UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

# Lista Duplamente Encadeada

Prof. Rafael Liberato
liberato@utfpr.edu.br

# Objetivos

- Entender o funcionamento de uma Lista Duplamente Encadeada
- Compreender o porquê do encadeamento duplo
- Ser capaz de implementar as operações definidas no TAD Lista manipulando uma estrutura dinâmica de armazenamento com encadeamento duplo.

# Roteiro

- TAD Lista
- Lista Duplamente Encadeada
- Simulação
- Implementação

# TAD Lista

# TAD Lista

```
#define ItemType int

typedef struct{

}List;

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);
```

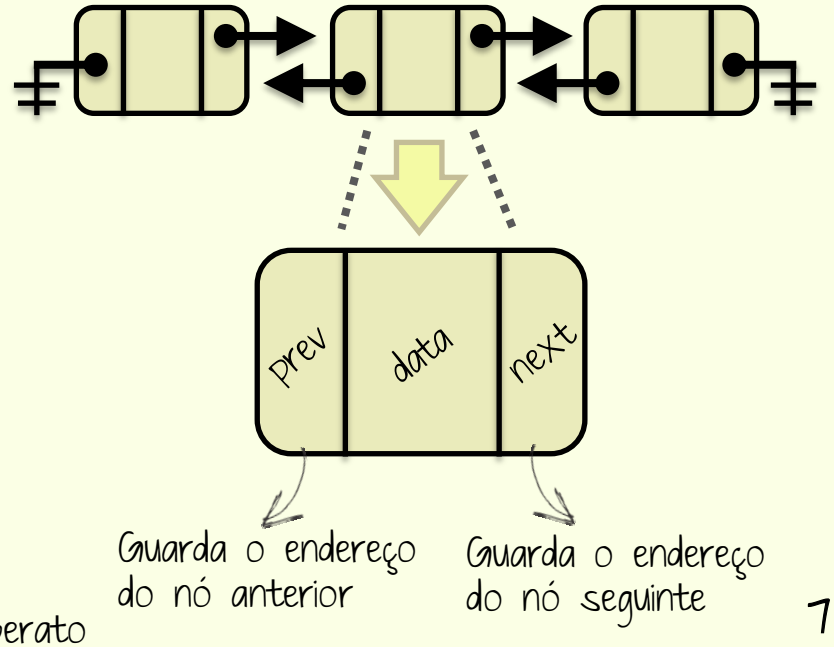
Vamos identificar os atributos que representarão a lista duplamente encadeada

# Encadeamento Duplo

# Encadeamento Duplo

- O encadeamento duplo da estrutura dinâmica permite o percurso em ambos os sentidos ⇄
  - → Essa característica faz com que alguns algoritmos fiquem mais eficientes. Por exemplo, a remoção do penúltimo elemento

```
typedef struct node{
   ItemType    data;
   struct node *prev;
   struct node *next;
}Node;
```

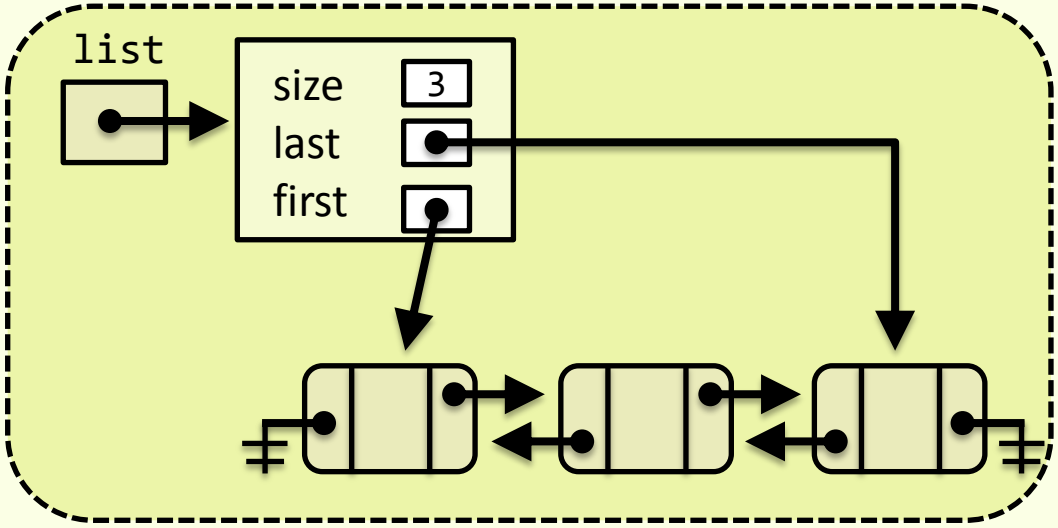


Estrutura utilizada para armazenar os dados

# Lista Duplamente Encadeada

# Lista Duplamente Encadeada

- A lista **duplamente encadeada** é idêntica a lista encadeada, salvo o duplo encadeamento dos seus nós.
- Os atributos são os mesmos




```
typedef struct node{
    ItemType     data;
    struct node  *prev
    struct node  *next;
}Node;
```

