UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Campo Mourão

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Lista com alocação estática

Prof. Rafael Liberato
liberato@utfpr.edu.br

Objetivos

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

Roteiro

- TAD Lista
- Lista Estática
- 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 estática

Estrutura utilizada para armazenar os dados

Lista Estática

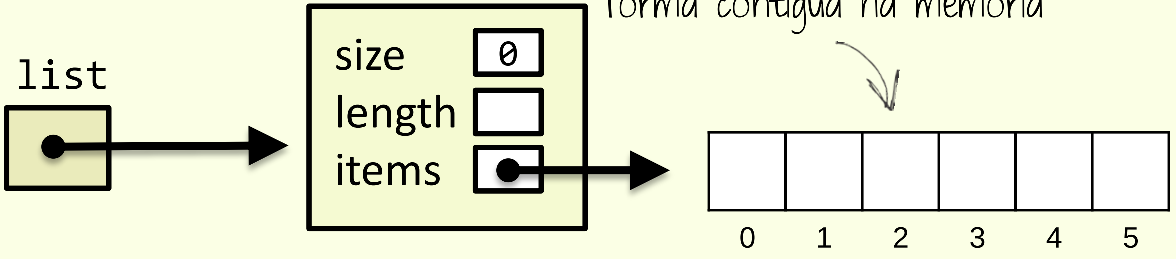
Lista Estática

- A lista 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 lista são armazenados em um vetor
- Como a lista estática utiliza um vetor para armazenar os dados não precisamos de um atributo para representar o primeiro elemento
  - Precisaremos representar somente o último elemento

Lista Estática

Atributos

- O atributo `items` armazena o endereço do array utilizado para armazenar os elementos da lista
- O atributo `size` possui dois significados
  - Armazena a quantidade de elementos da lista
  - Representa a primeira posição vazia do array, marcando o final da lista.



Os elementos são armazenados de forma contígua na memória

```
typedef struct{
    int size;
    int length;
    ItemType *items;
}List;
```

Simulação

Simulação

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

Simulação

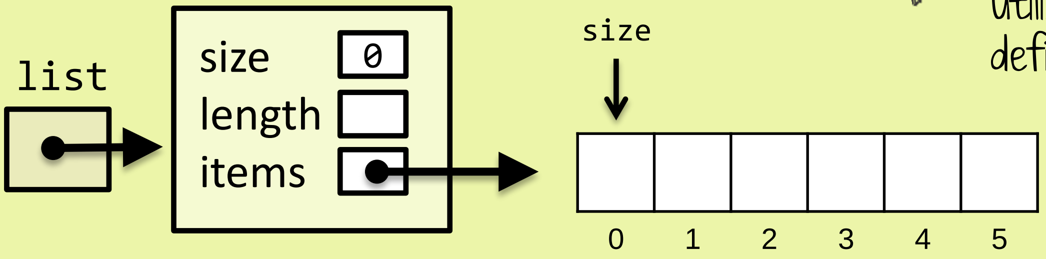
```
List *l = createList();
addList(l,10);
addList(l,20);
addList(l,30);
addList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
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;
```

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

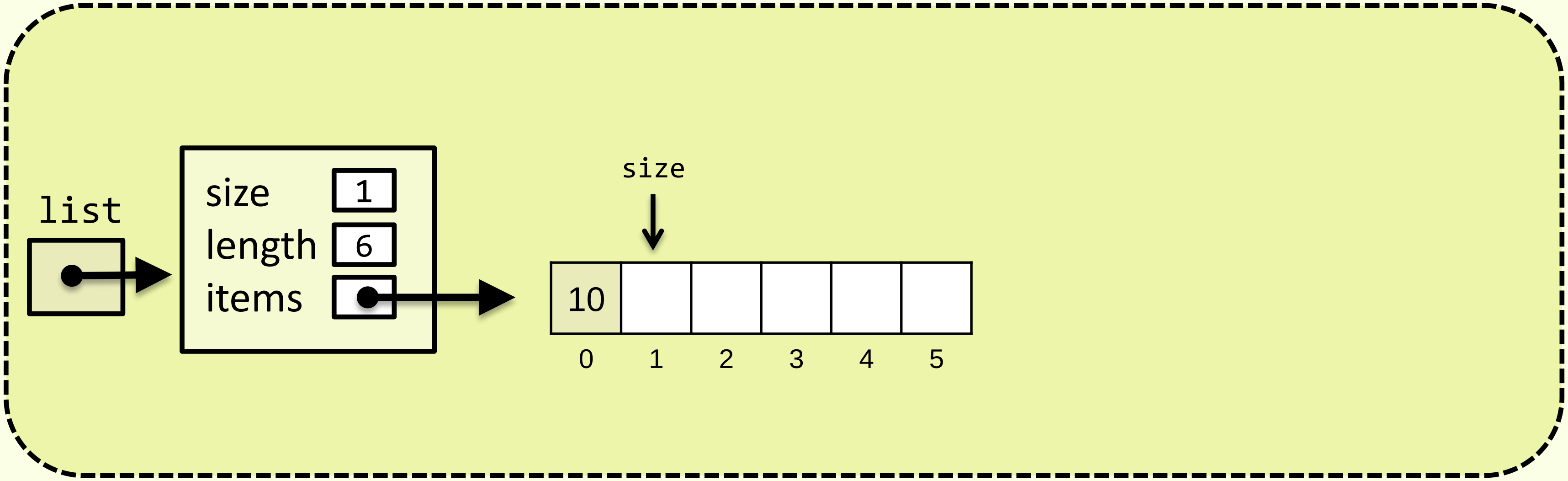


O tamanho do vetor é definido dentro da função. Normalmente utilizamos uma constante para definir o tamanho do vetor

