UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Pilha

Uma Lista Linear com restrições

Prof. Rafael Liberato
liberato@utfpr.edu.br

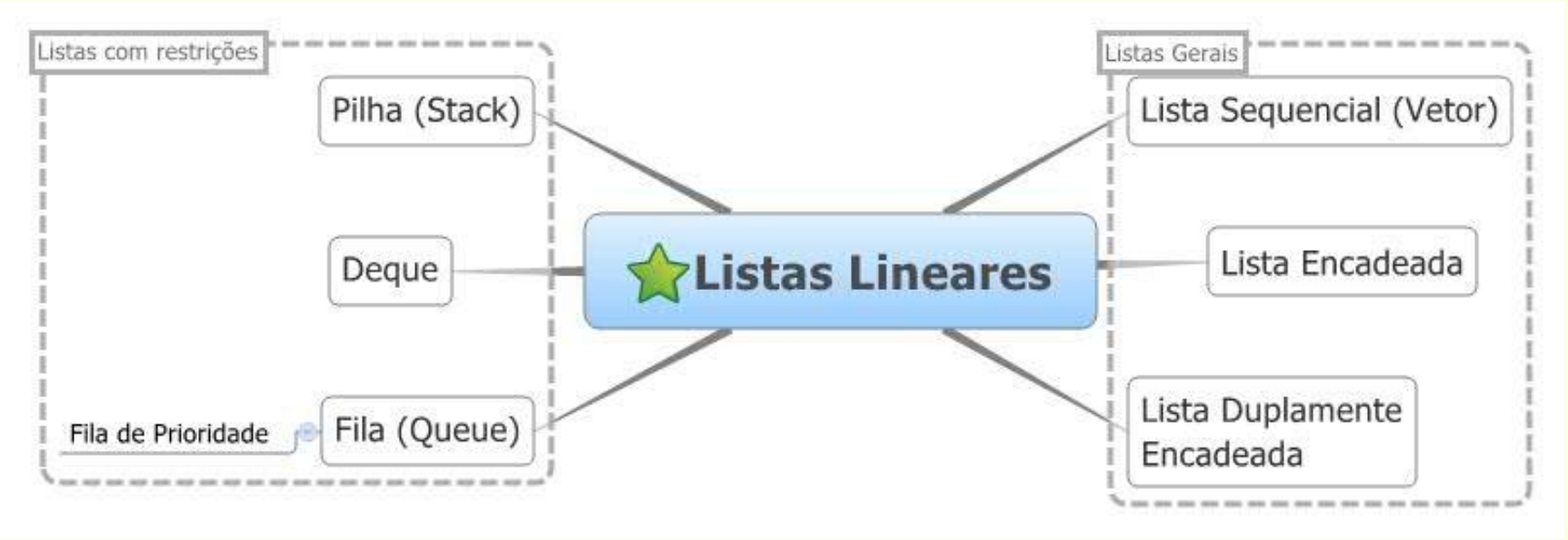
Objetivos

- Definir o TAD Pilha
  - Entender a estrutura de dados utilizada na Pilha, tanto estática quanto dinâmica.
  - Entender qual é a responsabilidade de cada operação, independente da estrutura utilizada.

Roteiro

- **Conceito**
- **Definindo uma Pilha**
- **TAD Pilha (Dados/Operações)**
  - Definindo a estrutura que armazenará os **dados**
  - Transformando **operações** em Funções
- **Descrição dos dados**
- **Descrição das operações da Pilha**

Relembrando...



Pilha - Conceito

Conceito

- **Pilha ou Stack é uma lista linear com restrições.**
  - A inserção e remoção da pilha é realizada em uma única extremidade denominada TOPO
  - O acesso também é restrito a esta extremidade.