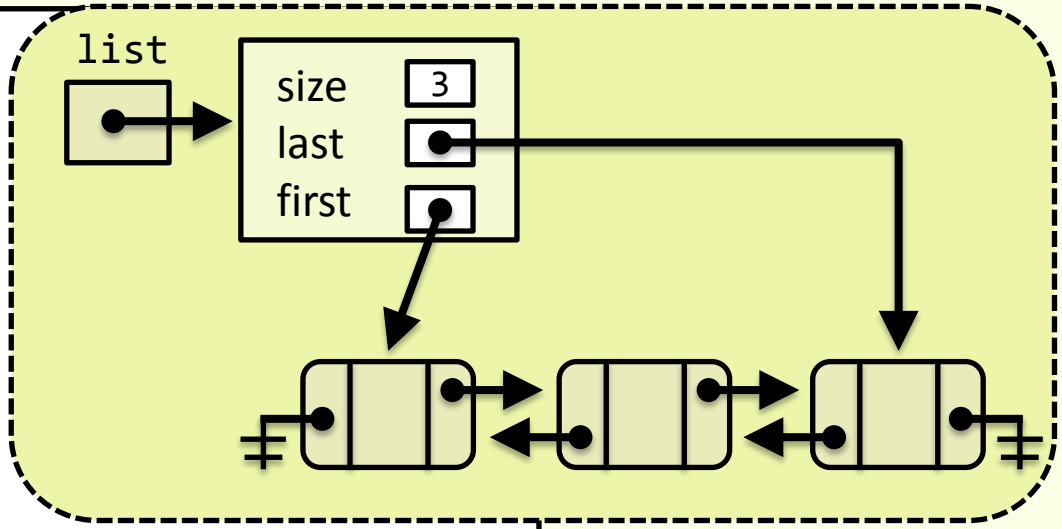
```
typedef struct{
    Node  *first;
    Node  *last;
    int   size;
}List;
```

# Lista Duplamente Encadeada

```
#define ItemType int

typedef struct{
    Node *first;
    Node *last;
    int size;
}List;

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




```
typedef struct node{
    ItemType     data;
    struct node  *prev
    struct node  *next;
}Node;
```

# Simulação

# Simulação

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

# Simulação

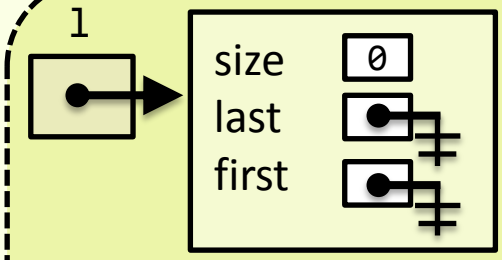
```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```

# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

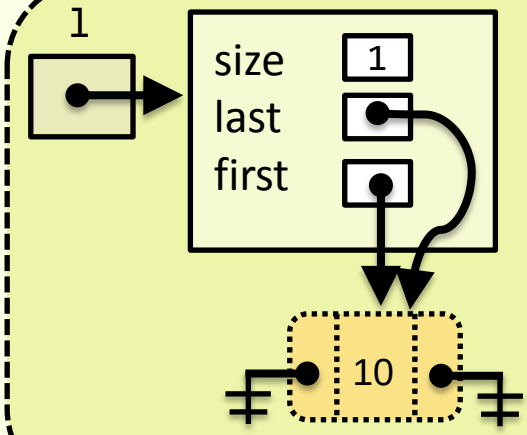
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

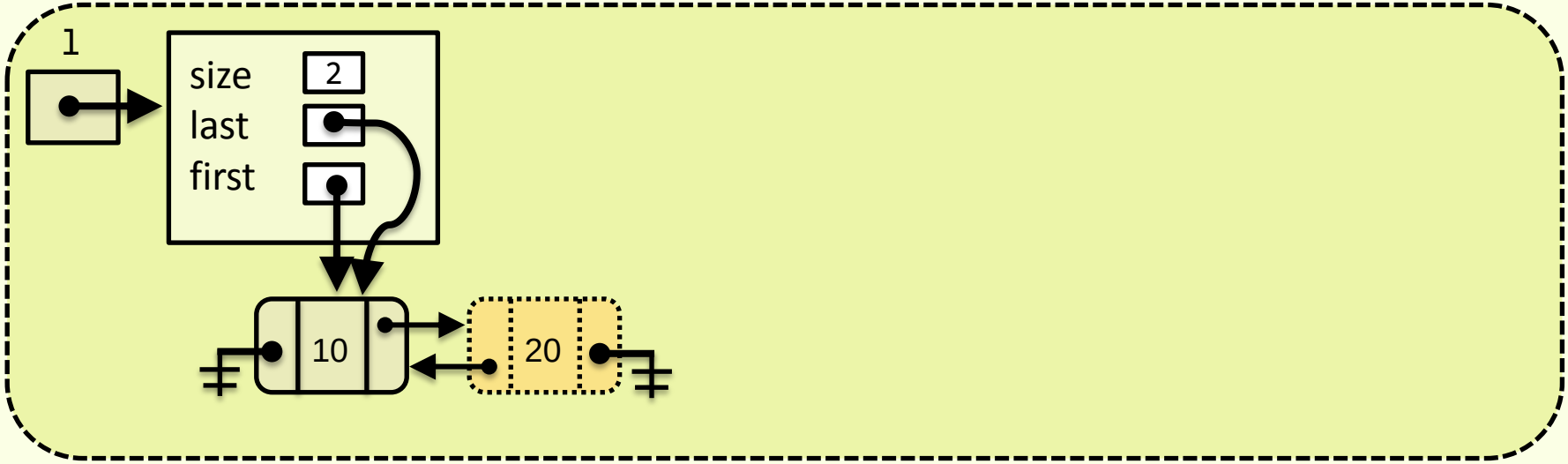
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