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;
```

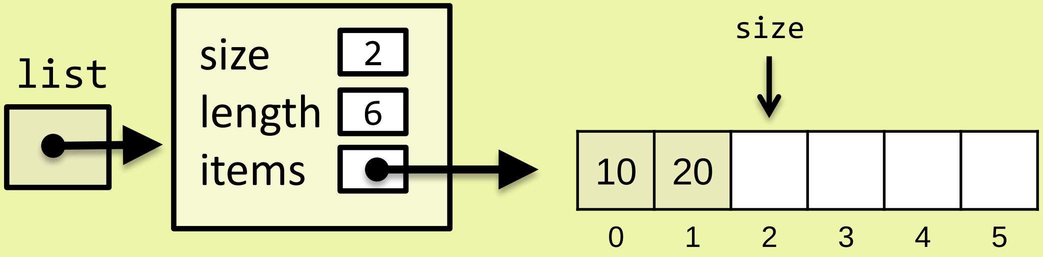
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

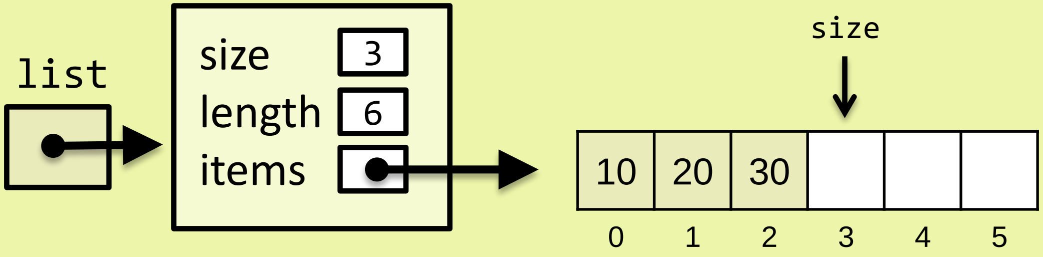
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

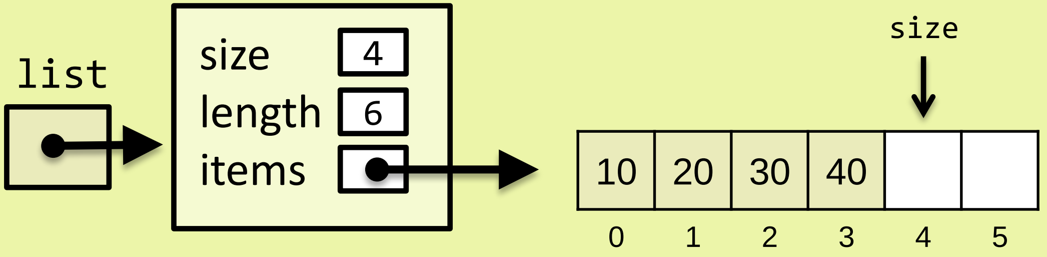
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

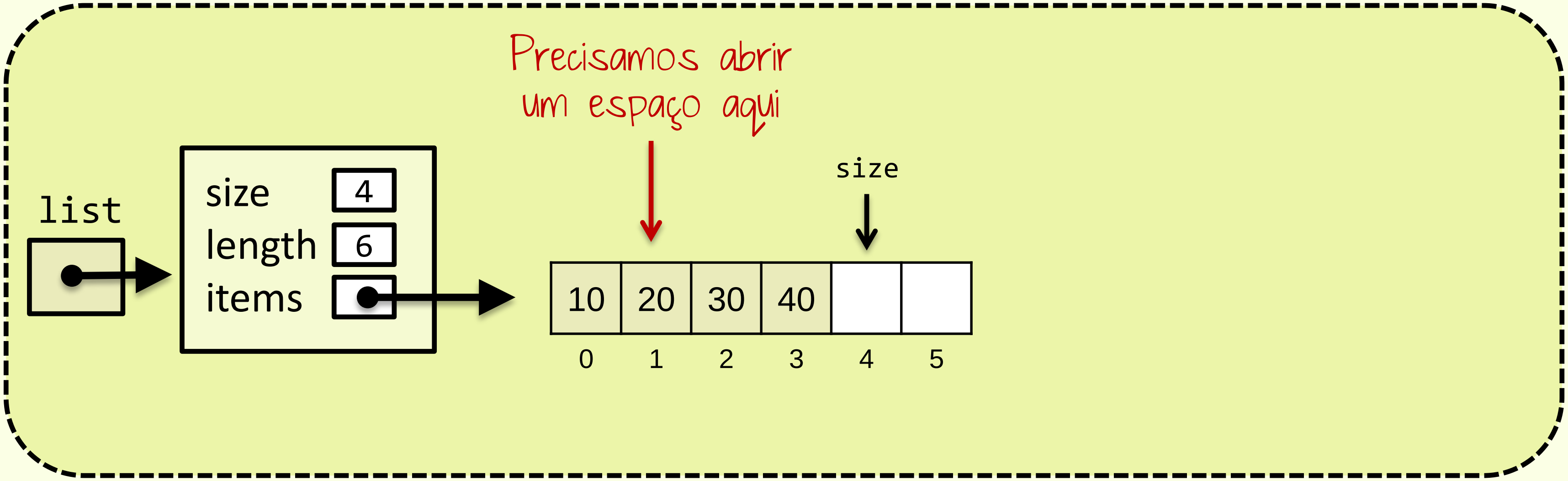
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

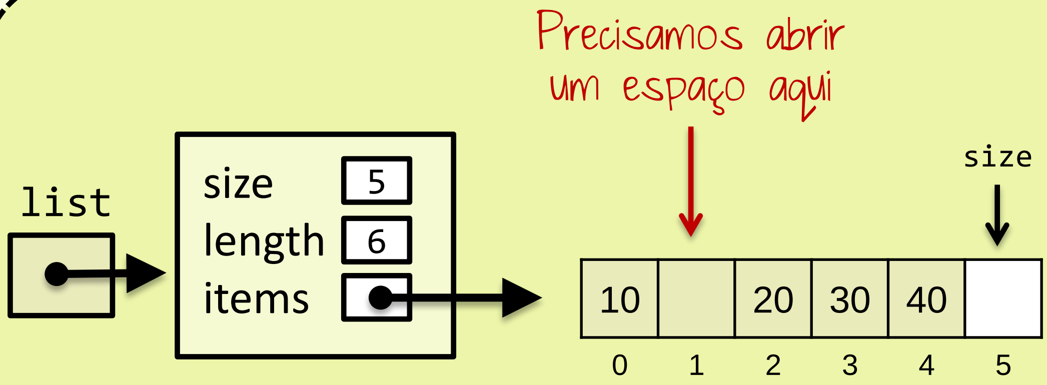
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```


Precisamos abrir um espaço aqui

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;
```

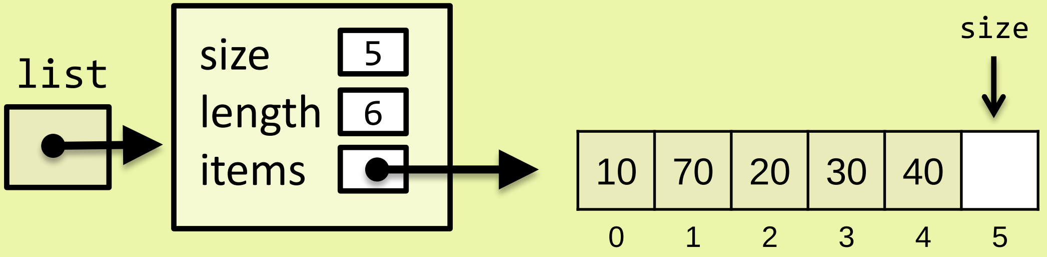
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

