UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Lista com alocação dinâmica (Encadeada)

Prof. Rafael Liberato
liberato@utfpr.edu.br

Objetivos

- Entender o funcionamento de uma Lista Dinâmica
- Ser capaz de implementar as operações definidas no TAD Lista manipulando uma estrutura dinâmica de armazenamento.

Roteiro

- TAD Lista
- Lista Dinâmica (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);
```

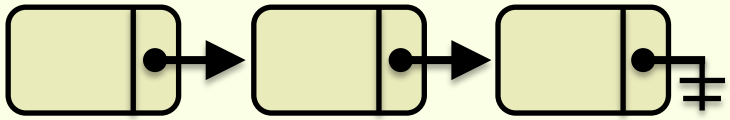
Vamos identificar os atributos que representarão a lista dinâmica

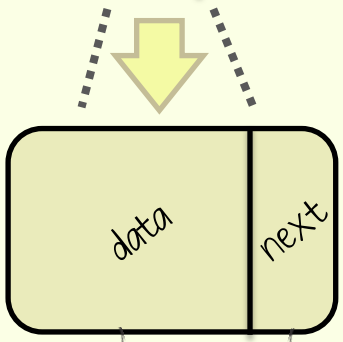
Estrutura utilizada para armazenar os dados

Lista Dinâmica (Encadeada)

Lista Dinâmica (Encadeada)

- A lista **encadeada** utiliza uma estrutura de alocação dinâmica de memória para o armazenamento dos dados
- Portanto, temos que utilizar uma estrutura própria para armazenar e interligar os dados
  - ⇒ Um encadeamento de nós





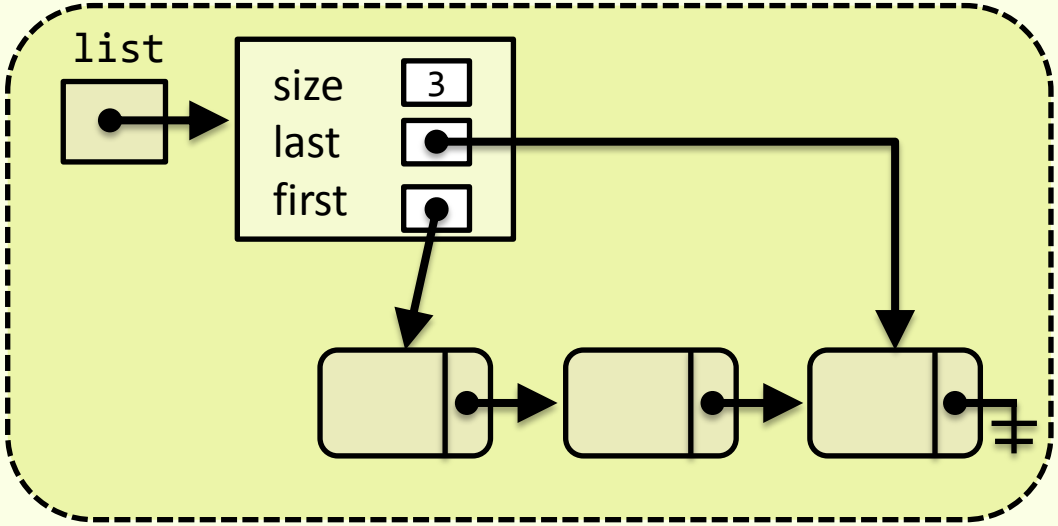
Guarda o endereço do dado armazenado na lista

Guarda o endereço do nó seguinte da lista

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

Lista Dinâmica (Encadeada)

- Para armazenar os elementos a lista deve guardar o endereço do primeiro e do último nó do encadeamento.
- Também utilizaremos um atributo para guardar a quantidade de elementos contidos na lista.




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

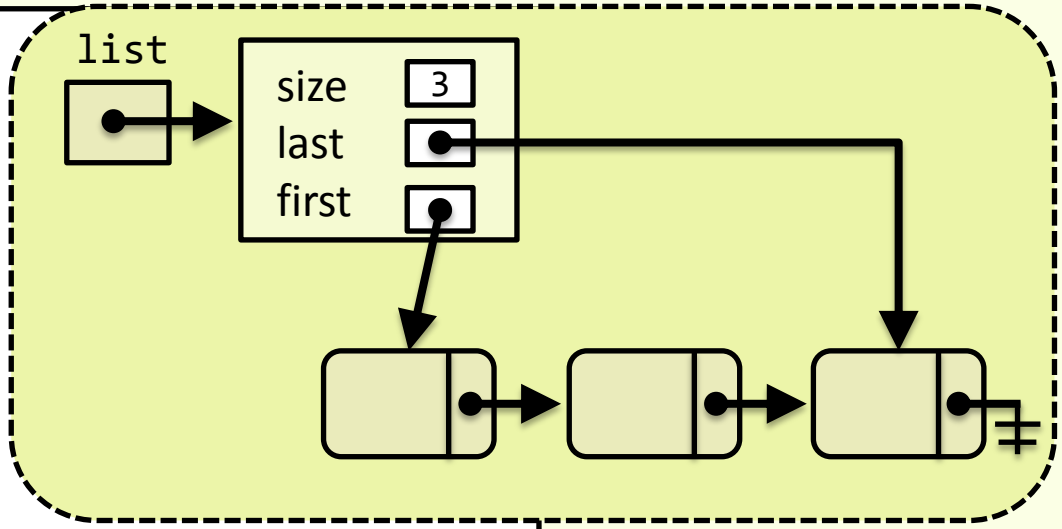
```
typedef struct{
    Node *first;
    Node *last;
    int size;
}List;
```

Lista Dinâmica (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* l, ItemType e, int index);
int removeList(List* l, int index, ItemType *e);
int removeList(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 *next;
}Node;
```