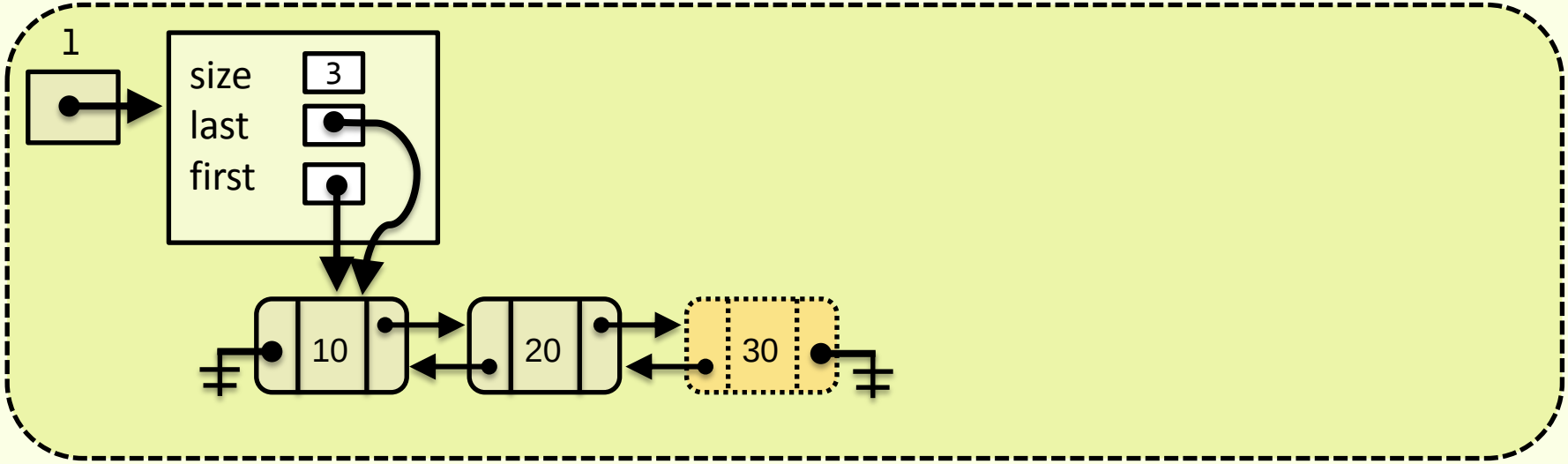
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

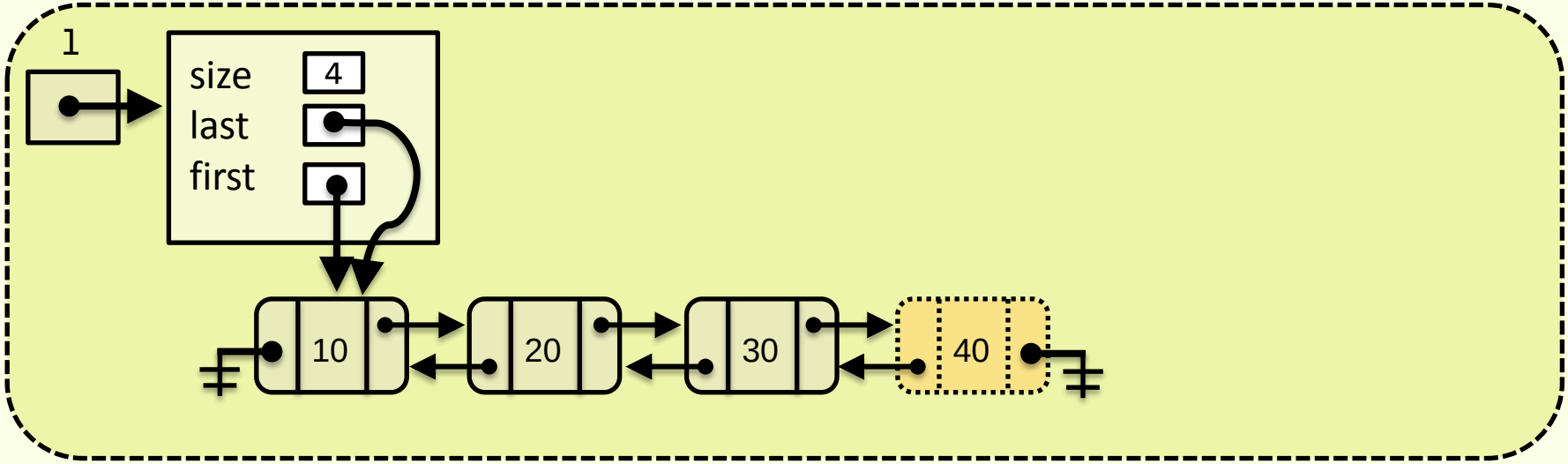
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