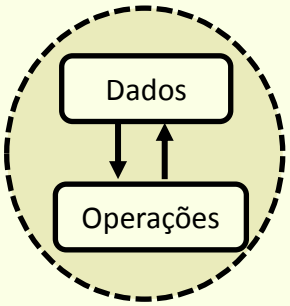
- **Também é conhecida como LIFO (Last In - First Out)**
  - O último que entra é o primeiro que sai

Aplicação

- **Aplicações**
  - Histórico de páginas de um navegador
  - Sequência de desfazer de um aplicativo
  - Cadeia de chamadas das funções em um programa
- **Aplicações Indiretas**
  - Estruturas de dados auxiliares para algoritmos
  - Componentes para outras estruturas de dados

Definindo uma PILHA

- **Agora que conhecemos um pouco mais sobre a Pilha, vamos especificar o que necessitamos que elas façam**
- **Para isso, precisamos:**
  1. definir o que vamos armazenar na Pilha
  2. definir quais operações serão realizadas na Pilha



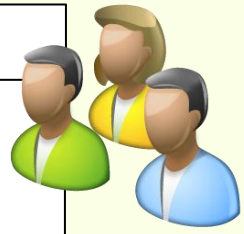
- **Quando levantamos essas informações, nós definimos o TAD**
  - **Neste momento, não vamos nos preocupar em COMO implementar, e sim em O QUE precisamos**

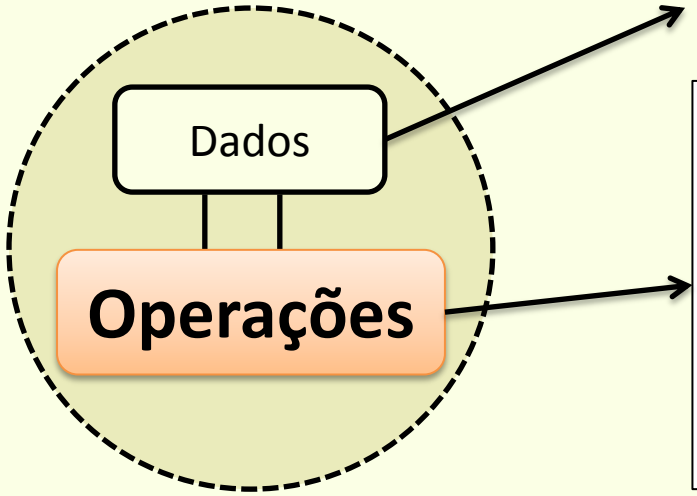
Definindo uma PILHA

- **Vamos especificar o Tipo Abstrato de Dados chamado Pilha para definir o que precisamos**

1. Quais dados serão manipulados?
2. Quais operações serão disponibilizadas para manipular os dados? O que precisamos?

Aluno
- ra: int - nome: String - email: String - notas: float[ ]