Simulação

Simulação

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

Simulação

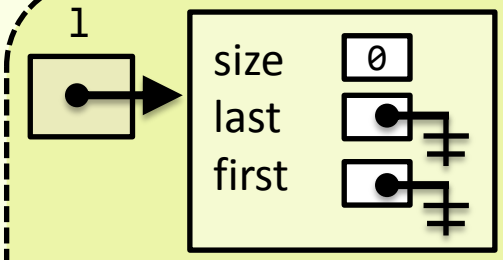
```
List *l = createList();
addList(l,10);
addList(l,20);
addList(l,30);
addList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

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

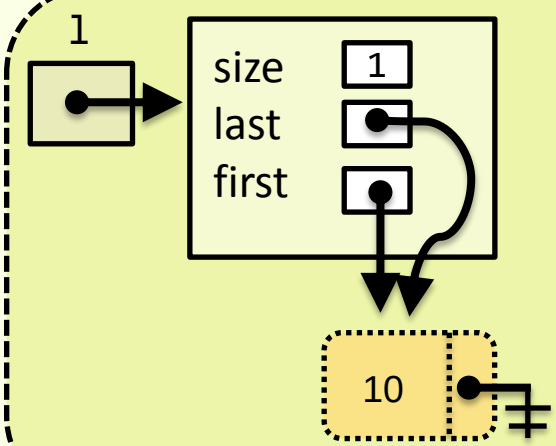
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

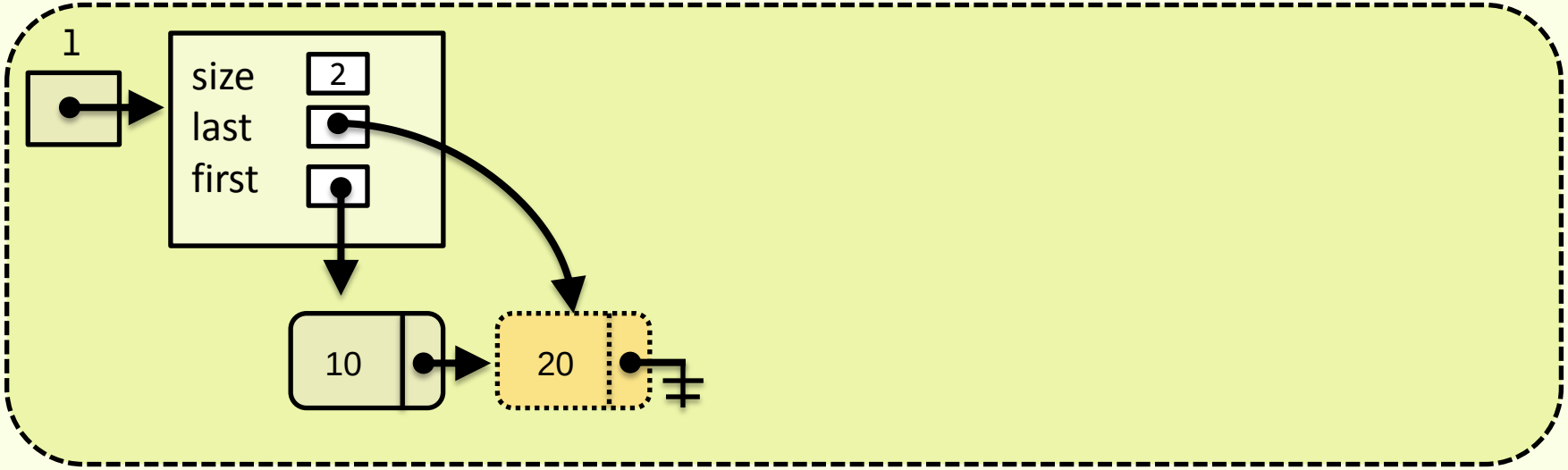
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

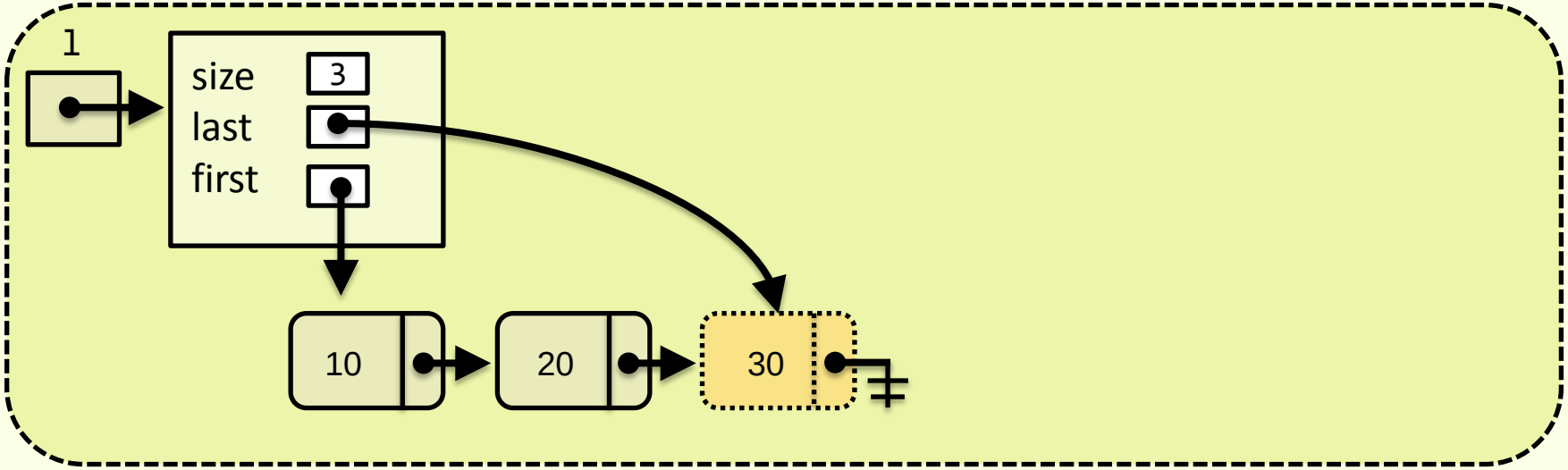
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

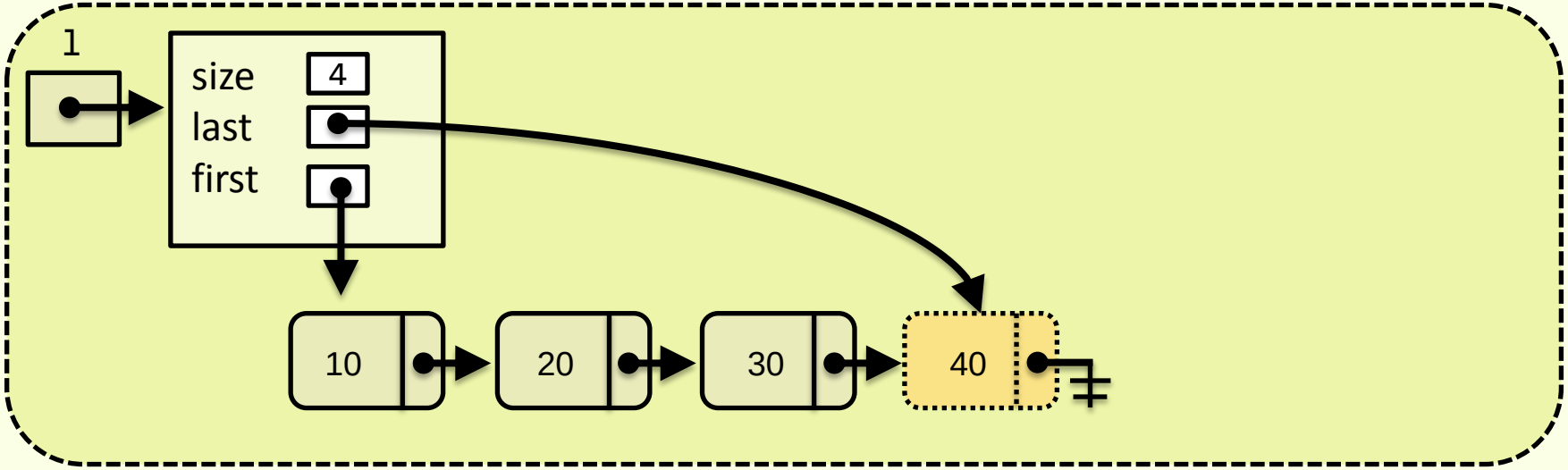