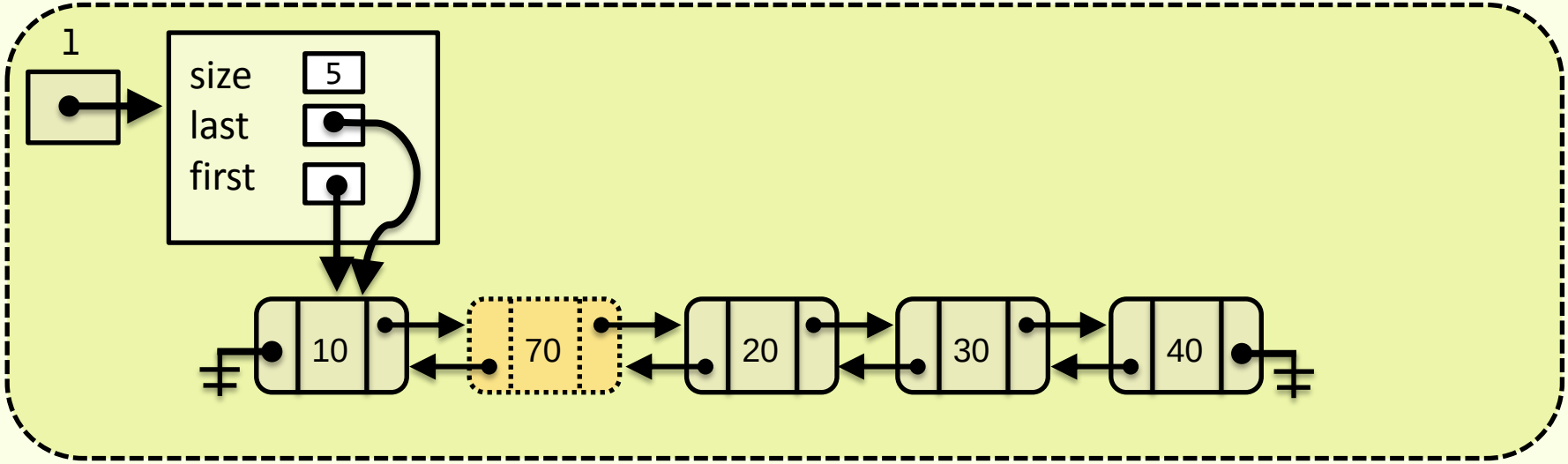
```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```

# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

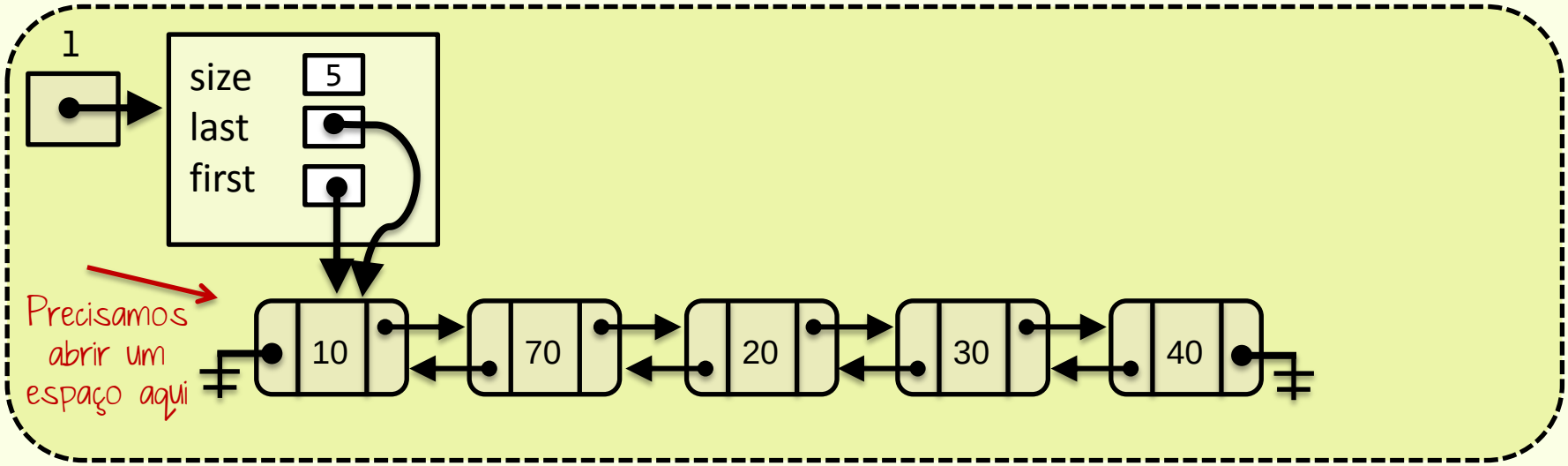
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

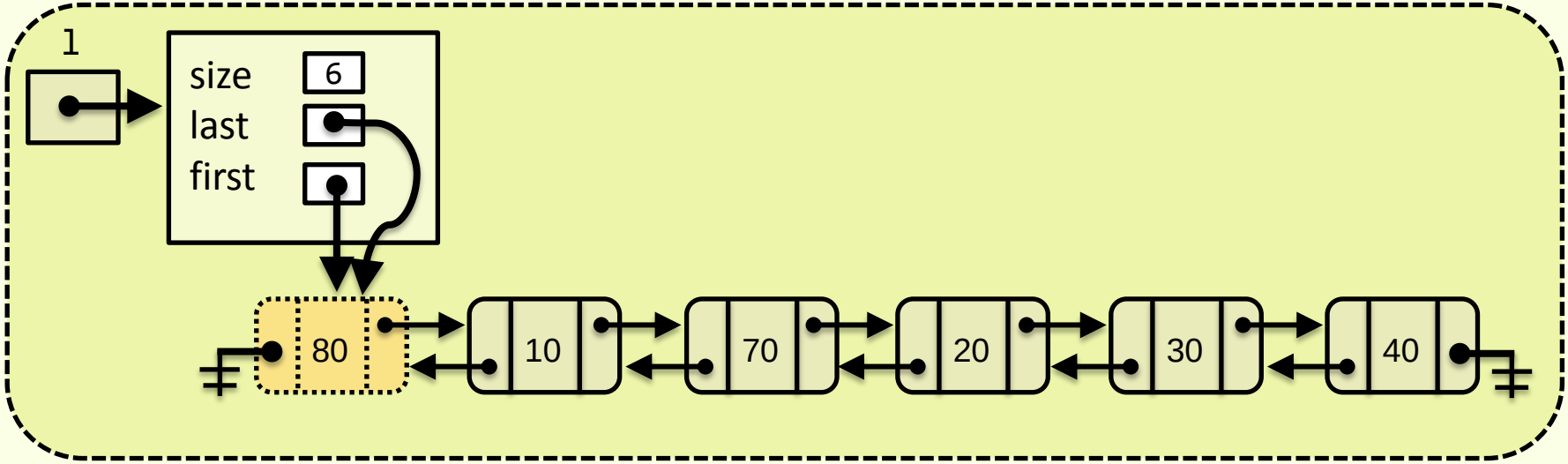
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

