

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**
- **M** Implementação

TAD Lista



TAD Lista

```
#define ItemType int
                                  Vamos identificar os atributos que
typedef struct{
                                  representarão a lista duplamente
}List;
                                  encadeada
List *createList ();
void initializeList(List *1);
int addLastList(List *1, ItemType e);
int addList(List* 1, ItemType e, int index);
int removeList(List* 1, int index, ItemType *e);
int removeElementList(List* 1, ItemType* e);
int getList(List* 1, int index, ItemType* e);
int setList(List* 1, int index, ItemType* e);
int indexOfList(List* 1, ItemType* e);
int containsList(List* 1, ItemType *e);
int sizeList(List* 1);
int isEmptyList(List* 1);
void printList(List* 1);
```

Encadeamento Duplo

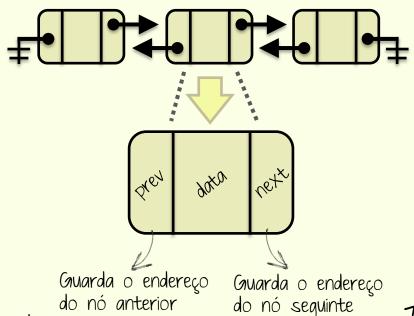




Encadeamento Duplo

- 🛞 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

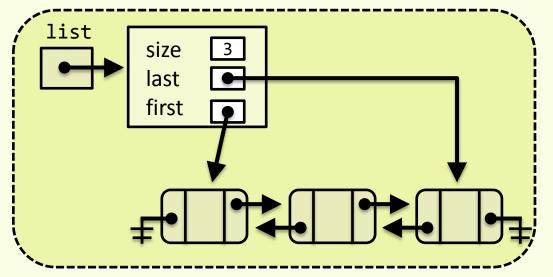
Lista Duplamente Encadeada





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
                                       list
                                                 size
typedef struct{
                                                 last
   Node *first;
                                                 first
   Node *last;
   int size:
}List;
List *createList ();
void initializeList(List *1);
int addList(List *1, ItemType e);
int addList(List* q, ItemType e, int index);
int removeList(List* q, int index, ItemType *e);
int removeList(List* q, ItemType* e);
                                                      typedef struct node{
int getList(List* q, int index, ItemType* e);
                                                          ItemType
                                                                       data;
int setList(List* q, int index, ItemType* e);
                                                          struct node *prev
int indexOfList(List* q, ItemType* e);
                                                          struct node *next:
int containsList(List* q, ItemType *e);
                                                      }Node;
int sizeList(List* q);
int isEmptyList(List* q);
void printList(List* q);
```

Simulação (**)



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



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

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```

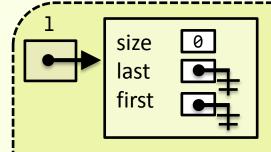






```
List *1 = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



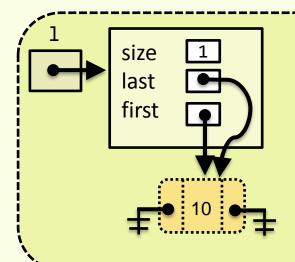






```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



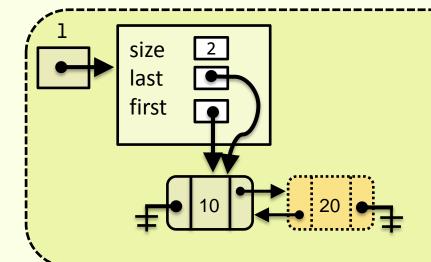






```
List *1 = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



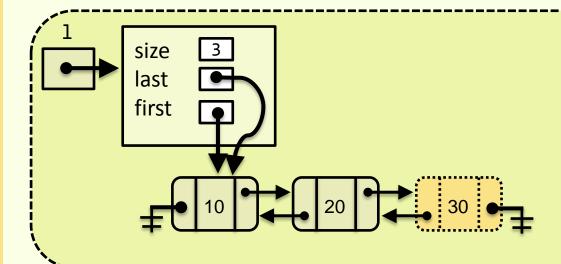






```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



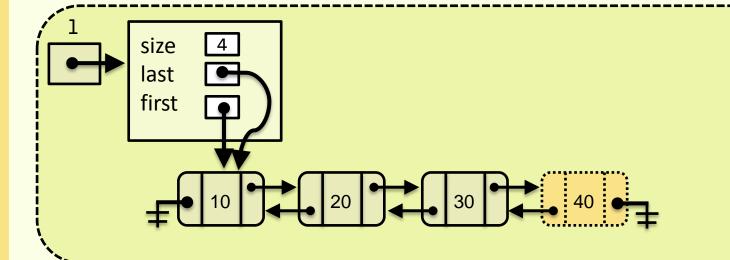