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

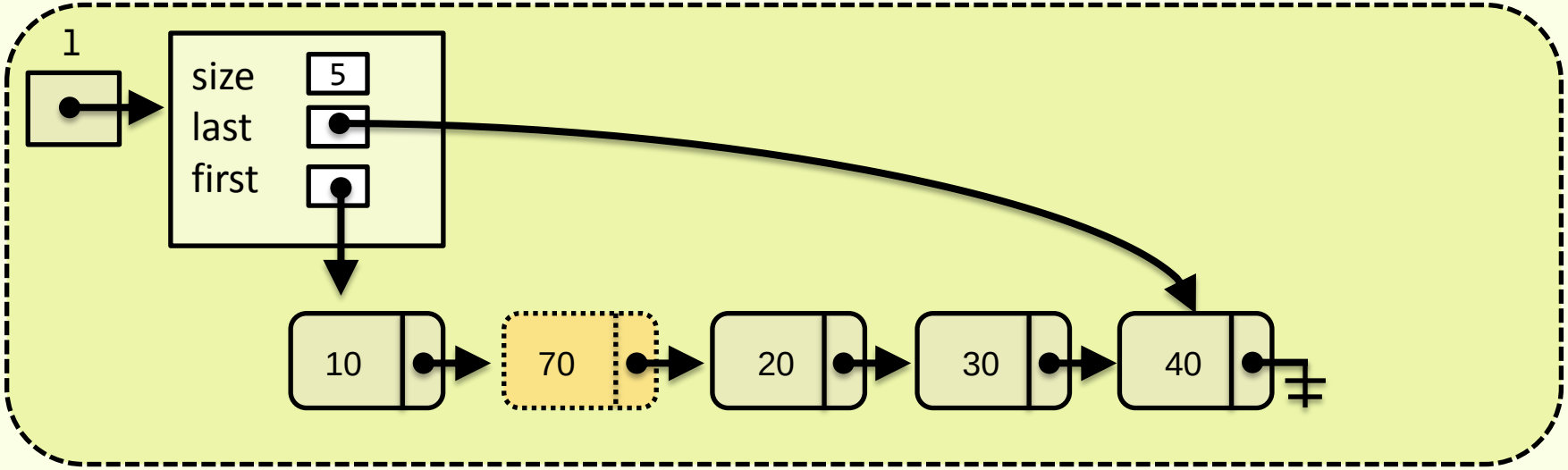
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

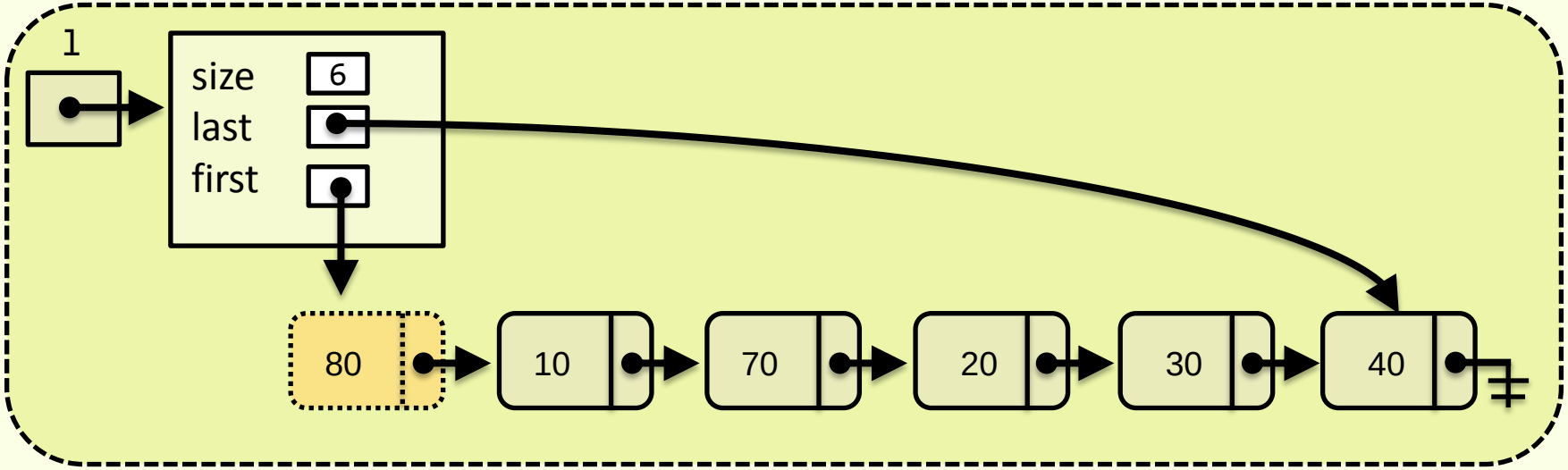
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

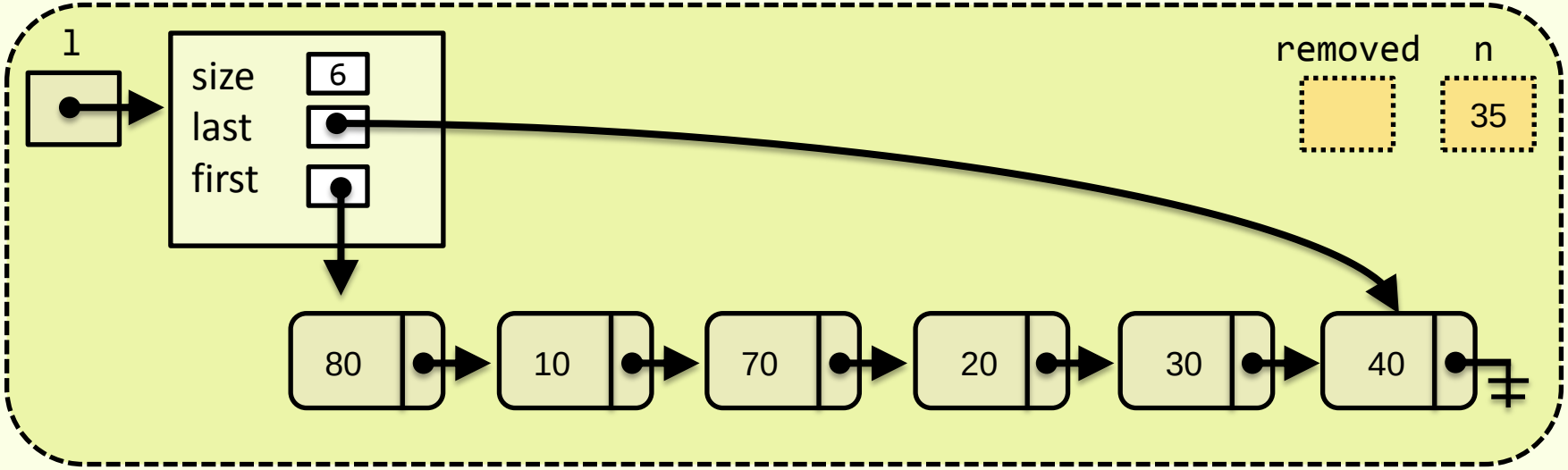
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

