UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Lista

Uma Lista Linear SEM restrições

Prof. Rafael Liberato
liberato@utfpr.edu.br

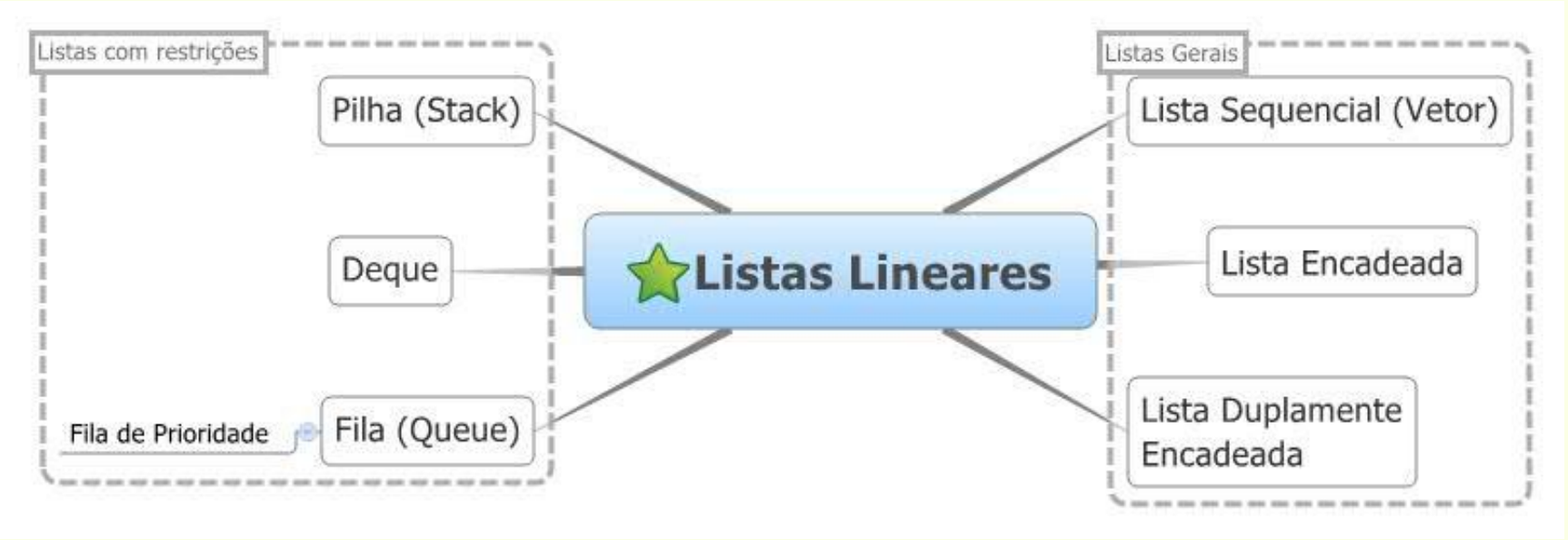
Objetivos

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

Roteiro

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

Relembrando...



Lista - Conceito

Conceito

- **A estrutura Lista é uma lista linear sem restrições.**
  - As inserções e remoções podem ser feitas em qualquer lugar

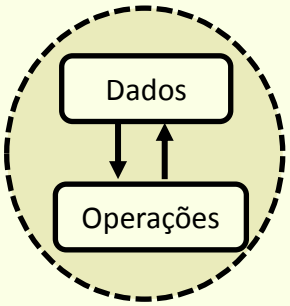
- **Não existe uma política para manipulação dos elementos**

Definindo uma LISTA

- **Agora que conhecemos um pouco mais sobre a LISTA, vamos especificar o que necessitamos que elas façam**

- **Para isso, precisamos:**

  1. definir o que vamos armazenar na Lista
  2. definir quais operações serão realizadas na Lista



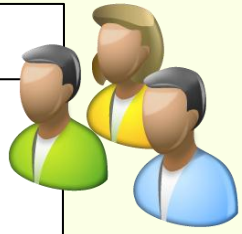
- **Quando levantamos essas informações, nós definimos o TAD**
  - **Neste momento, não vamos nos preocupar em COMO implementar, e sim em O QUE precisamos**

Definindo uma LISTA

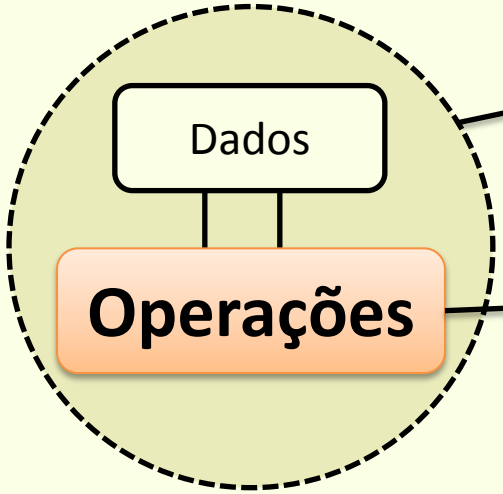
- **Vamos especificar o Tipo Abstrato de Dados chamado Lista para definir o que precisamos**