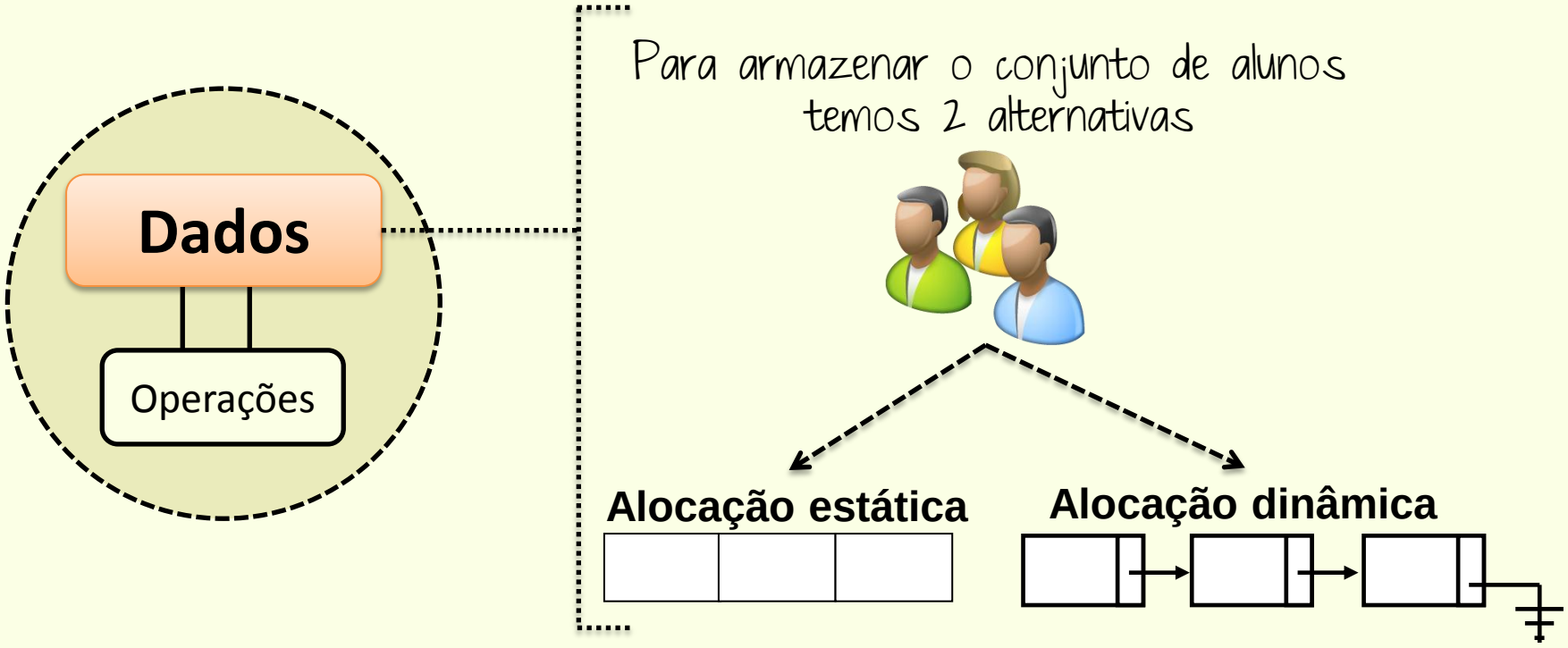


Conjunto de Alunos

- Criar a pilha
- Inserir um aluno na pilha;
- Remover um aluno da pilha;
- Recuperar informações do aluno que está no topo da pilha;
- Verificar se um aluno está na pilha;
- Verificar quantos alunos existem na pilha;
- Verificar se a pilha está vazia;
- Ver todo o conteúdo da pilha.

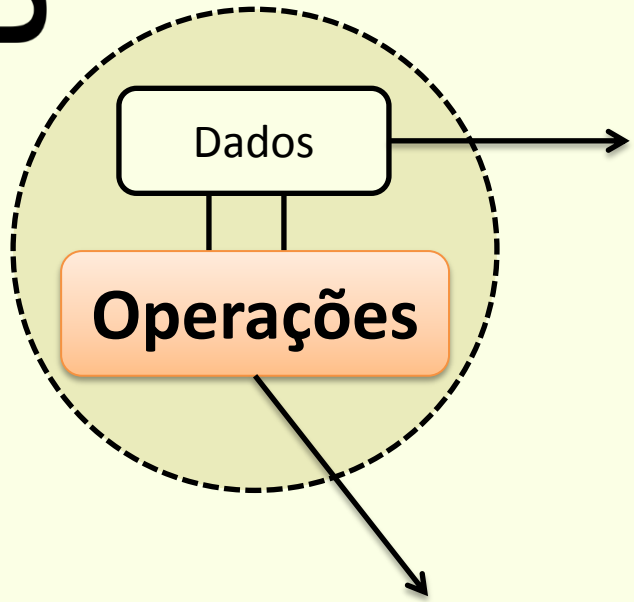
TAD Pilha

- **E quanto a estrutura que armazenará os dados?**
  - Ela será definida quando o TAD for materializado (implementado) por uma Estrutura de Dados



TAD Pilha

TAD Pilha



Stack será o nome da `struct` que organizará os dados da Pilha, independentemente da estratégia de alocação de memória utilizada.

```
Stack *createStack();                          // Criar a pilha
int push(Stack* stack, ItemType e);            // Inserir um elemento na pilha;
void initializeStack(Stack *stack);            // Inicializa a pilha
int pop(Stack* stack, ItemType* e);            // Remover um elemento da pilha
int peek(Stack* stack, ItemType* e);           // Recuperar informações do topo da  pilha
int contains(Stack* stack, ItemType *e)        // Verificar se um elemento  está na pilha
int sizeStack(Stack* stack);                   // Verificar quantos elementos  existem na pilha;
int isEmptyStack(Stack* stack);                // Verificar se a pilha está vazia
void printStack(Stack* stack);                 // Ver todo o conteúdo da pilha.
```