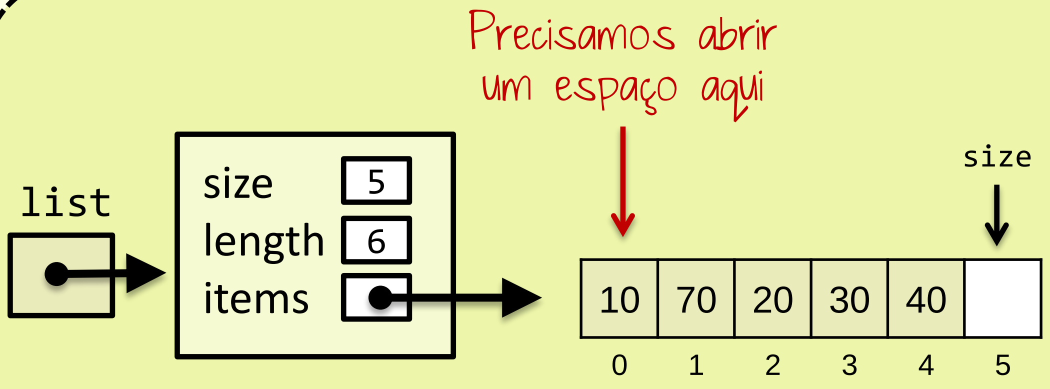
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

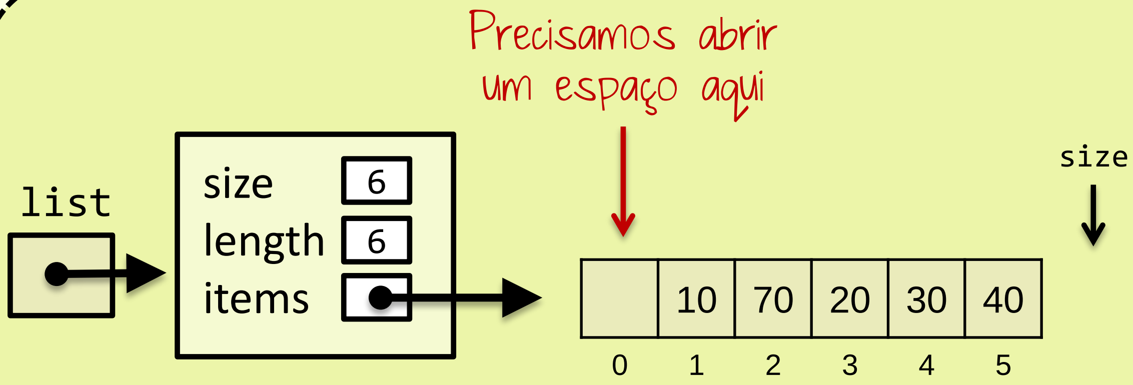
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

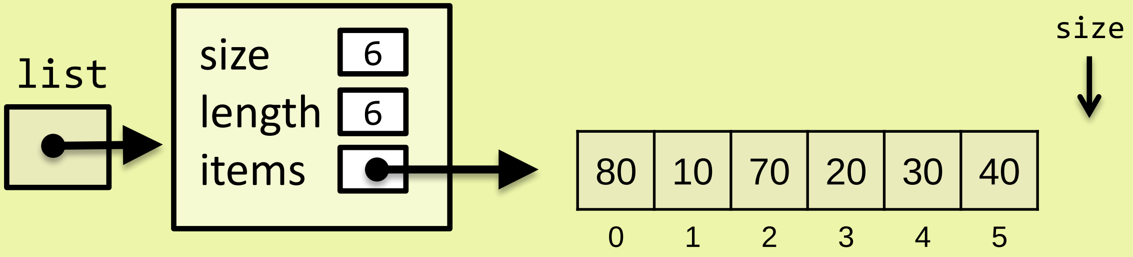
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

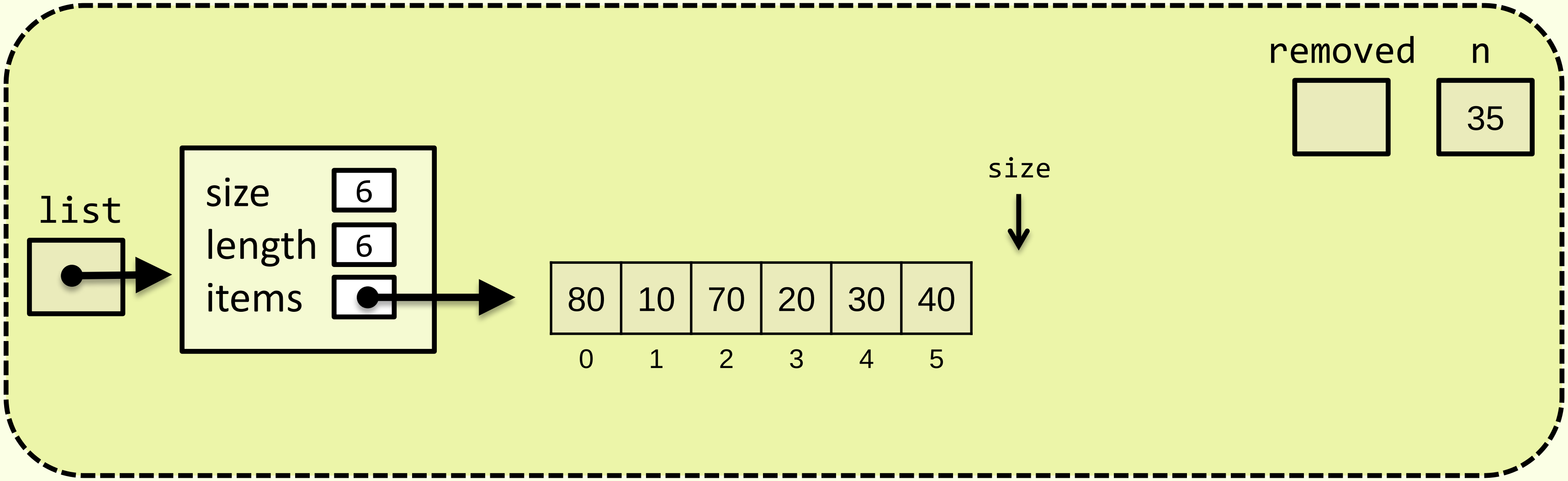
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