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

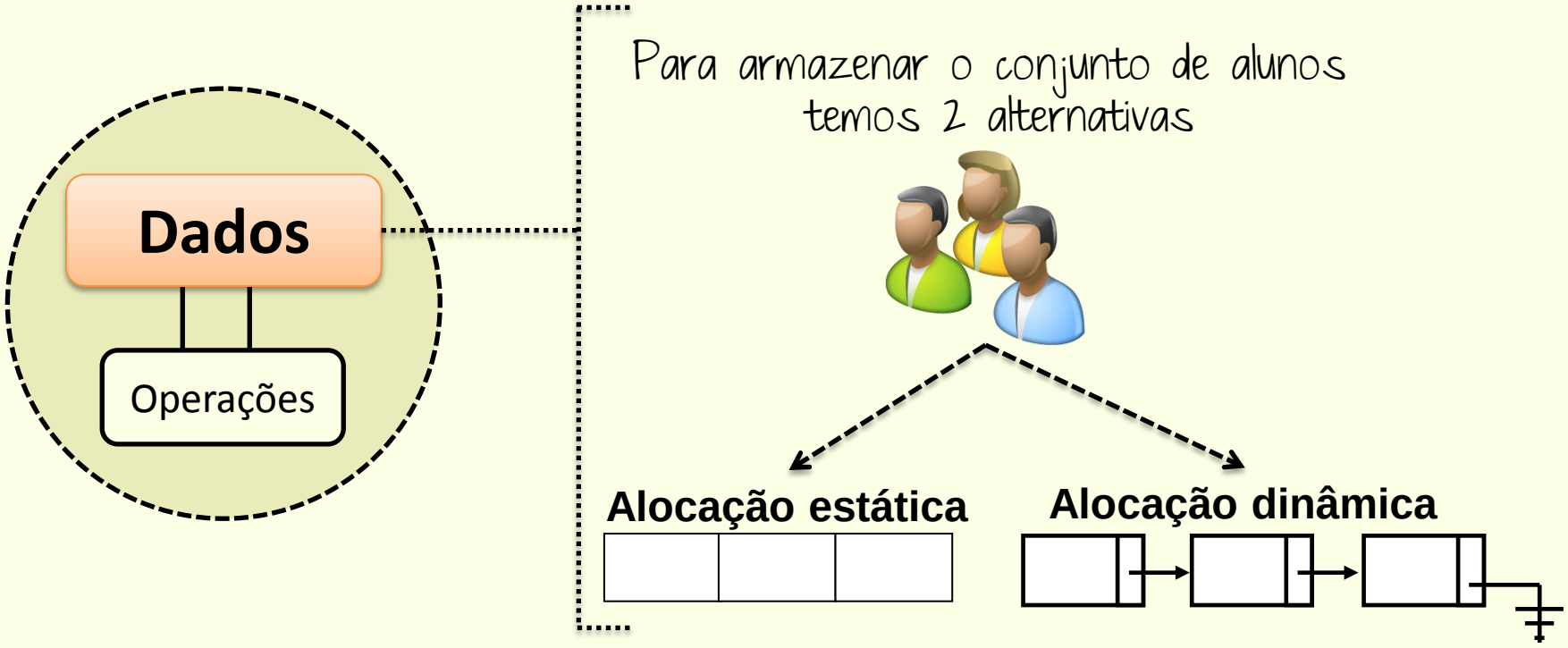


- Criar a LISTA
- Inserir um elemento no final da LISTA;
- Inserir um elemento em qualquer posição da LISTA;
- Remover um elemento de qualquer posição da LISTA;
- Recuperar qualquer elemento da LISTA;
- Alterar qualquer elemento da LISTA;
- Verificar se um elemento está na LISTA;
- Verificar a posição de um determinado elemento LISTA;
- Verificar quantos elementos existem na LISTA;
- Verificar se a LISTA está vazia;
- Ver todo o conteúdo da LISTA.