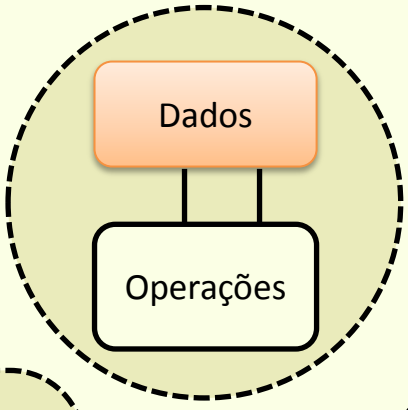
Descrição dos dados

Estrutura de dados PILHA

- Baseado nessas informações, vamos implementar a Estrutura de dados **Pilha** nas versões estática e dinâmica

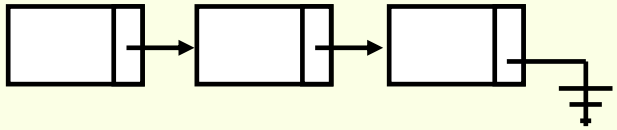
Pilha Estática

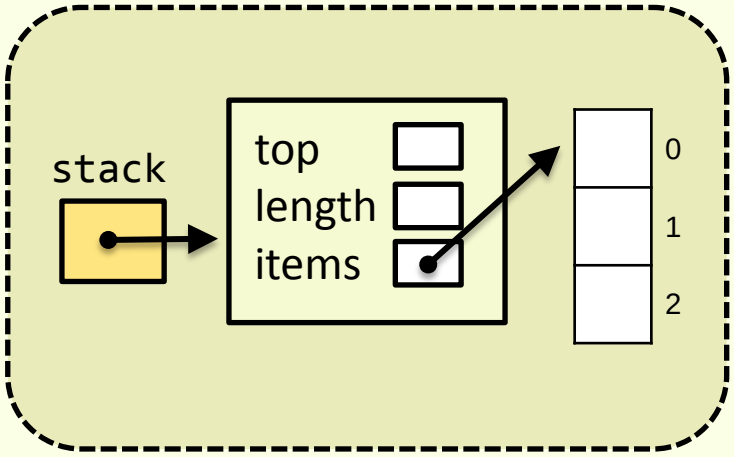
Utilizando alocação estática

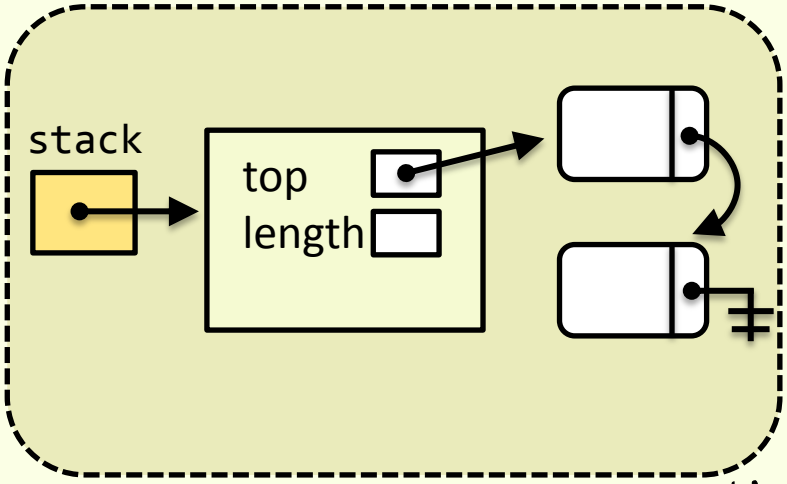


Pilha Dinâmica

Utilizando alocação dinâmica

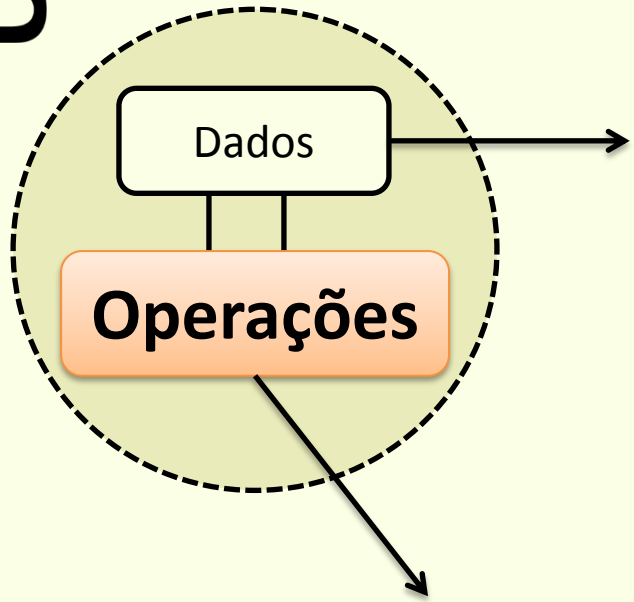






Descrição das operações

TAD Pilha



Stack será o nome da `struct` que organizará os dados da Pilha, independentemente da estratégia de alocação de memória utilizada.

```
Stack *createStack();                          // Criar a pilha
int push(Stack* stack, ItemType e);            // Inserir um elemento na pilha;
void initializeStack(Stack *stack);            // Inicializa a pilha
int pop(Stack* stack, ItemType* e);            // Remover um elemento da pilha
int peek(Stack* stack, ItemType* e);           // Recuperar informações do topo da  pilha