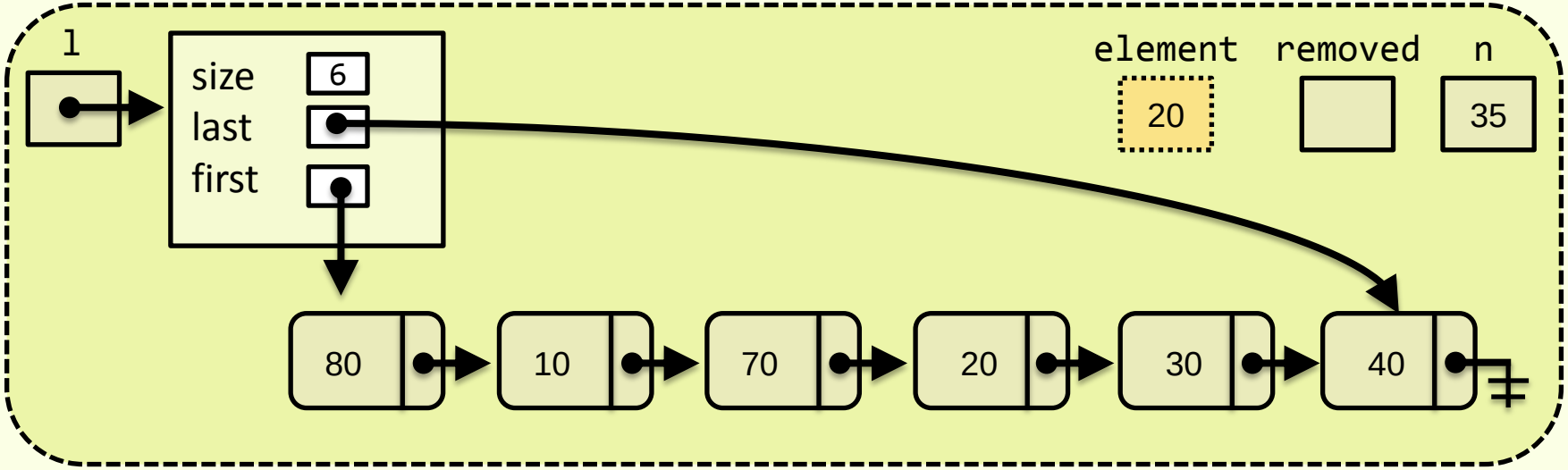
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

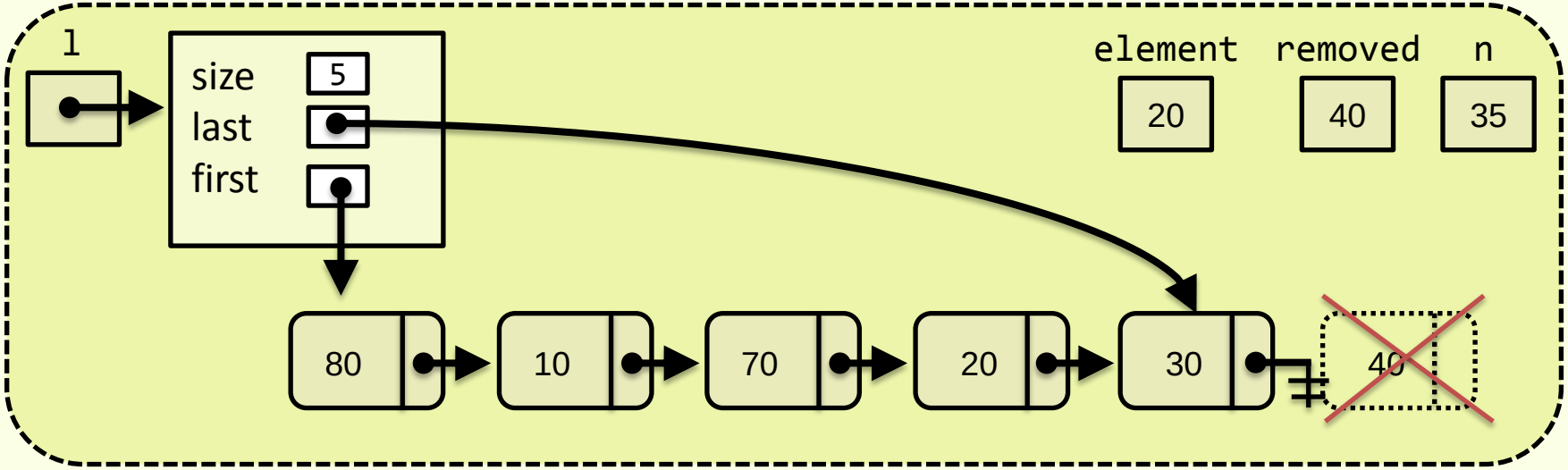
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

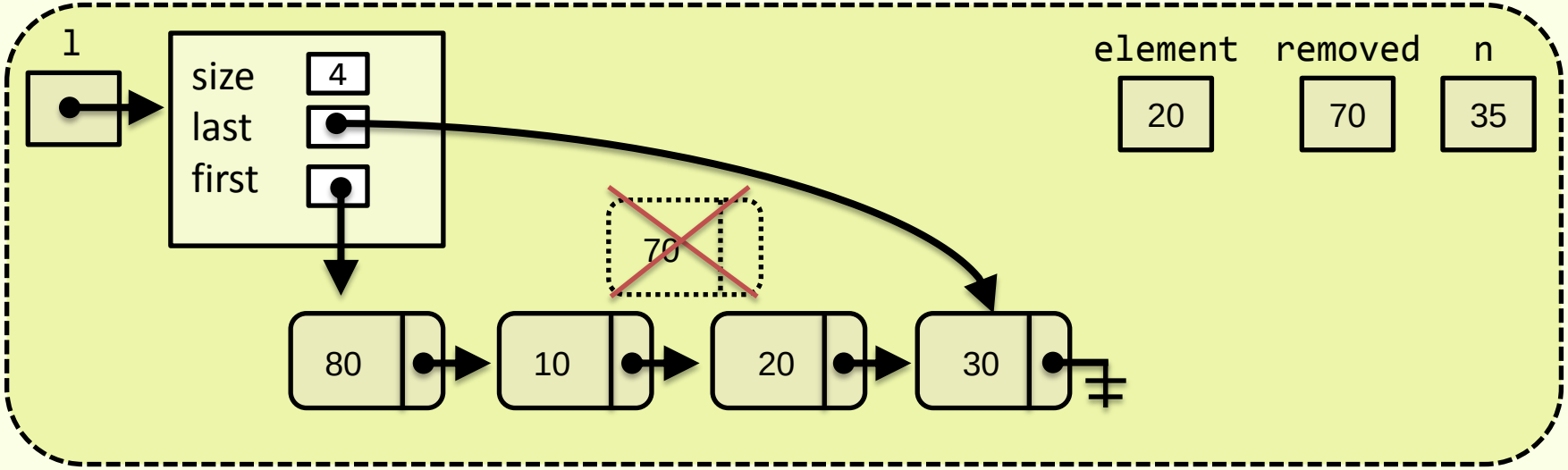
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

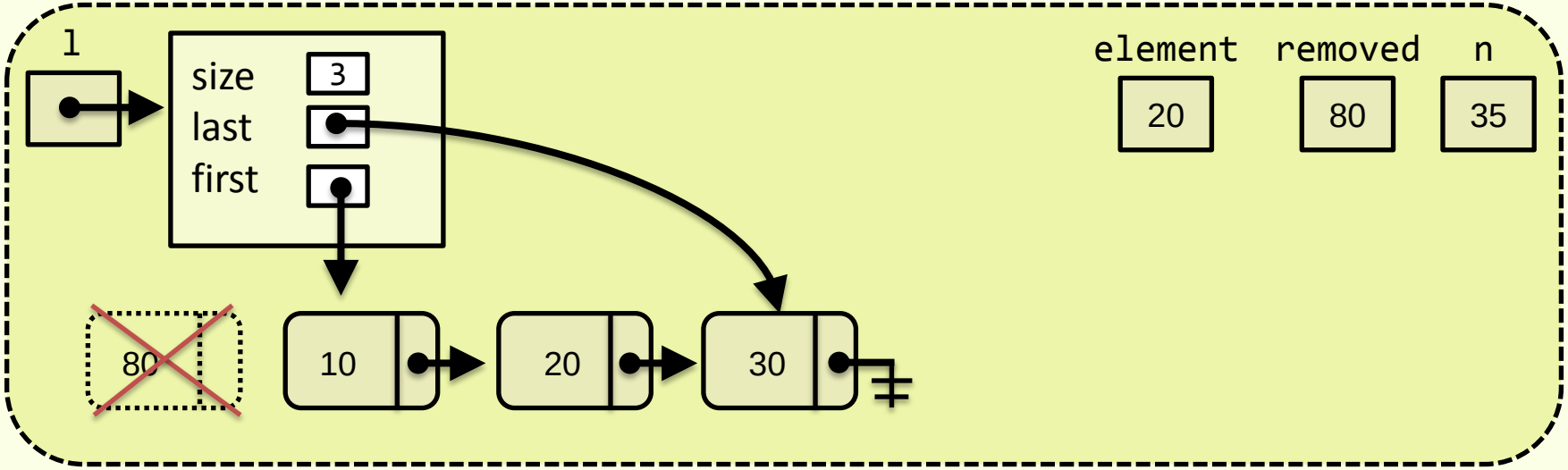