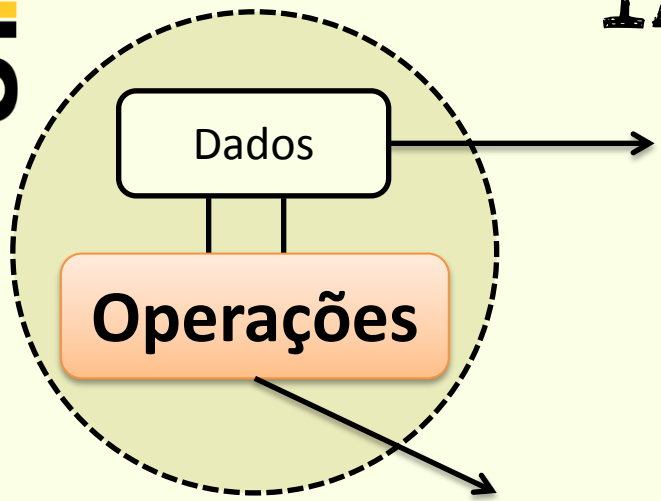
TAD Lista

TAD LISTA

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



TAD LISTA



List será o nome da **struct** que organizará os dados da Lista, independentemente da estratégia de alocação de memória utilizada.

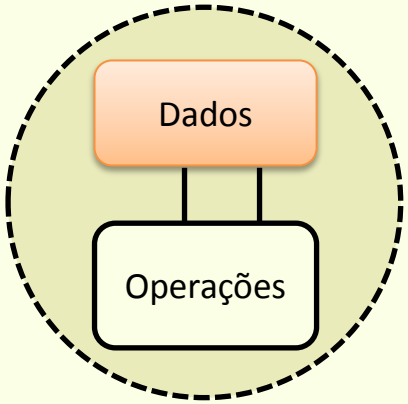
```
List *createList ();
void initializeList(List *l);
int addLastList(List *l, ItemType e);
int addList(List* l, ItemType e, int index);
int removeList(List* l, int index, ItemType *e);
int removeElementList(List* l, ItemType* e);
int getList(List* l, int index, ItemType* e);
int setList(List* l, int index, ItemType* e);
int indexOfList(List* l, ItemType* e);
int containsList(List* l, ItemType *e);
int sizeList(List* l);
int isEmptyList(List* l);
void printList(List* l);
```

Descrição dos dados

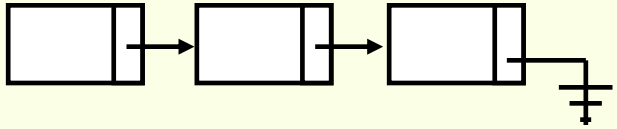
Estrutura de dados LISTA

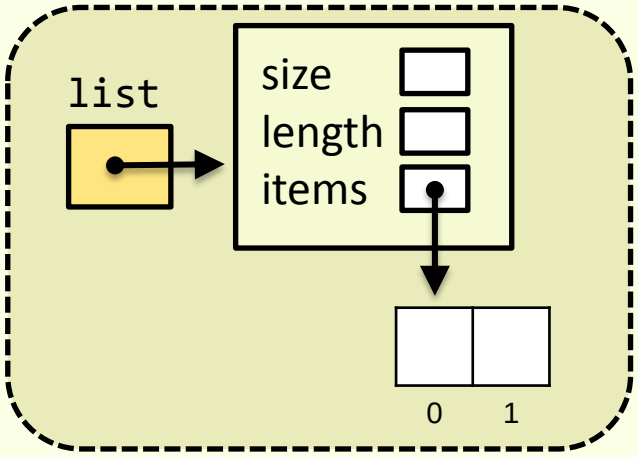
- Baseado nessas informações, vamos implementar a Estrutura de dados **Lista** nas versões estática e dinâmica

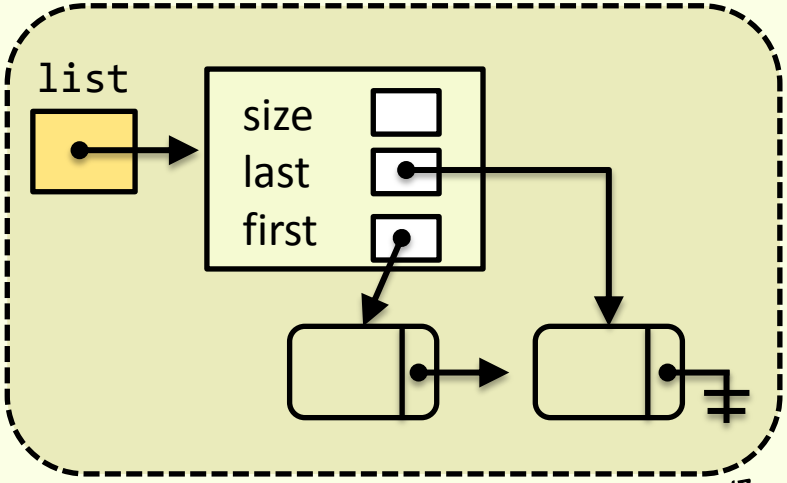
Lista Estática
Utilizando alocação estática



Lista Dinâmica
Utilizando alocação dinâmica







Descrição das operações

Descrição das Operações

```
List *createList();
```

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

Parâmetros:

Nenhum

Retorno:

O endereço de memória da lista criada e inicializada.