int contains(Stack* stack, ItemType *e)        // Verificar se um elemento  está na pilha
int sizeStack(Stack* stack);                   // Verificar quantos elementos  existem na pilha;
int isEmptyStack(Stack* stack);                // Verificar se a pilha está vazia
void printStack(Stack* stack);                 // Ver todo o conteúdo da pilha.
```

Descrição das Operações

```
Stack*createStack();
```

Aloca dinamicamente uma Pilha, inicializa-a e retorna seu endereço.

Parâmetros:

Nenhum

Retorno:

O endereço de memória da Pilha criada e inicializada.

Descrição das Operações

```
void initializeStack(Stack *s);
```

Inicializa uma especificada Pilha.

Parâmetros:

- **`Stack *s`**: endereco da Pilha a ser inicializada

Retorno:

nenhum

Descrição das Operações

```
int push(Stack* s, ItemType e);
```

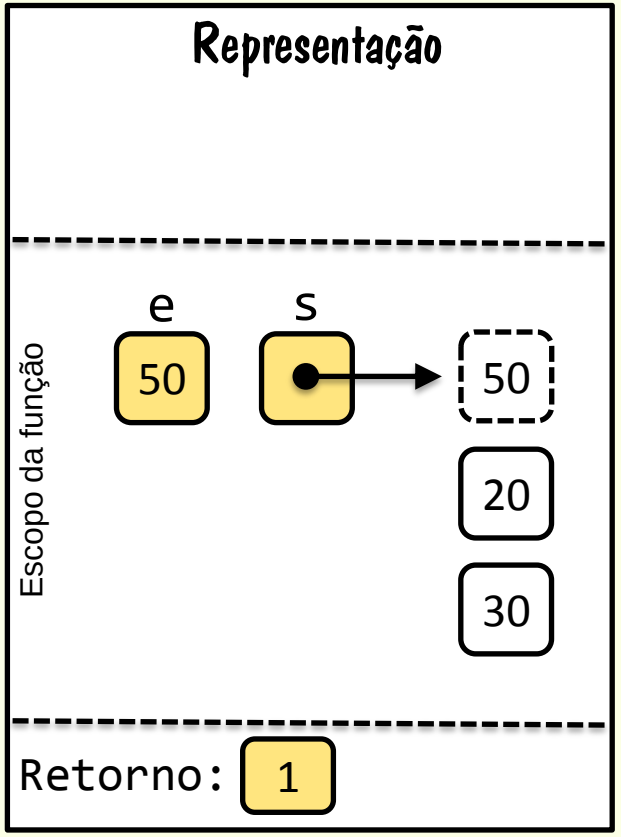
Insere o especificado elemento na Pilha.

Parâmetros:

- **Stack *s:** endereço da Pilha a ser manipulada.
- **ItemType e:** elemento a ser inserido.