```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



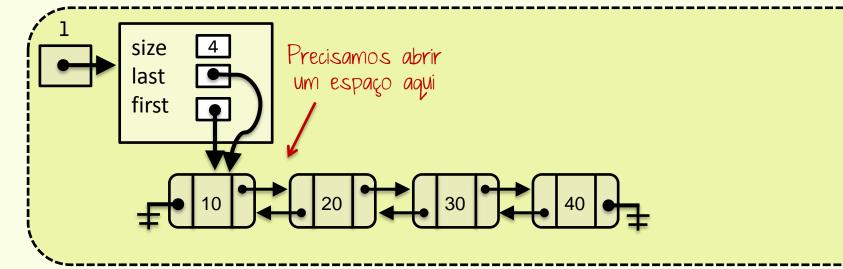






```
List *1 = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
ItemType elem removeList(1, removeList(1, removeList(1, no.8));
ItemType removed, n = 35;
ItemType removeList(1, removeList(1, no.8));
ItemType removed, n = 35;
ItemType removeList(1, no.8)
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



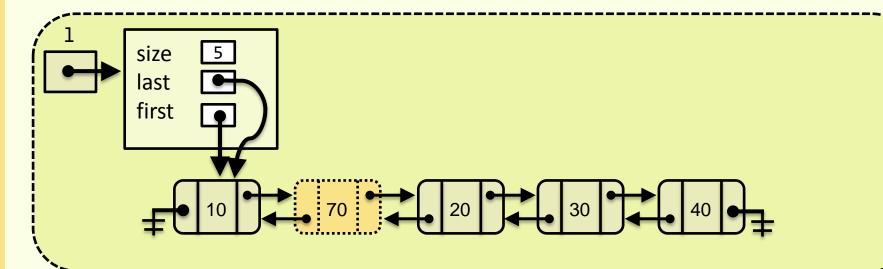






```
List *1 = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
ItemType removeList(1,10);
ItemType removed, n = 35;
ItemType removeList(1,10);
ItemType removed, n = 35;
ItemType removeList(1,10);
ItemType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

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



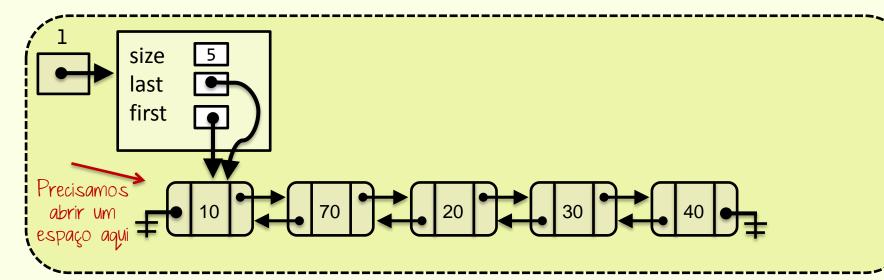






```
List *1 = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



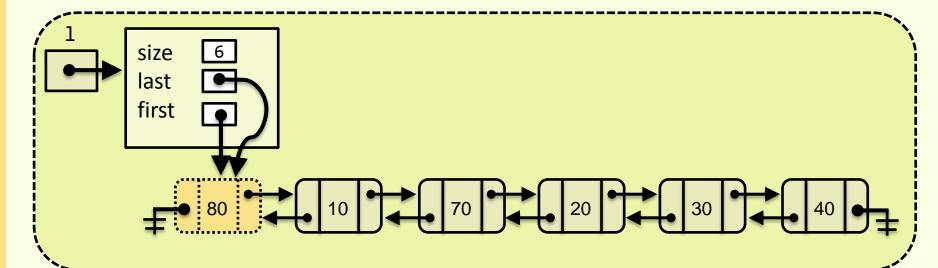






```
List *1 = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```



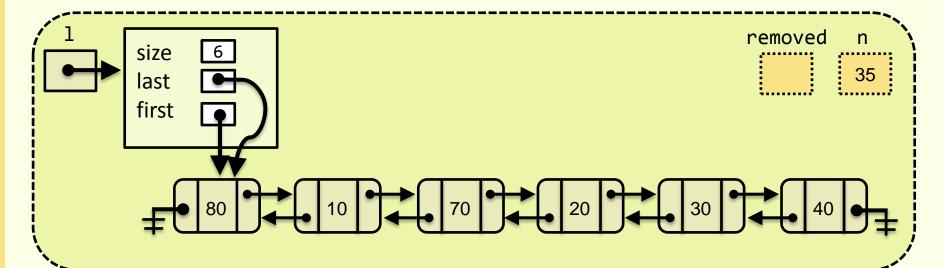






```
List *l = createList();
addLastList(l,10);
addLastList(l,20);
addLastList(l,30);
addLastList(l,40);
addList(l,70,1);
addList(l,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```

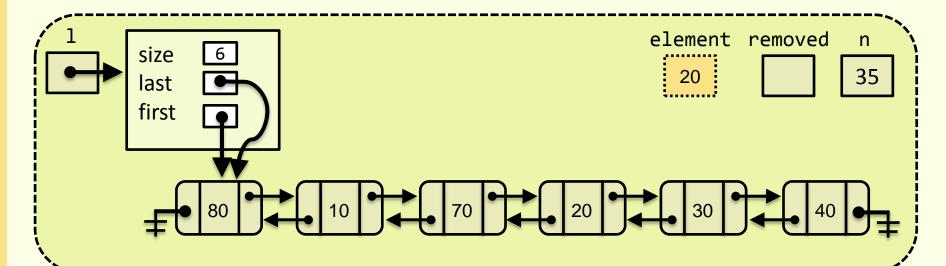