Descrição das Operações

```
void initializeList(List *l);
```

Inicializa uma especificada lista.

Parâmetros:

- **`List *l`**: endereco da lista a ser inicializada

Retorno:

nenhum

Descrição das Operações

```
int addLastList(List *l, ItemType e);
```

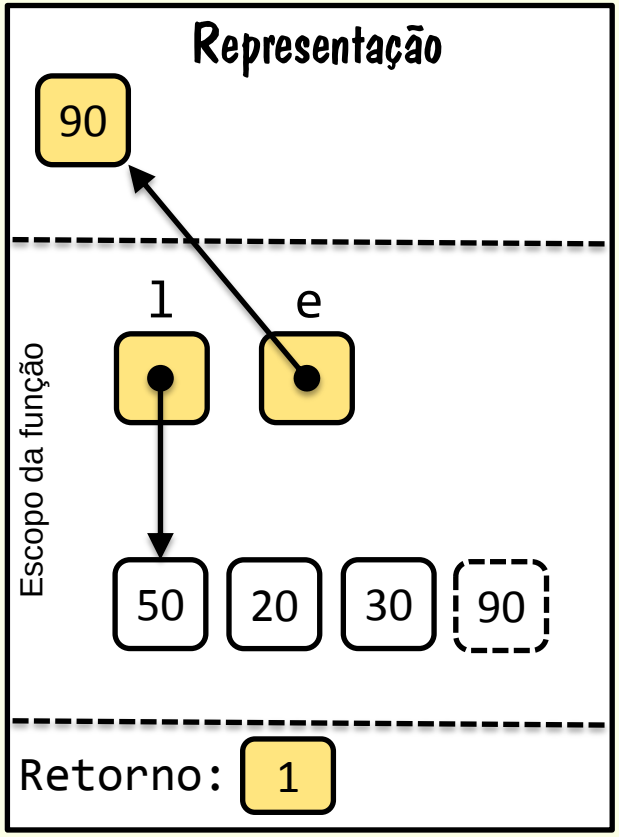
Insere o especificado elemento no FINAL da lista.

Parâmetros:

- **`List *l:`** endereço da lista a ser manipulada.
- **`ItemType e:`** elemento a ser inserido.

Retorno:

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



Descrição das Operações

```
int addList(List* l, ItemType e, int index);
```

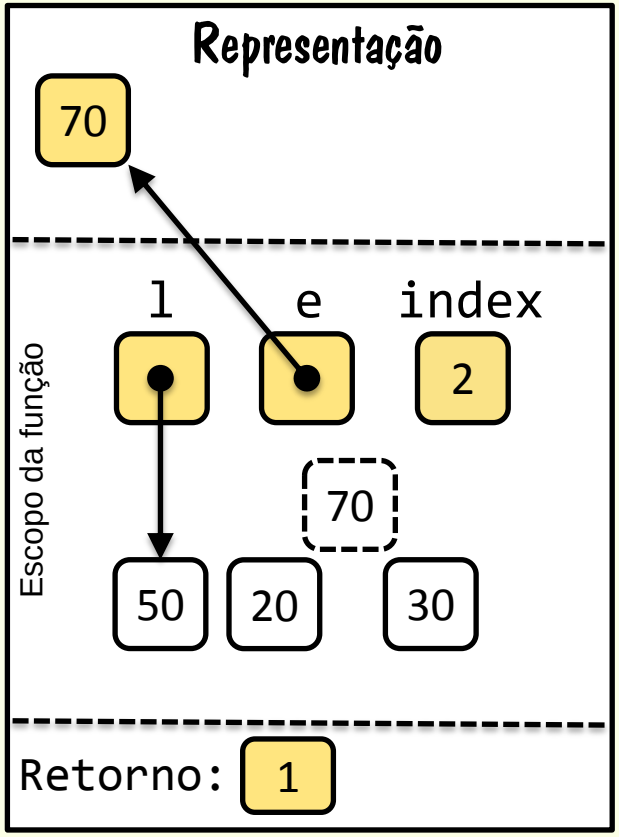
Insere o especificado elemento na especificada posição da lista.