Retorno:

- **true**: inserção realizada
- **false**: inserção **não** realizada



Descrição das Operações

```
int pop(Stack* s, ItemType* e);
```

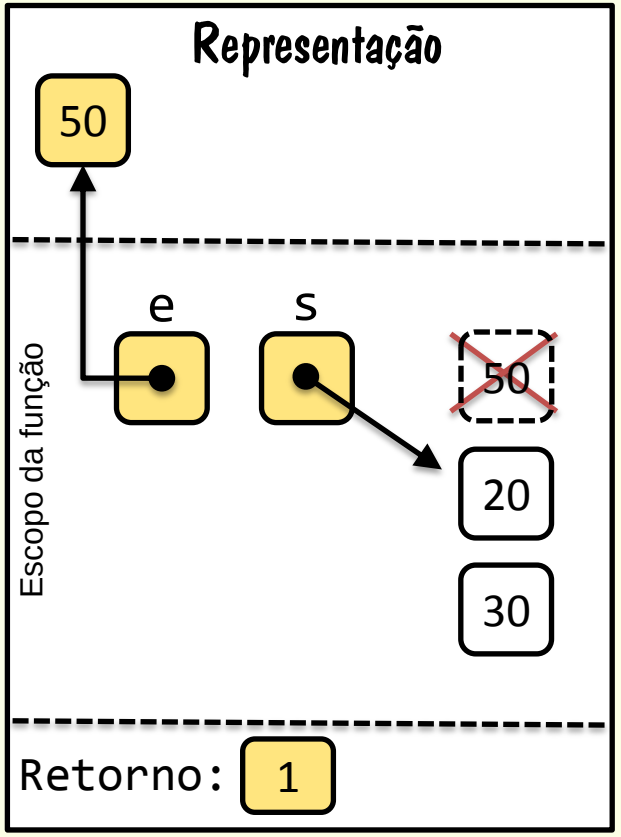
Insere o especificado elemento na Pilha.

Parâmetros:

- **Stack *s:** endereço da Pilha a ser manipulada.
- **ItemType e:** elemento a ser inserido.

Retorno:

- **true**: inserção realizada
- **false**: inserção **não** realizada



Descrição das Operações

```
int top(Stack* stack, ItemType* e);
```

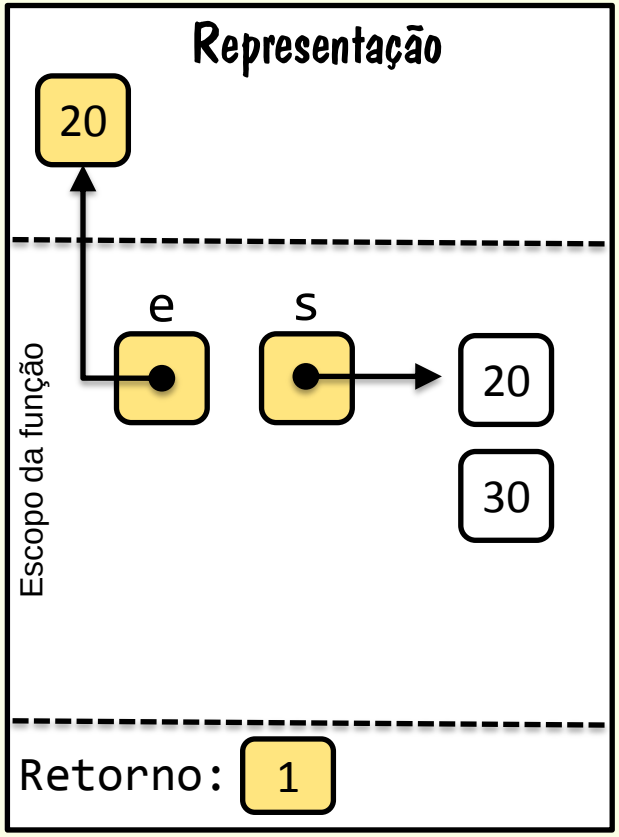
Insere o especificado elemento na Pilha.

Parâmetros:

- **Stack *s:** endereço da Pilha a ser manipulada
- **ItemType *e:** endereço utilizado armazenar o elemento do topo da Pilha.