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

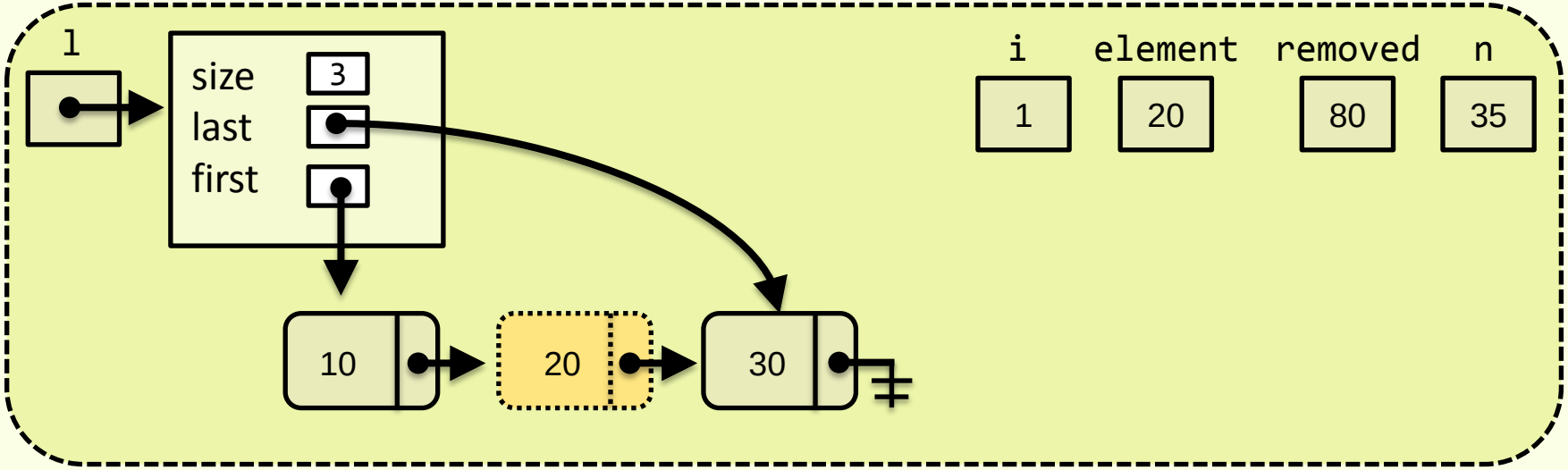
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

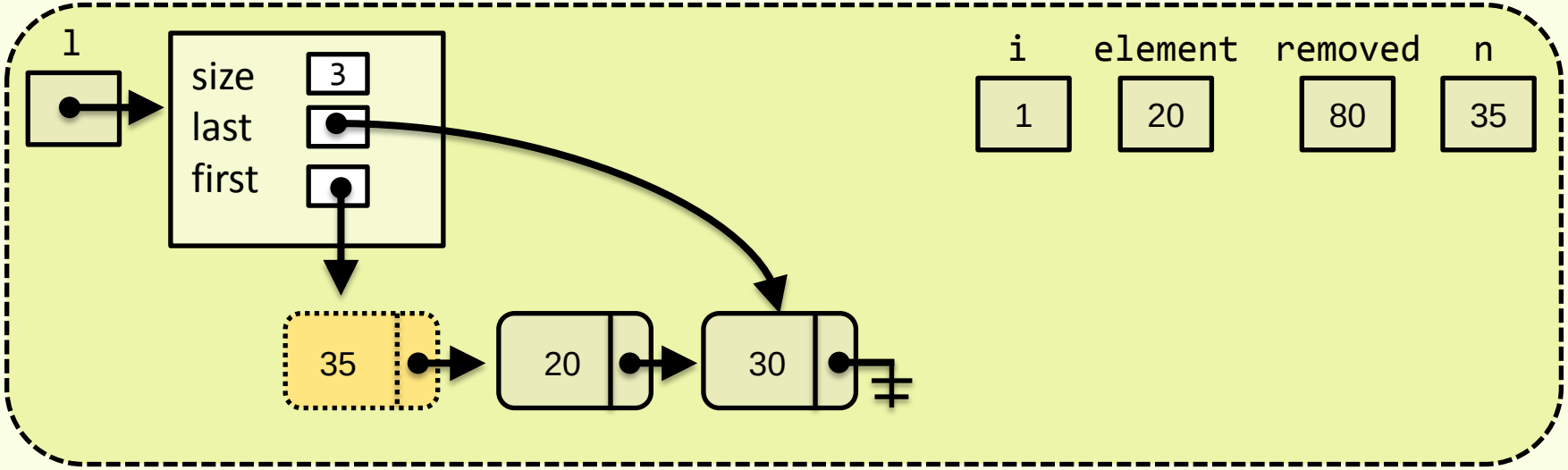
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

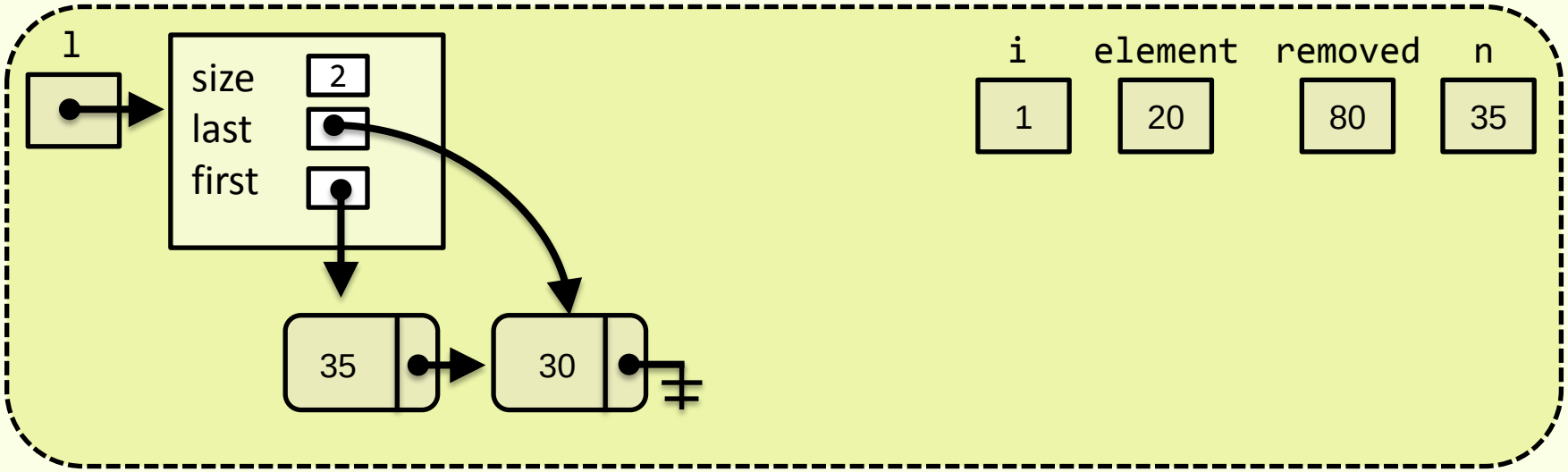
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