Parâmetros:

- **`List *l:`** endereço da lista a ser manipulada.
- **`ItemType e:`** elemento a ser inserido.
- **`int index:`** posição a ser inserido o elemento

Retorno:

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



Descrição das Operações

```
int removeList(List* l, int index, ItemType *e);
```

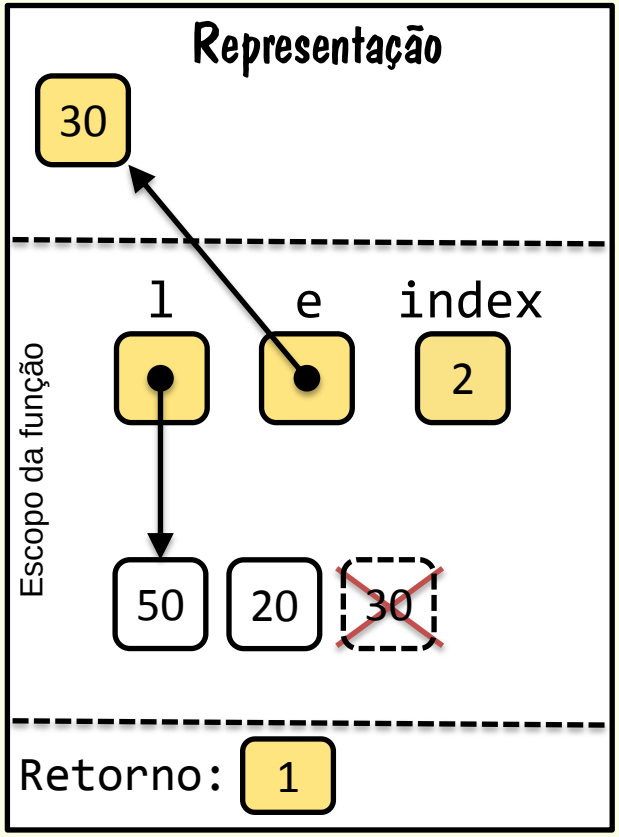
Remove o elemento da lista de uma especificada posição e armazena no endereço **e**.

Parâmetros:

- **List *l:** endereço da lista a ser manipulada
- **int index:** posição do elemento a ser removido
- **ItemType *e:** endereço utilizado para o armazenamento do elemento removido

Retorno:

- **true**: remoção realizada
- **false**: remoção **não** realizada



Descrição das Operações

```
int removeElementList(List* l, ItemType* e);
```

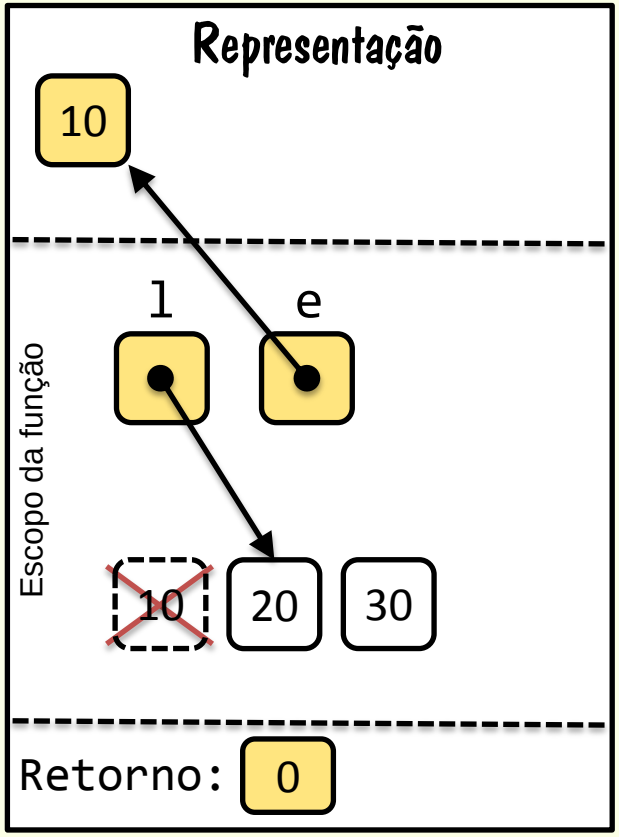
Remove um especificado elemento da lista.