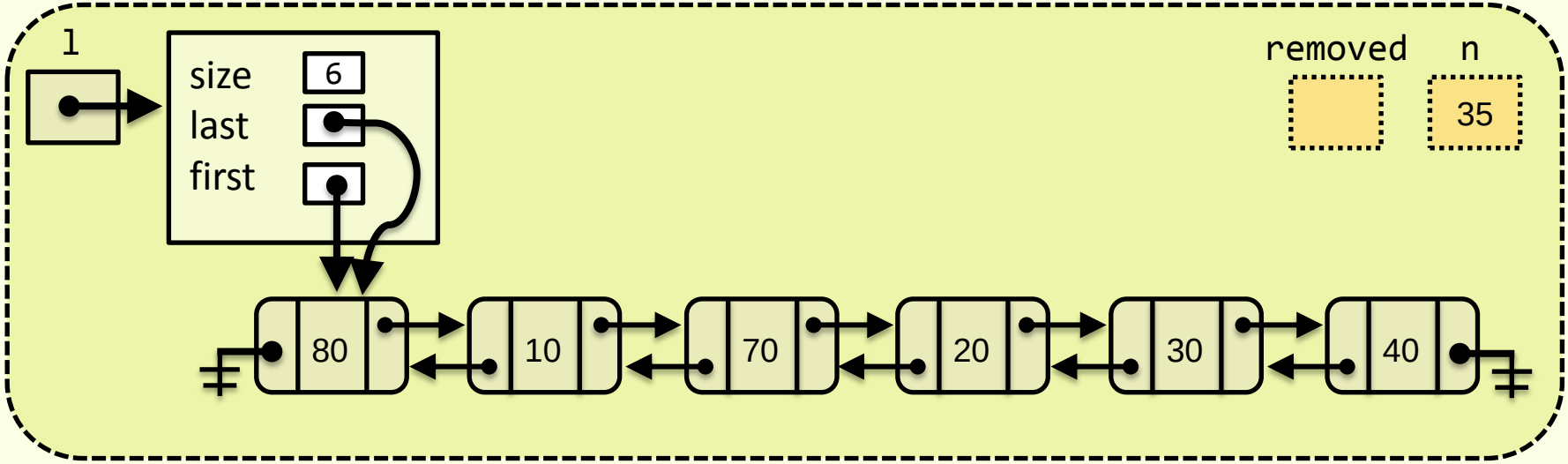
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

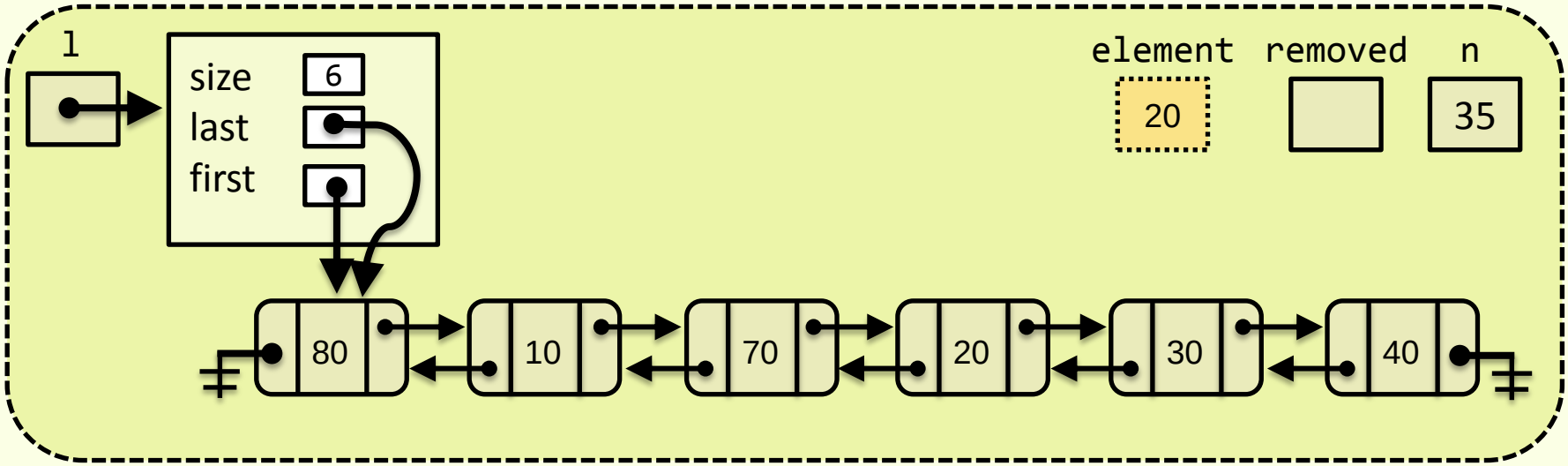
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

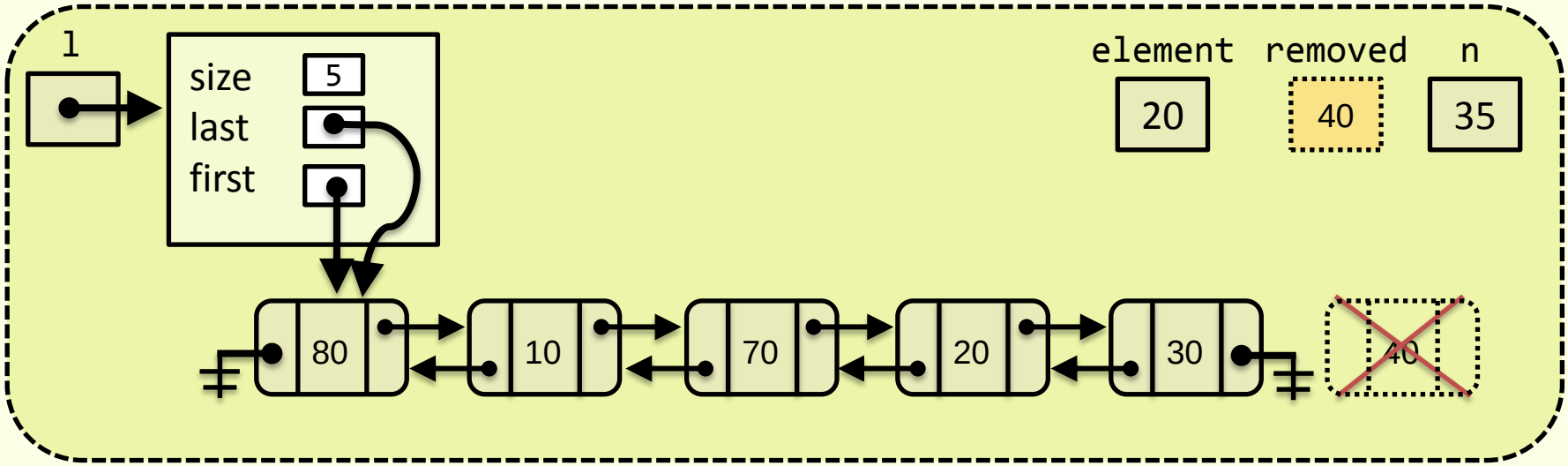
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

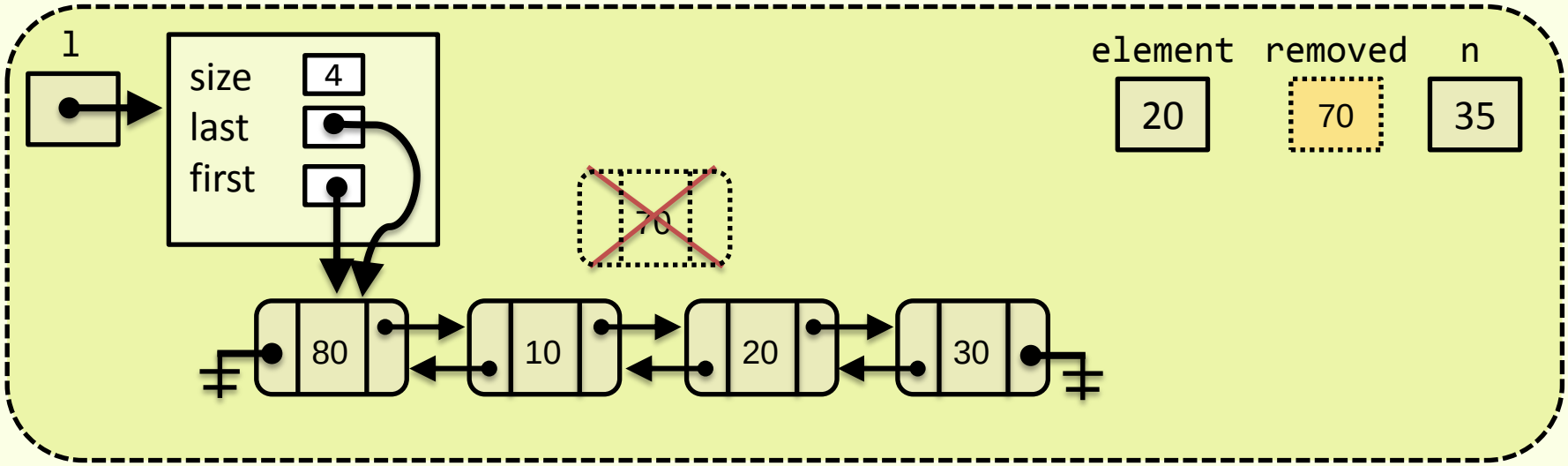
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