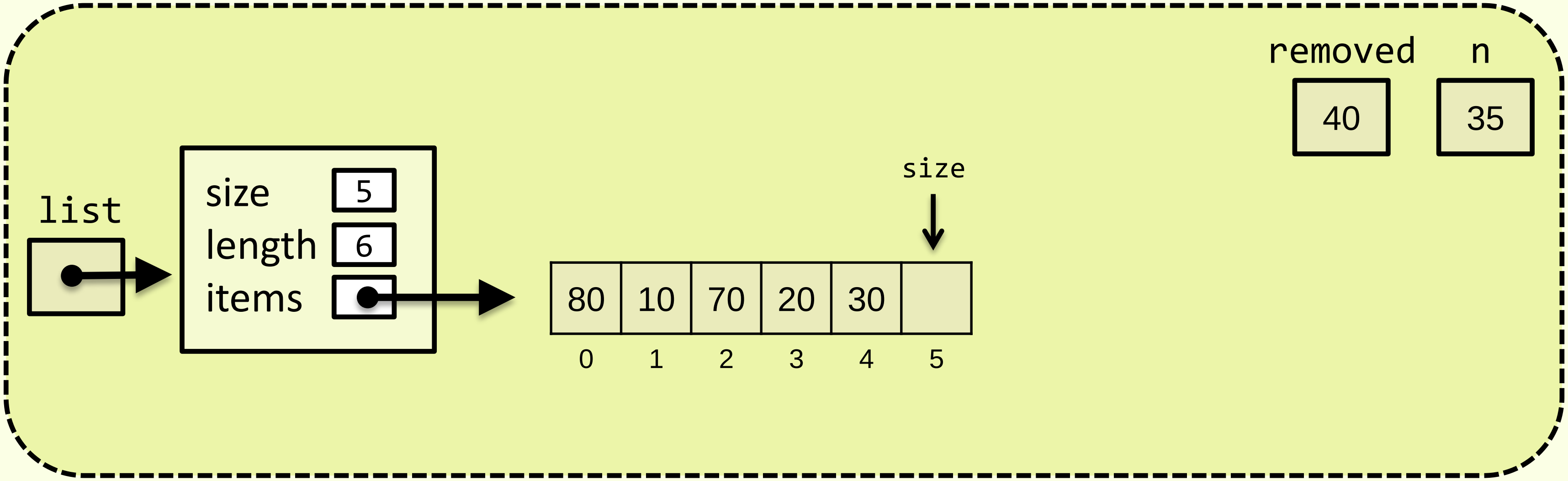
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