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

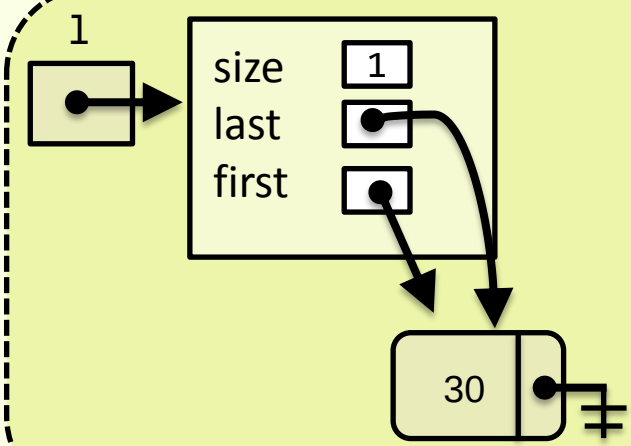
```
ItemType element = 20;
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



Simulação

```
List *l = createList();
addList(l,10);
addList(l,20);
addList(l,30);
addList(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	80	35

Implementação

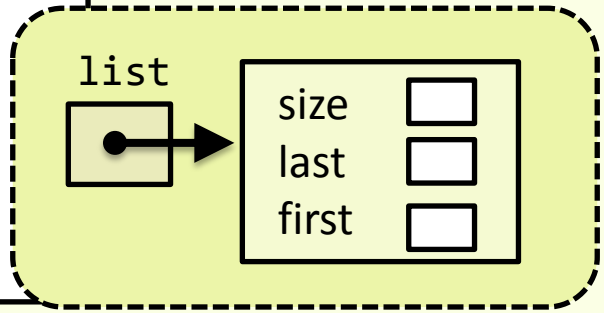
Implementação

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

```
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 *next;
}Node;

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



Implementação

LET´S DO IT

Referências