Parâmetros:

- **List *l:** endereço da lista a ser manipulada
- **ItemType *e:** endereço que contem o valor do elemento a ser removido

Retorno:

- **Posição** do elemento removido
- **-1** quando o elemento não foi encontrado



Descrição das Operações

```
int getList(List* l, int index, ItemType* e);
```

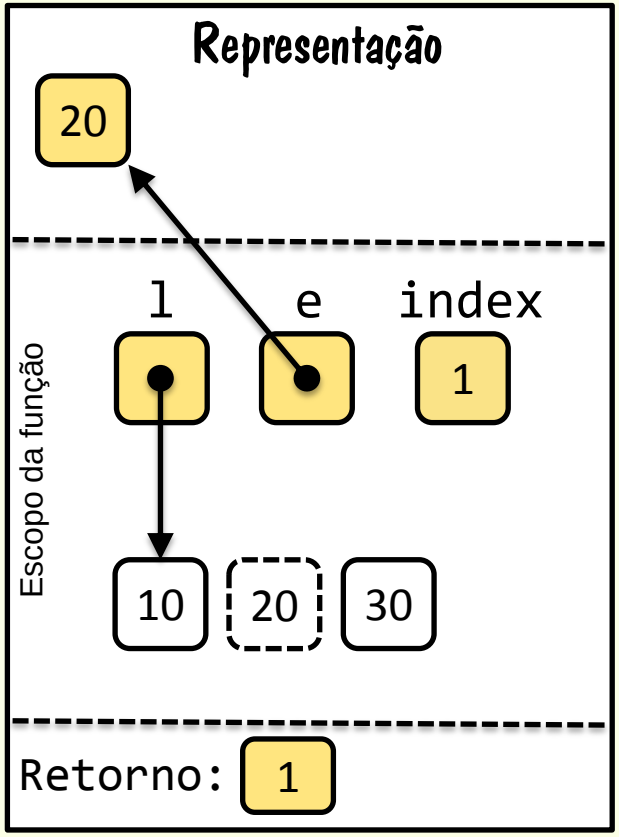
Recupera um elemento da lista de uma especificada posição.

Parâmetros:

- **List *l:** endereço da lista a ser manipulada.
- **int index:** posição do elemento a ser recuperado.
- **ItemType *e:** endereço utilizado para o armazenamento do elemento desejado

Retorno:

- **true:** o elemento foi encontrado e recuperado.
- **false:** o elemento não foi encontrado.



Descrição das Operações

```
int setList(List* l, int index, ItemType* e);
```

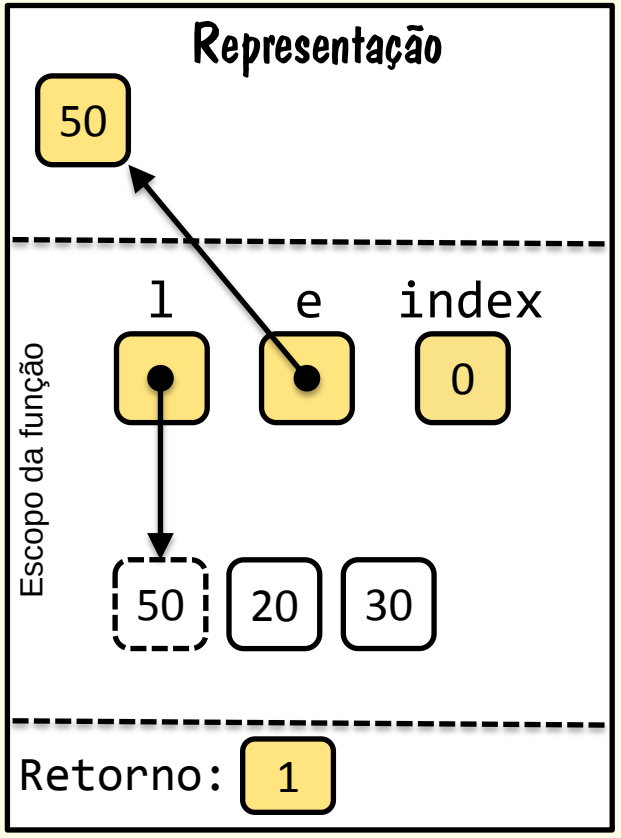
Substitui um elemento de uma especificada posição na lista.

Parâmetros:

- **List *l:** endereço da lista a ser manipulada.
- **int index:** posição do elemento a ser substituído.
- **ItemType *e:** endereço que contém o elemento a ser copiado

Retorno:

- **true:** se o elemento foi substituído
- **false:** se o elemento **não** foi substituído.