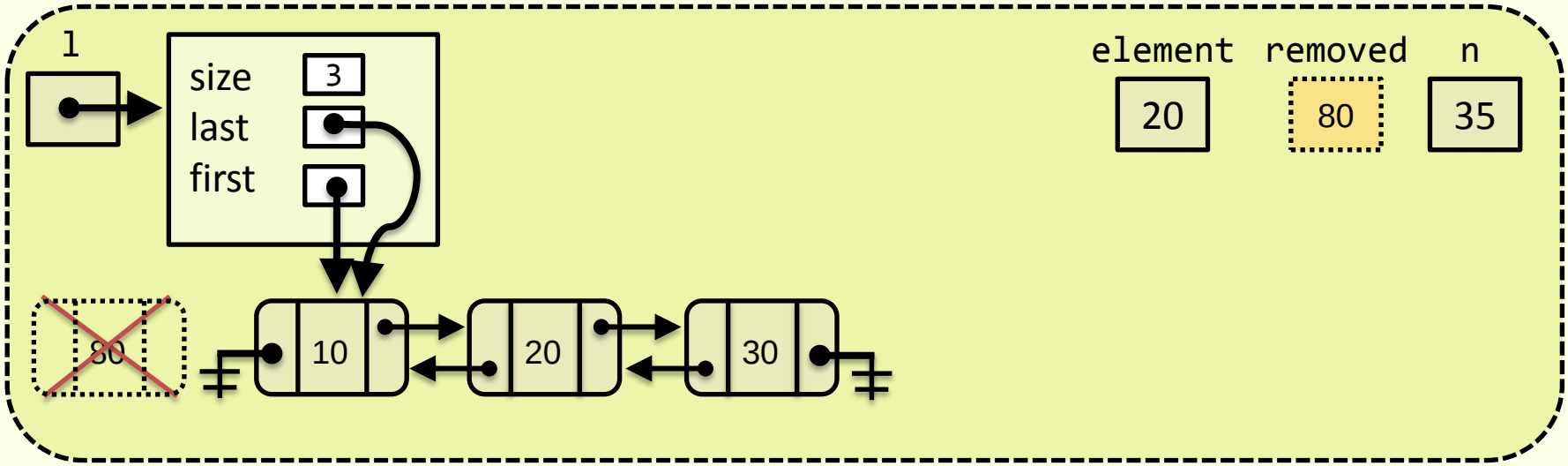
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

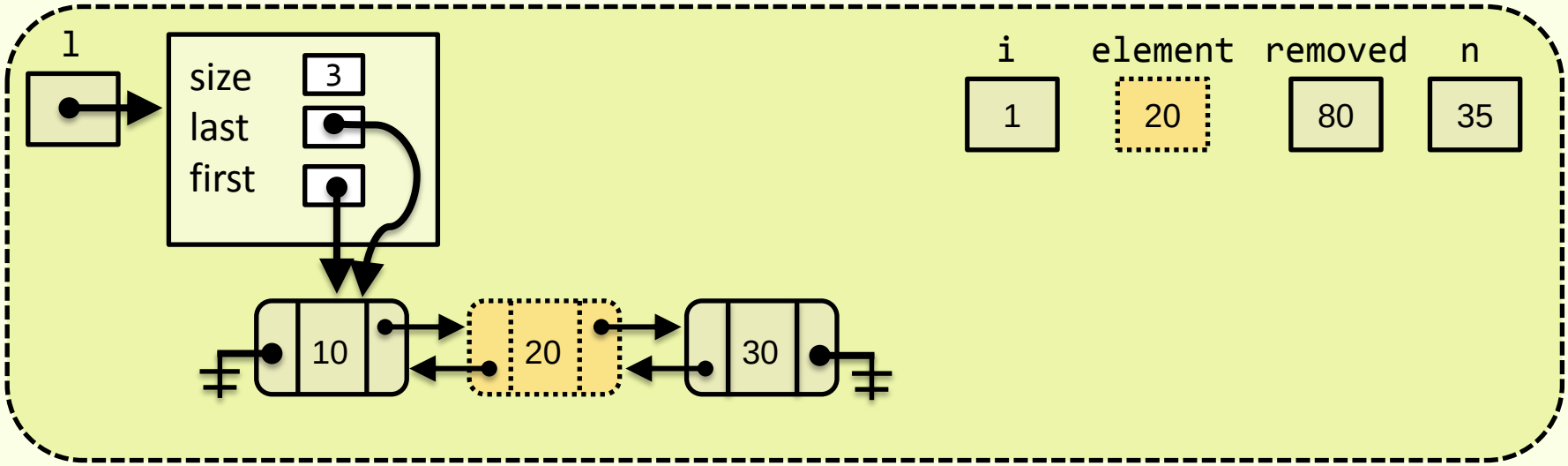
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

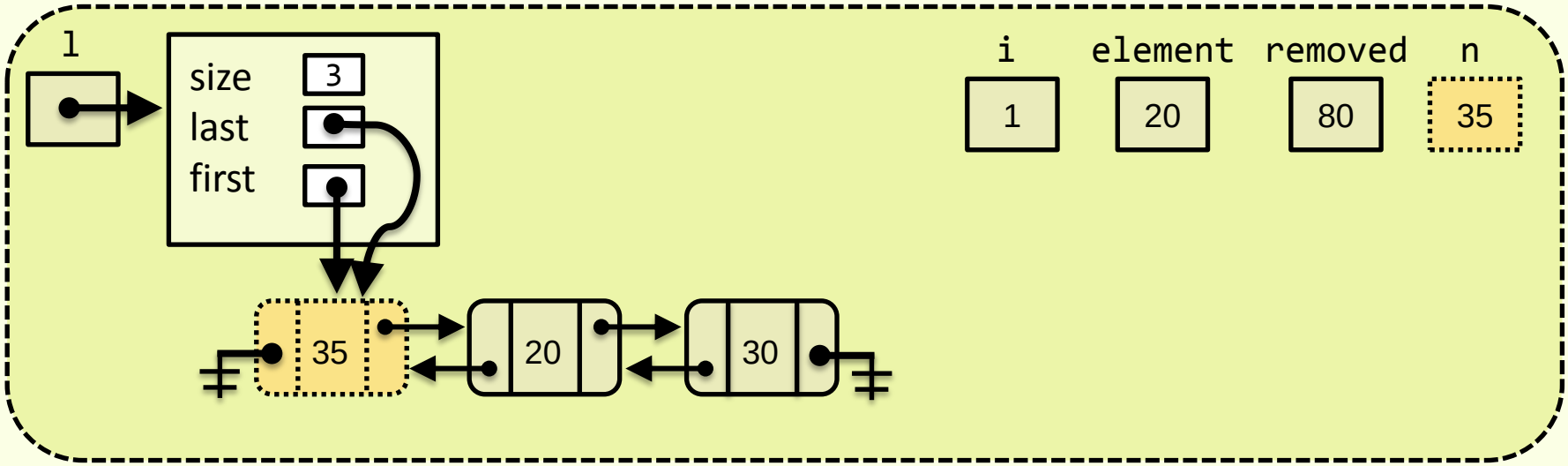
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

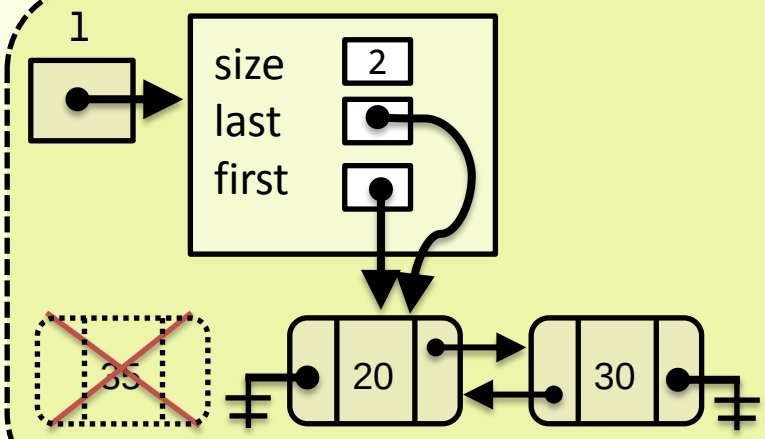