Retorno:

- **true:** o elemento foi armazenado com sucesso.
- **false:** o elemento **não** foi recuperado.



Descrição das Operações

```
int containsStack(Stack* stack, ItemType *e);
```

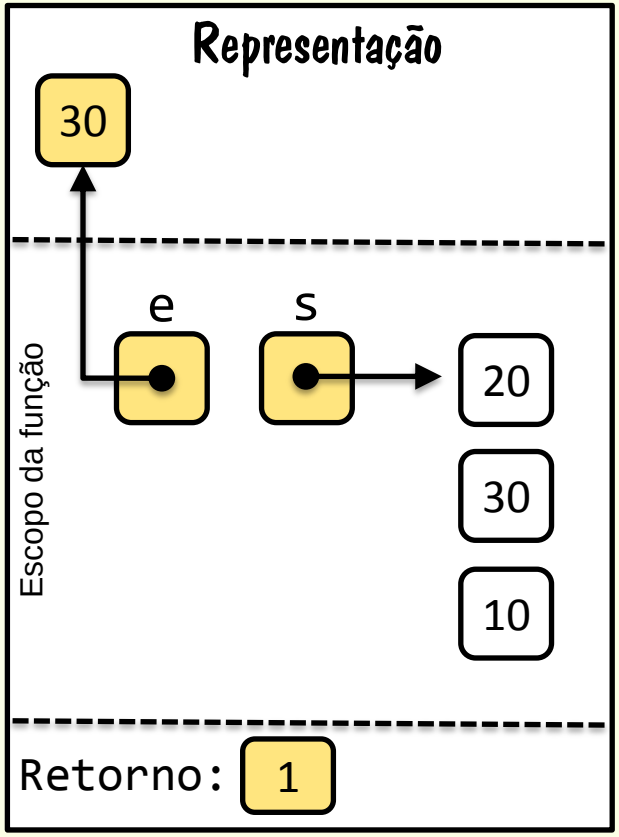
Verifica se o especificado elemento está contido na Pilha.

Parâmetros:

- **`Stack *s:`** endereço da Pilha a ser manipulada
- **`ItemType *e:`** endereço que contém o elemento desejado.

Retorno:

- **`true:`** o elemento foi armazenado com sucesso.
- **`false:`** o elemento **não** foi recuperado.



Descrição das Operações

```
int sizeStack(Stack* stack);
```

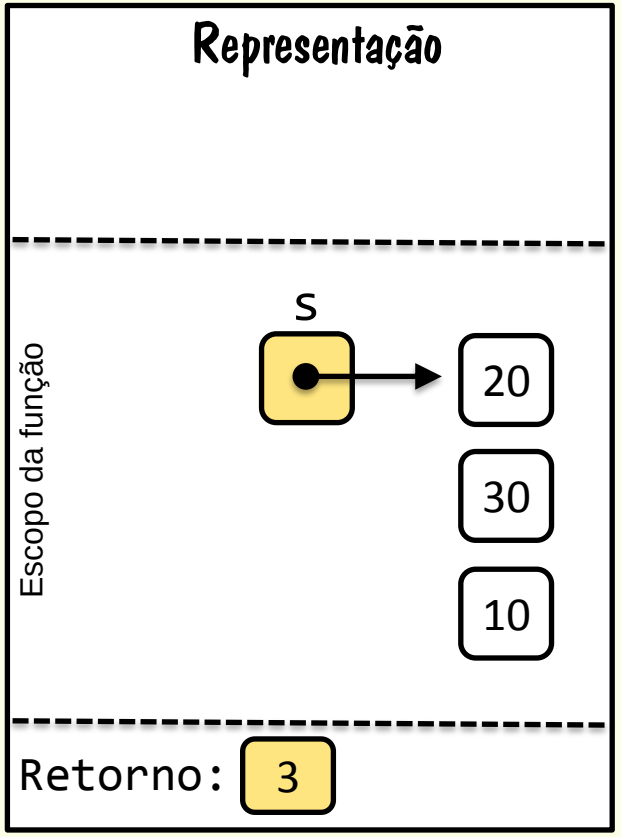
Retorna a quantidade de elementos da Pilha

