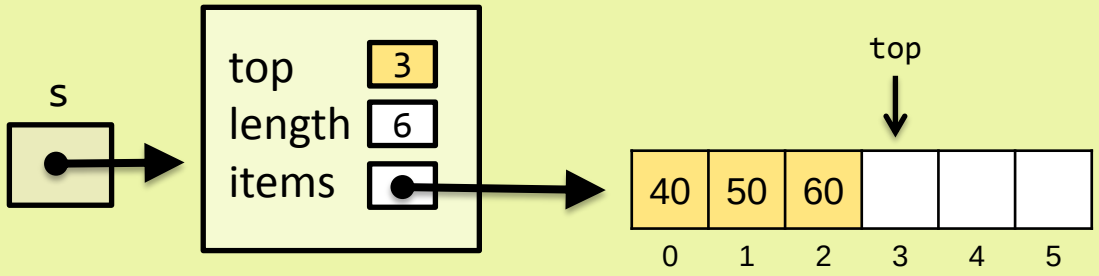
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

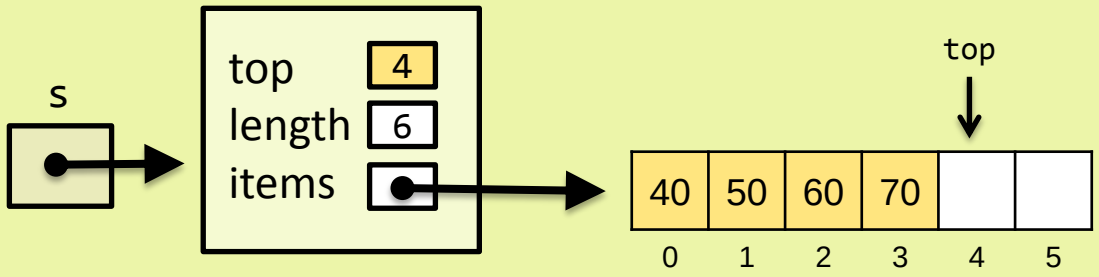
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

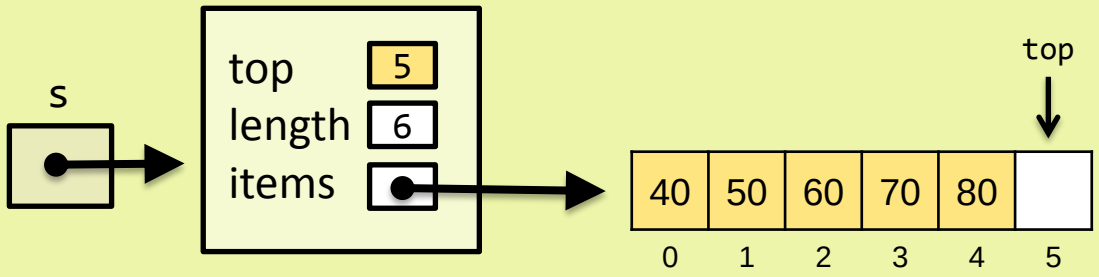
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

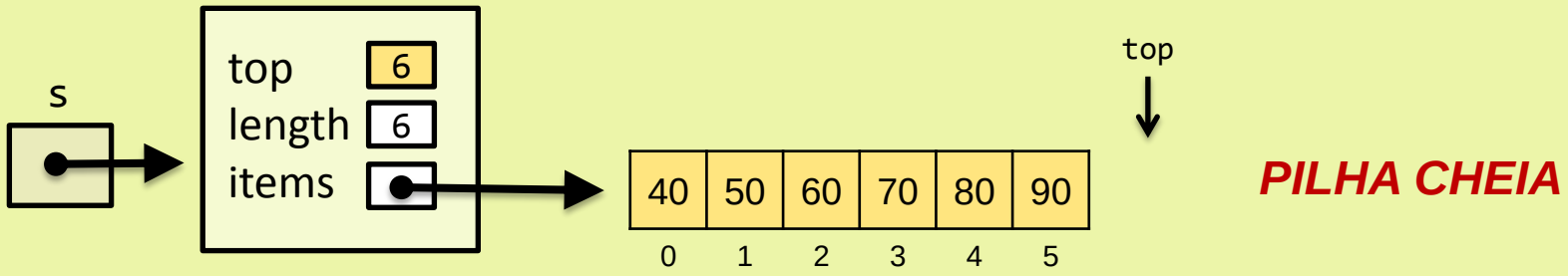
```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```



Simulação

```
Stack* s = createStack();
push(s,10);
push(s,20);
push(s,30);
ItemType removed;
pop(s, &removed);
pop(s, &removed);
pop(s, &removed);
```

```
push(s,40);
push(s,50);
push(s,60);
push(s,70);
push(s,80);
push(s,90);
```

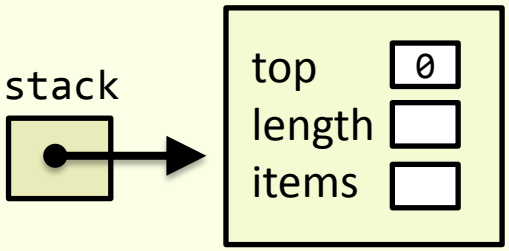


Implementação

Implementação

- A partir dessa simulação é possível extrair o comportamento das funções sobre os atributos da **Pilha Estática**

```
Stack* createStack();
void initializeStack(Stack* stack);
int push(Stack* stack, ItemType e);
int pop(Stack* stack, ItemType* e);
int top(Stack* stack, ItemType* e);
void printStack(Stack* stack);
int containsStack(Stack* stack, ItemType *e);
int sizeStack(Stack* stack);
int isEmptyStack(Stack* stack);
```




```
typedef struct{
    int top;
    int length;
    ItemType *items;
}Stack;
```

Implementação

LET'S DO IT