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

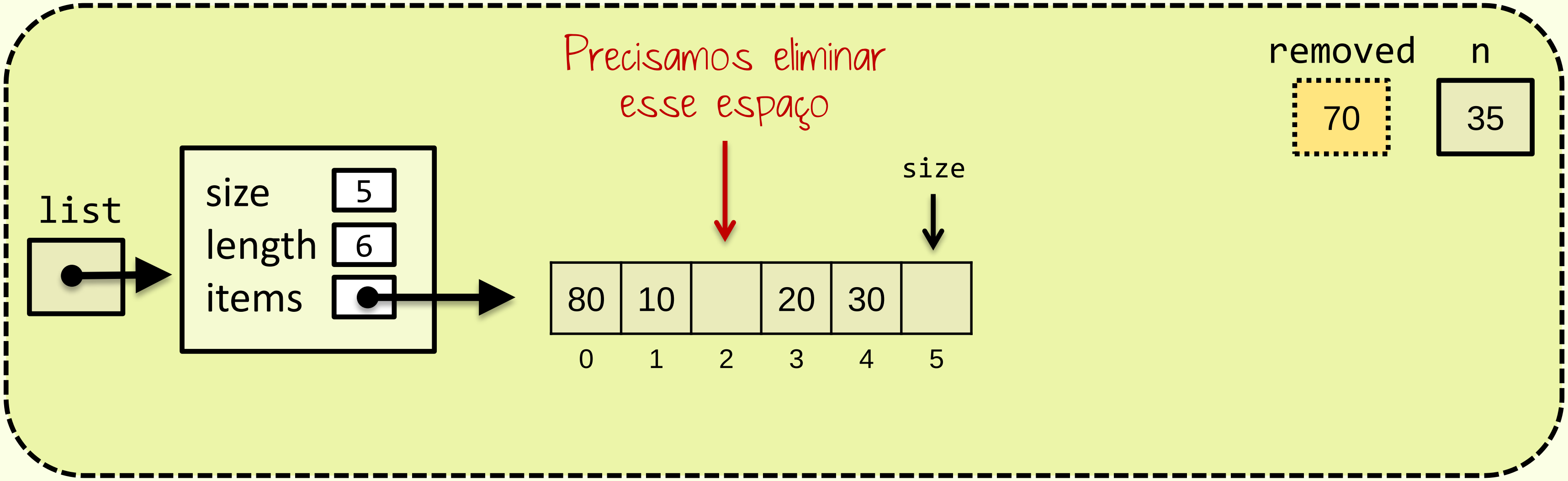


0	1	2	3	4	5
80	10	70	20	30	

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;
```

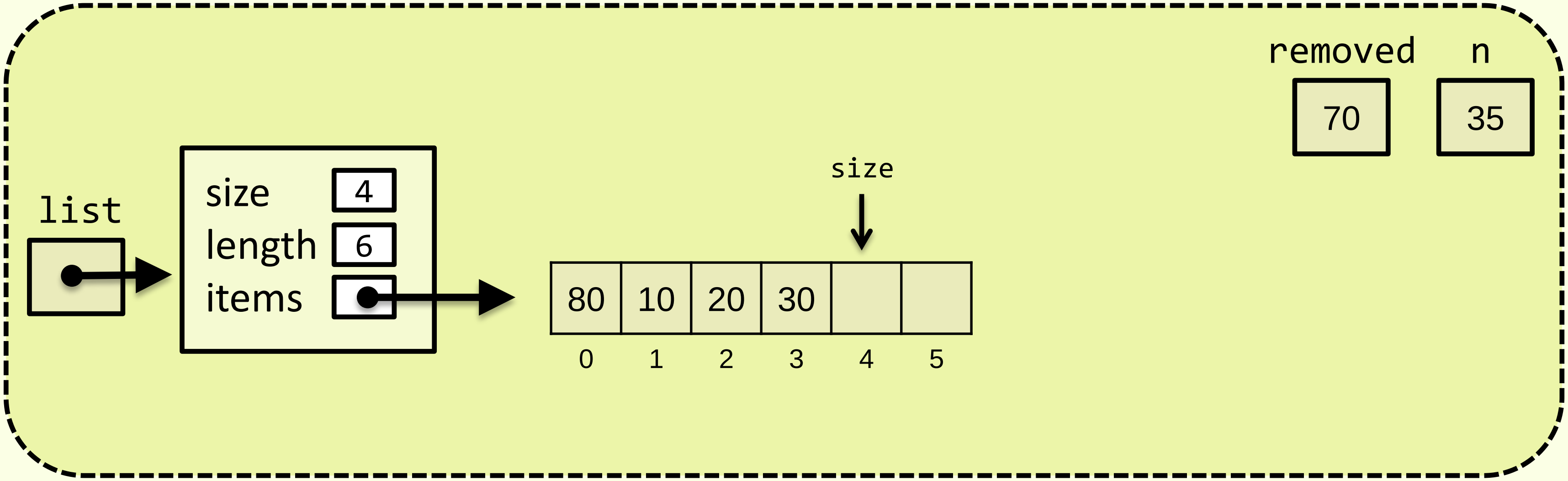
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

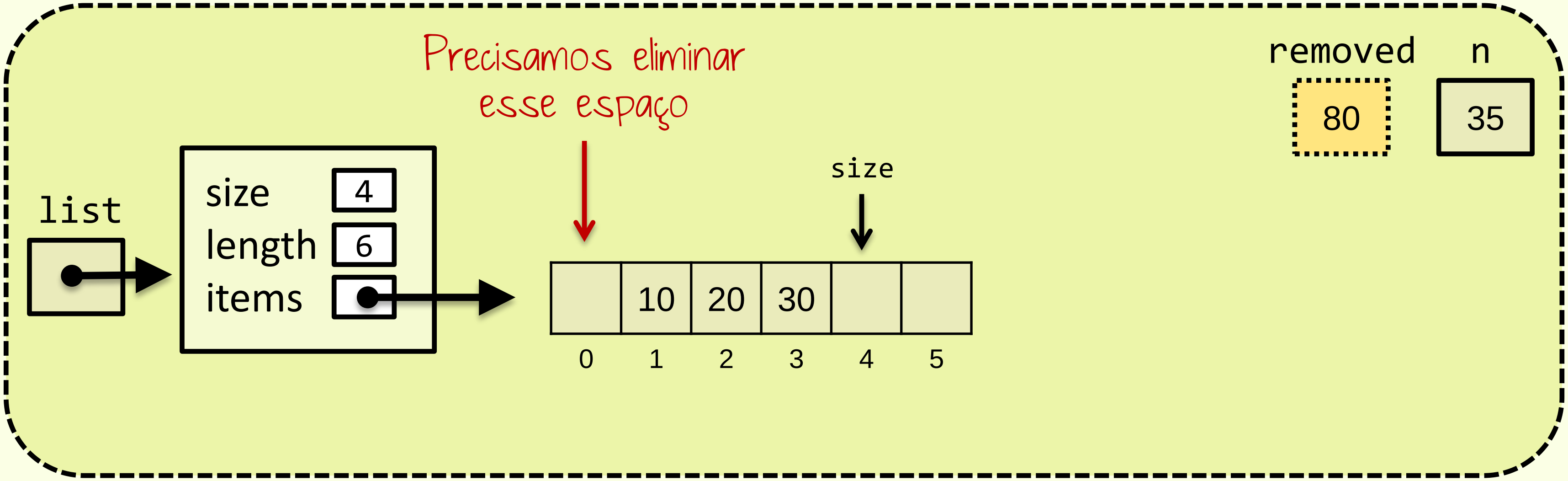
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

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

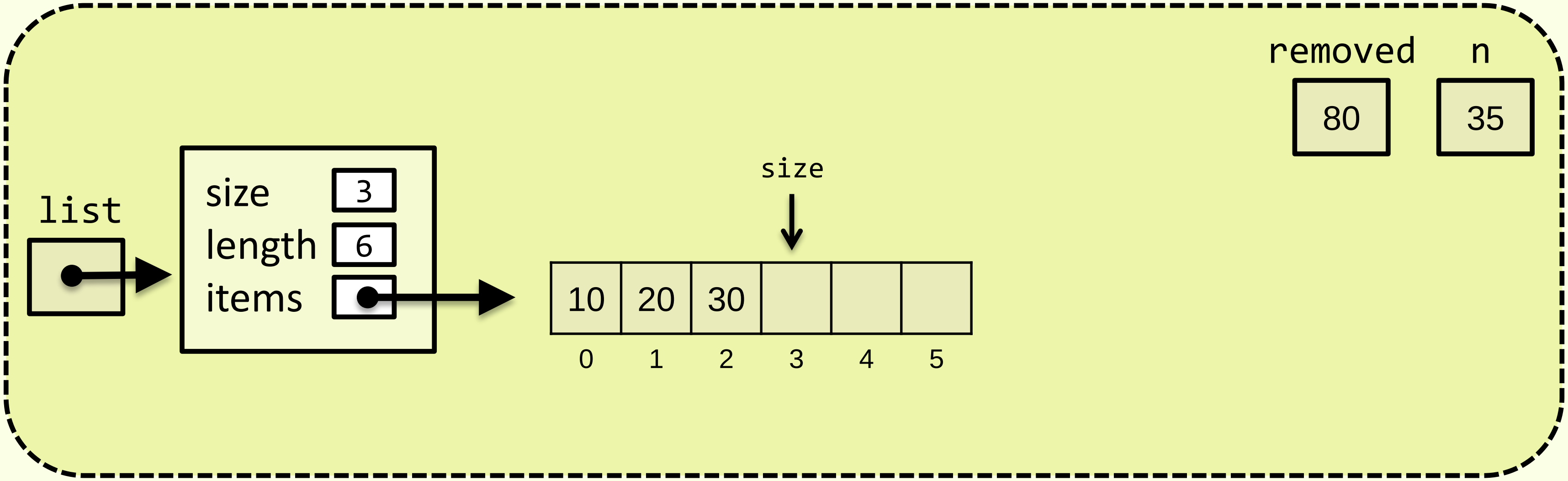


Precisamos eliminar esse espaço

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;
```