Parâmetros:

- **Stack *s:** endereço da Pilha a ser manipulada

Retorno:

- **Número de elementos da Pilha**



Descrição das Operações

```
int isEmptyStack(Stack* stack);
```

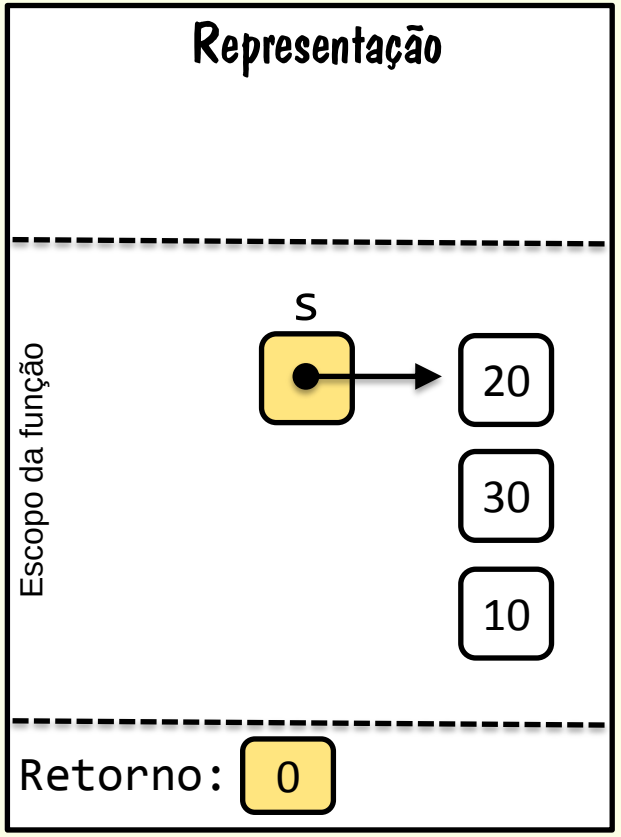
Verifica se a Pilha está vazia.

Parâmetros:

- **Stack *q:** endereço da Pilha a ser manipulada.

Retorno:

- **true:** a Pilha está vazia
- **false:** a Pilha **não** está vazia.



Descrição das Operações

```
void printStack(Stack* stack);
```

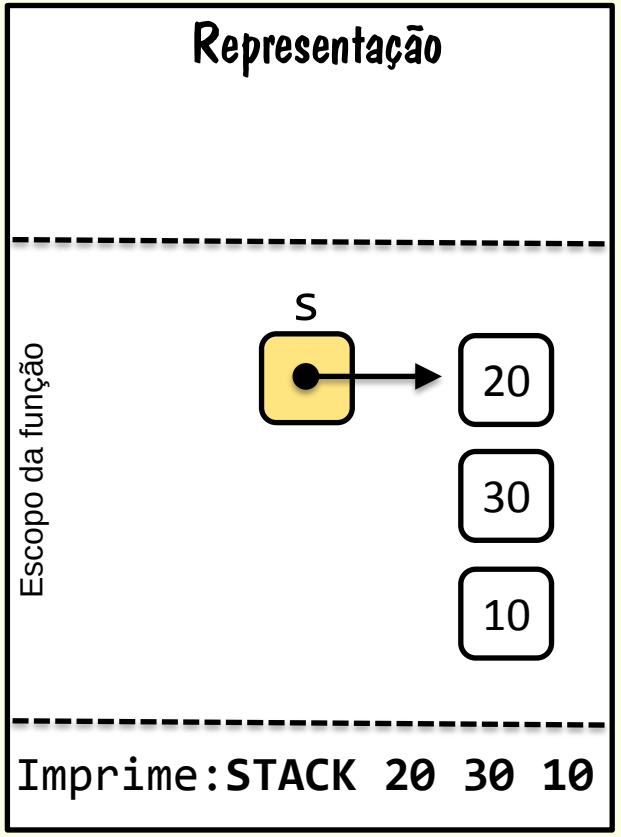
Verifica se a Pilha está vazia.

Parâmetros:

- **Stack *q:** endereço da Pilha a ser manipulada.

Retorno:

nenhum.



Referências

- http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html