Descrição das Operações

```
int indexOfList(List* l, ItemType* e);
```

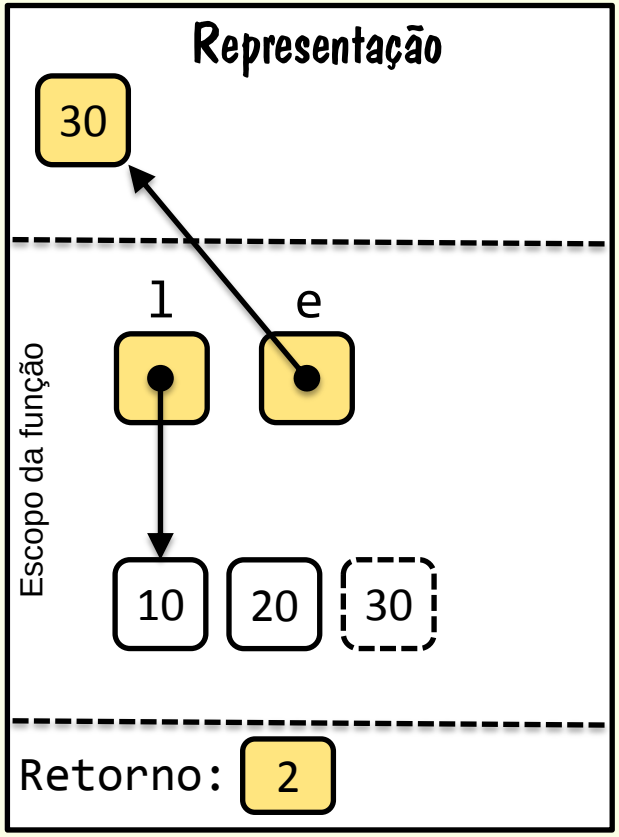
Procura o especificado elemento e retorna sua posição na lista

Parâmetros:

- **List *l:** endereço da lista a ser manipulada.
- **ItemType *e:** endereço que contém o elemento desejado

Retorno:

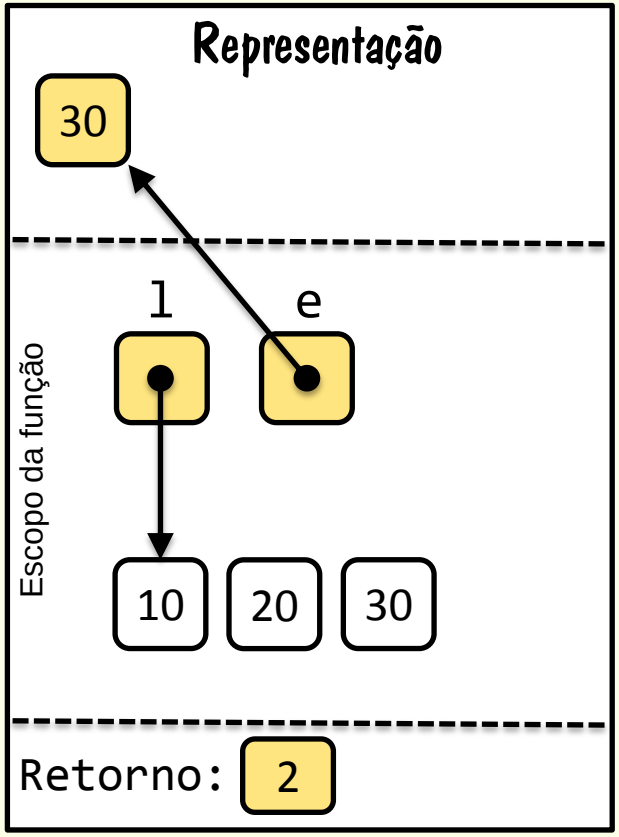
- **Posição** do elemento na lista
- **-1** quando o elemento não foi encontrado



Descrição das Operações

```
int containsList(List* l, ItemType *e);
```

Verifica se o especificado elemento esta contido na lista.



Parâmetros:

- **`List *l:`** endereço da lista a ser manipulada.
- **`ItemType *e:`** endereço que contém o elemento desejado

Retorno:

- **`true:`** se o elemento está contido na lista.
- **`false:`** se o elemento **não** está contido.

Descrição das Operações

```
int sizeList(List* l);
```

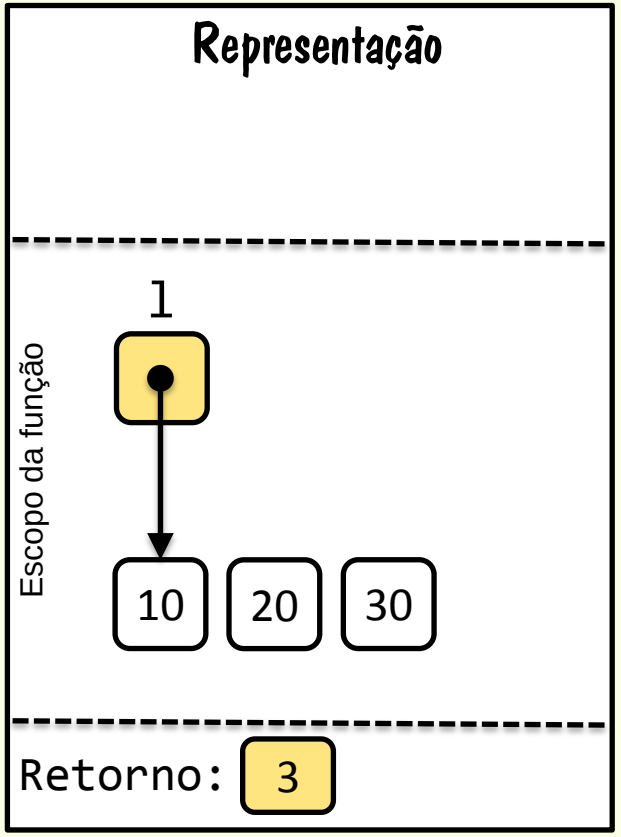
Retorna a quantidade de elementos da lista