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

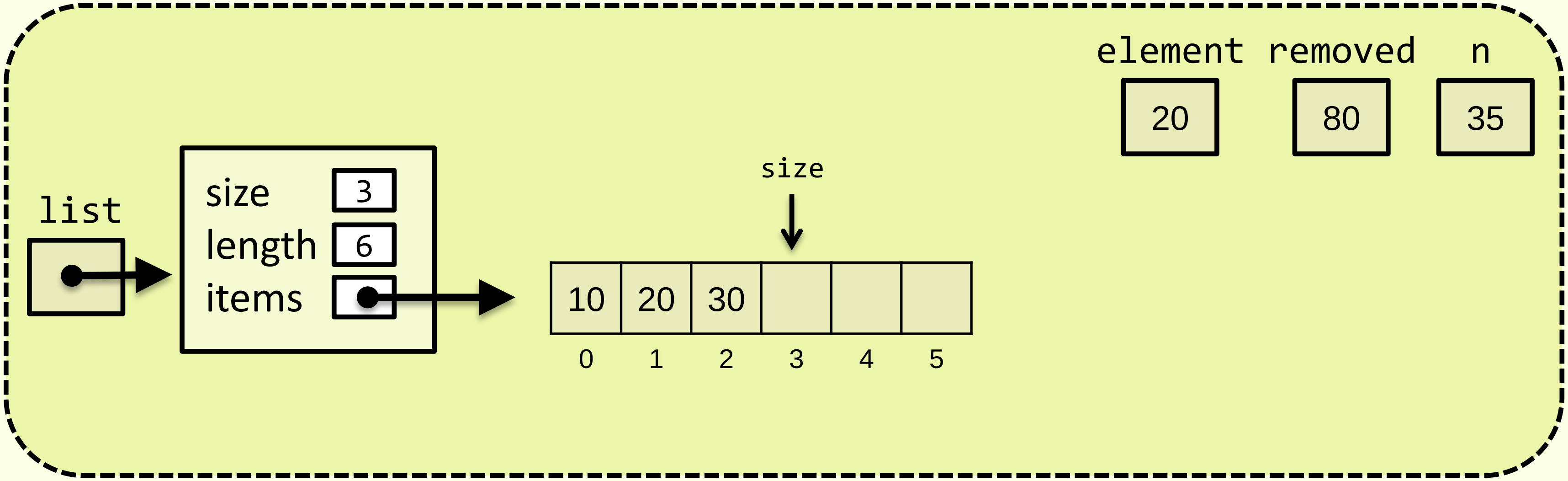


Prof. Rafael Liberato

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;
```

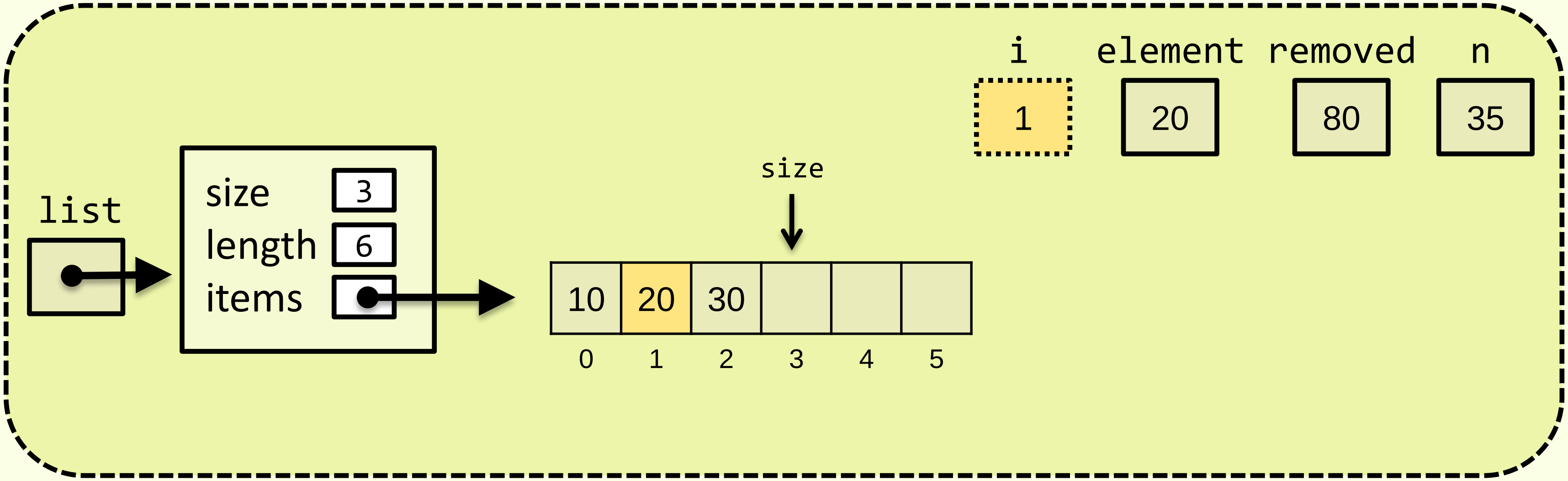
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

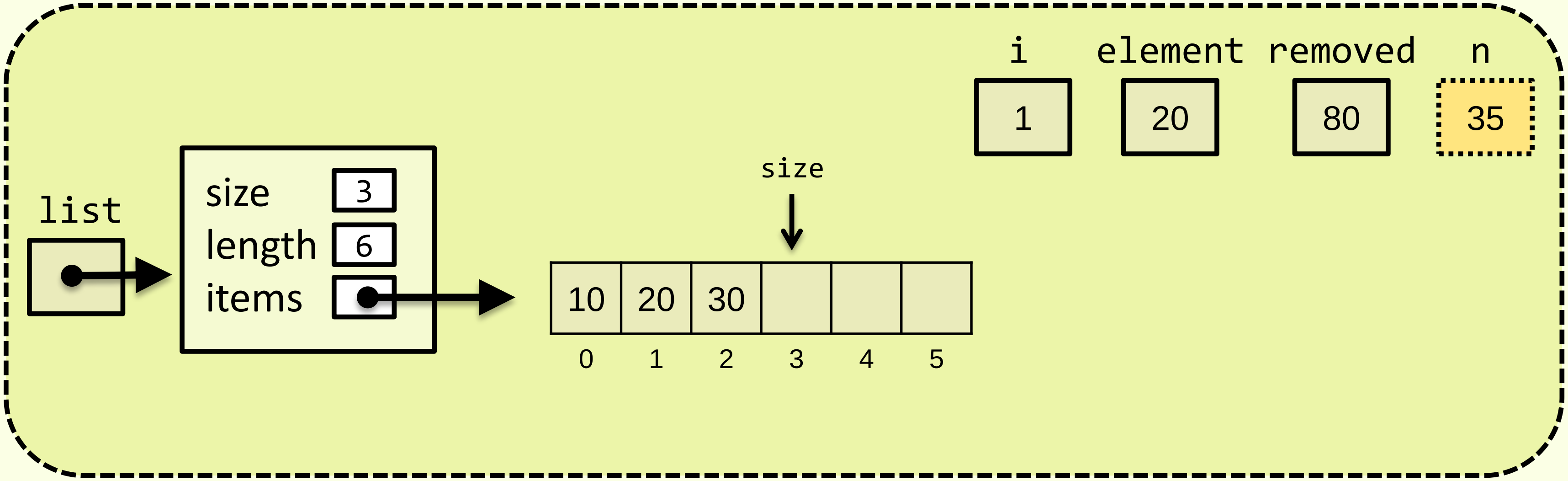
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

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



i element removed n

1 20 80 35

list

size 3

length 6

items

size

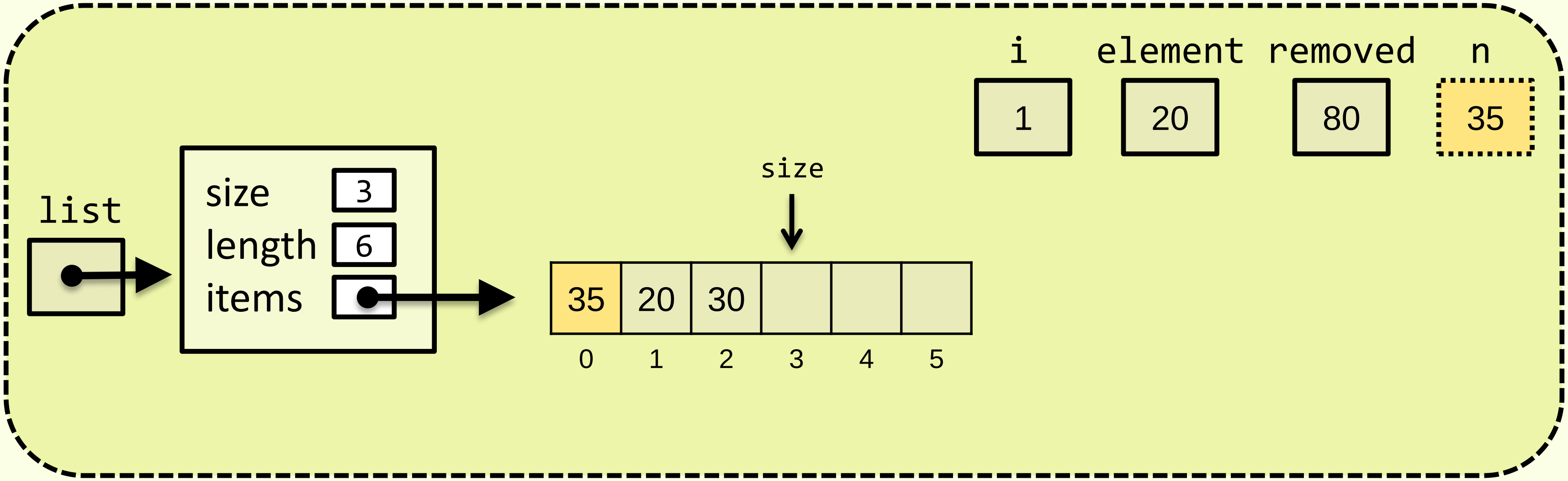
10 20 30

0 1 2 3 4 5

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;
```