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```

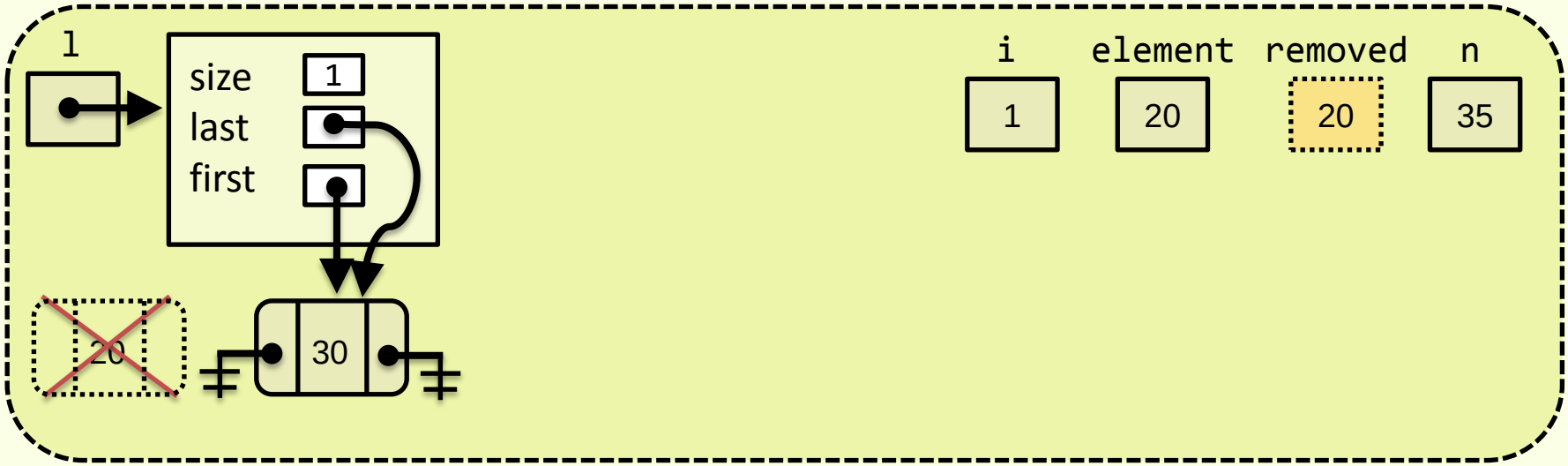



| i | element | removed | n |
|---|---|---|---|
| 1 | 20 | 35 | 35 |

# Simulação

```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



# Implementação

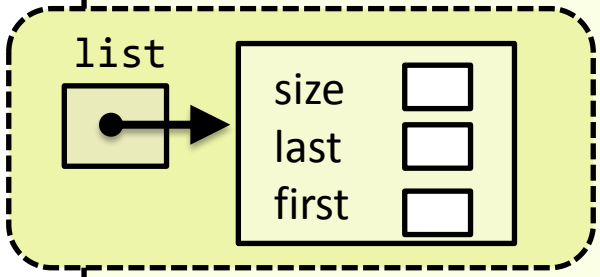
# Implementação

- A partir dessa simulação é possível extrair o comportamento das funções sobre os atributos da lista **duplamente encadeada**

```
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);
```

```
typedef struct node{
    ItemType     data;
    struct node  *prev;
    struct node  *next;
}Node;

typedef struct{
    Node  *first;
    Node  *last;
    int   size;
}List;
```



# Implementação

## LET'S DO IT

# Referências