Parâmetros:

- **List *l:** endereço da lista a ser manipulada.

Retorno:

- **Número de elementos da lista**



Descrição das Operações

```
int isEmptyList(List* l);
```

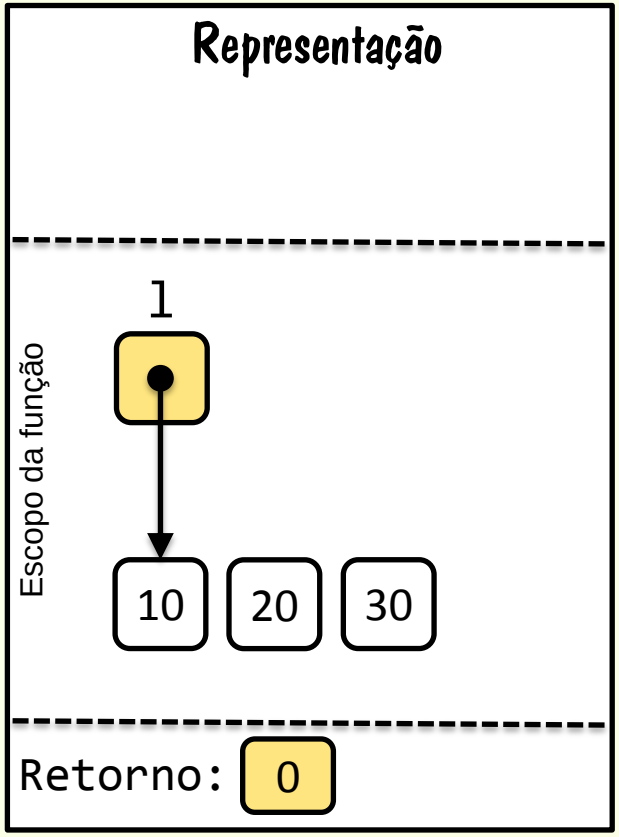
Verifica se a lista esta vazia.

Parâmetros:

- **List *l:** endereço da lista a ser manipulada.

Retorno:

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



Descrição das Operações

```
void printList(List* l);
```

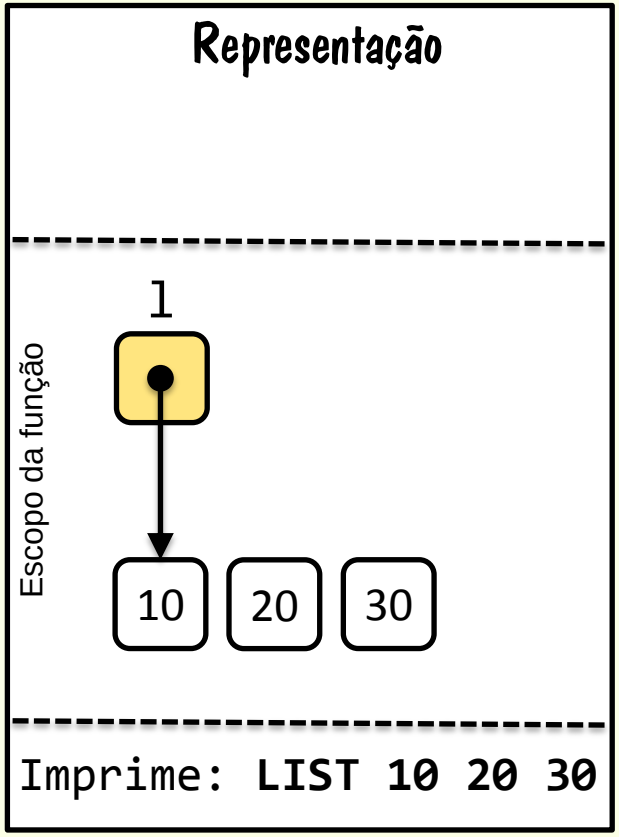
Imprime todos os elementos da lista.

Parâmetros:

- **List *l:** endereço da lista a ser manipulada.

Retorno:

nenhum



Referências

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