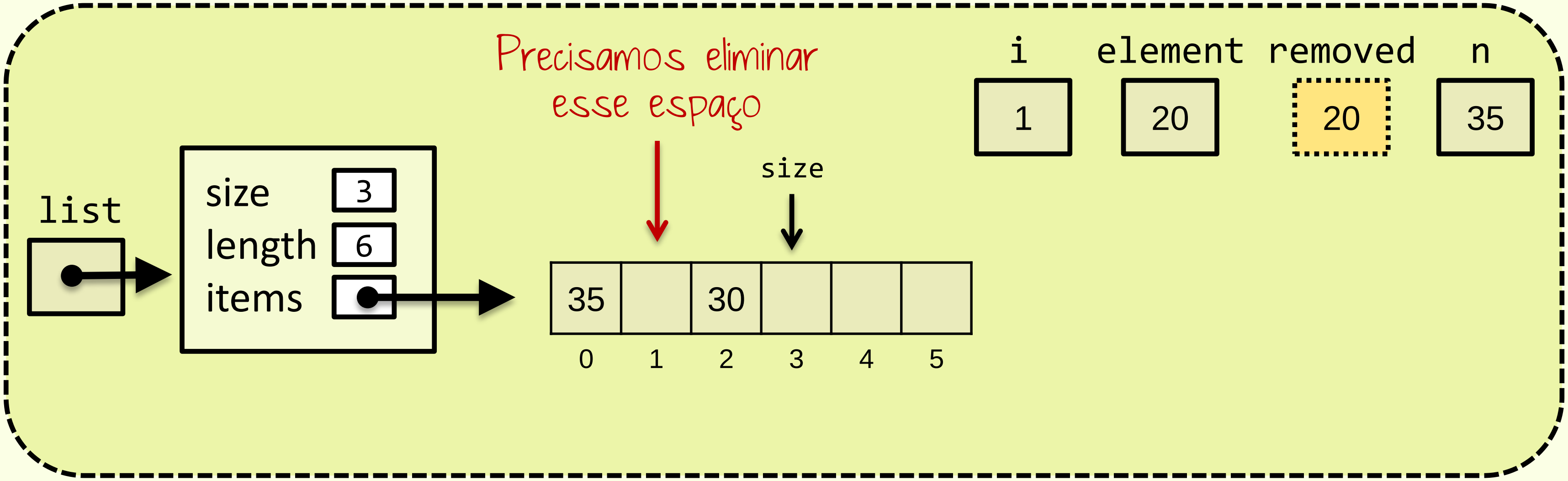
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

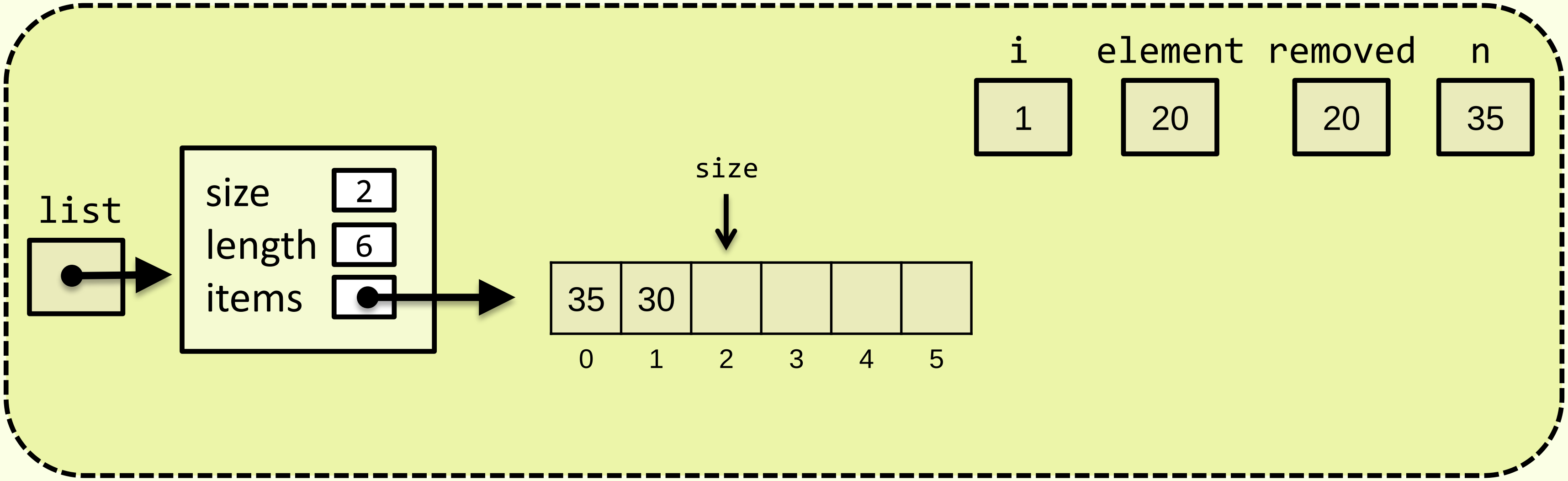
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

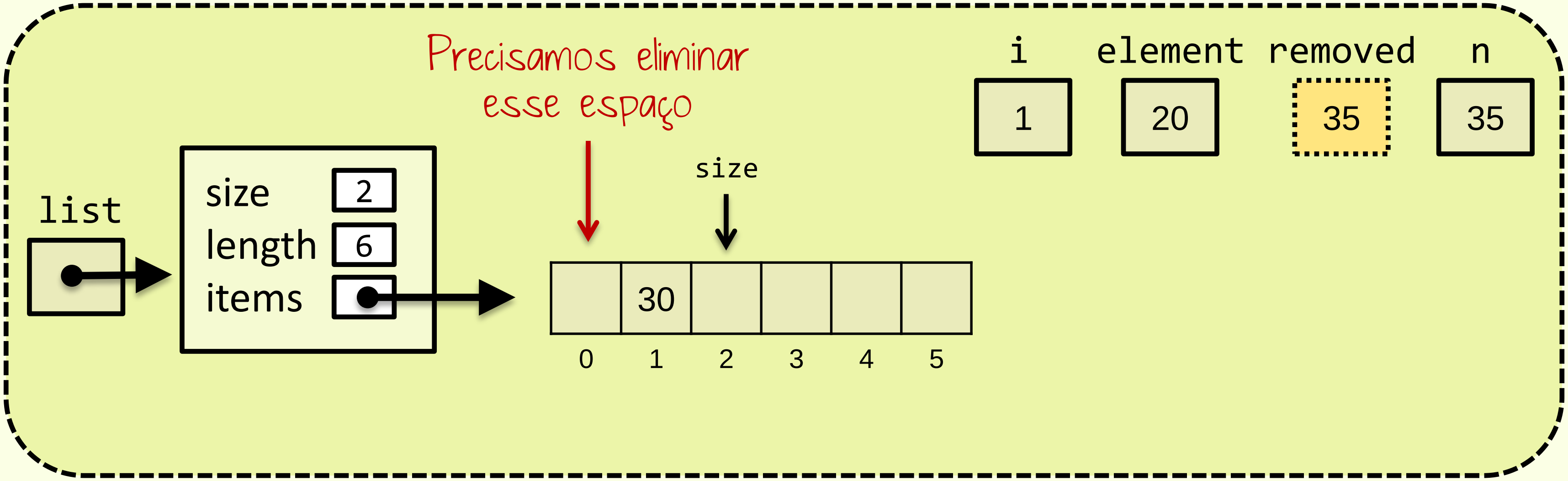
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

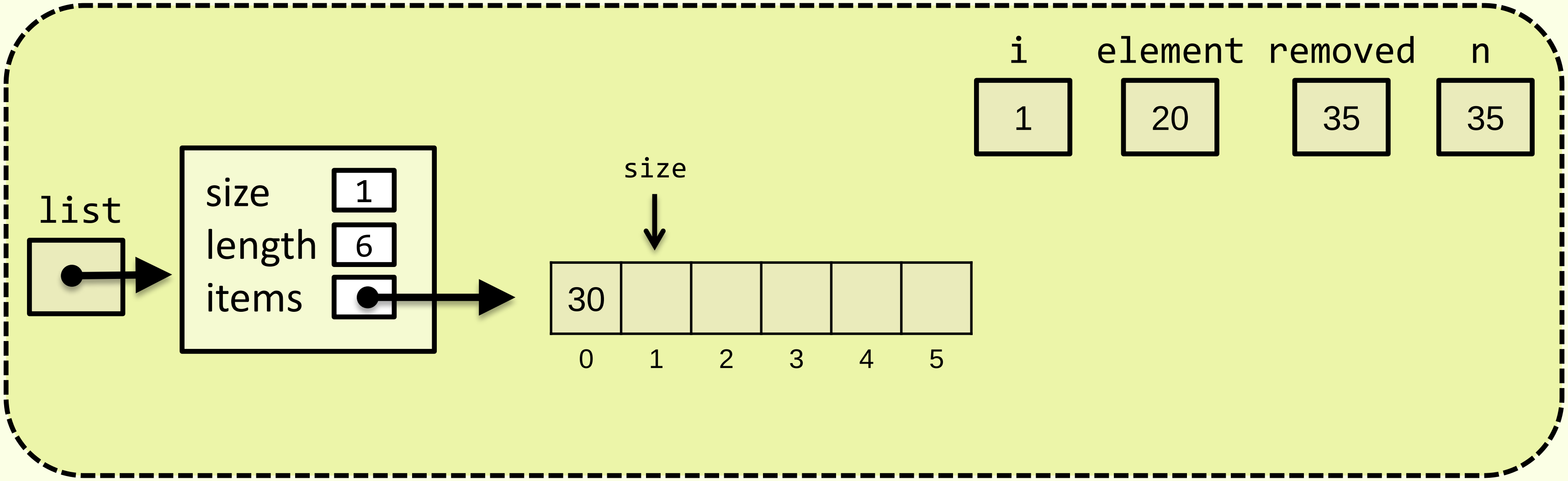
```
removeList(l,5,&removed);
removeList(l,2,&removed);
removeList(l,0,&removed);
ItemType element = 20;
int i = indexOfList(l,&element);
setList(l,0,&n);
removeList(l,&element);
removeList(l,0,&removed);
```



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;
```

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



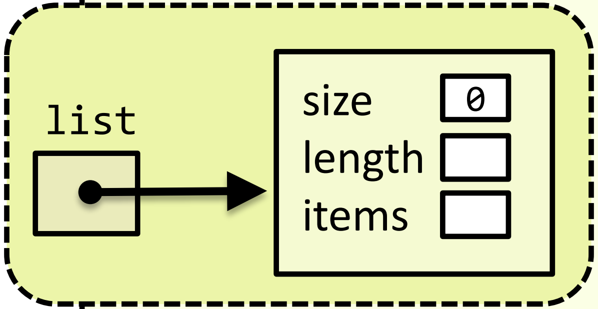
Implementação

Implementação

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

```
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{
    int size;
    int length;
    ItemType *items;
}List;
```



Implementação

LET'S DO IT

Referências