```
List *l = createList();
                                      ItemType element = 20;
                                      removeList(1,5,&removed);
addLastList(1,10);
                                      removeList(1,2,&removed);
addLastList(1,20);
                                      removeList(1,0,&removed);
addLastList(1,30);
                                      int i = indexOfList(1,&element);
addLastList(1,40);
addList(1,70,1);
                                      setList(1,0,&n);
addList(1,80,0);
                                      removeList(1,&element);
ItemType removed, n = 35;
                                      removeList(1,0,&removed);
```

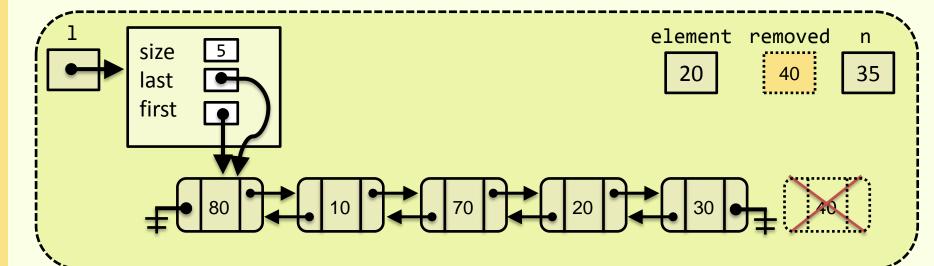








```
List *l = createList();
                                      ItemType element = 20;
                                      removeList(1,5,&removed);
addLastList(1,10);
                                      removeList(1,2,&removed);
addLastList(1,20);
                                      removeList(1,0,&removed);
addLastList(1,30);
                                      int i = indexOfList(1,&element);
addLastList(1,40);
addList(1,70,1);
                                      setList(1,0,&n);
addList(1,80,0);
                                      removeList(1,&element);
ItemType removed, n = 35;
                                      removeList(1,0,&removed);
```

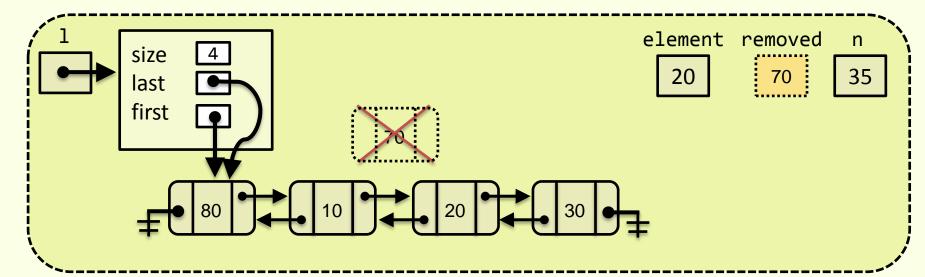








```
List *l = createList();
                                      ItemType element = 20;
                                      removeList(1,5,&removed);
addLastList(1,10);
                                      removeList(1,2,&removed);
addLastList(1,20);
addLastList(1,30);
                                      removeList(1,0,&removed);
                                      int i = indexOfList(1,&element);
addLastList(1,40);
addList(1,70,1);
                                      setList(1,0,&n);
addList(1,80,0);
                                      removeList(1,&element);
ItemType removed, n = 35;
                                      removeList(1,0,&removed);
```









```
List *l = createList();

addLastList(l,10);

addLastList(l,20);

addLastList(l,30);

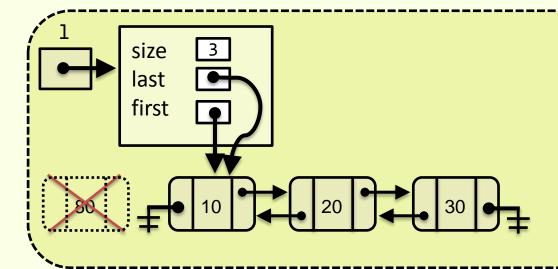
addLastList(l,40);

addList(l,70,1);

addList(l,80,0);

ItemType element removeList(l,5, removeList(l,2, removeList(l,2, removeList(l,0, setList(l,0, setL
```

ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);



element removed

20

80

35

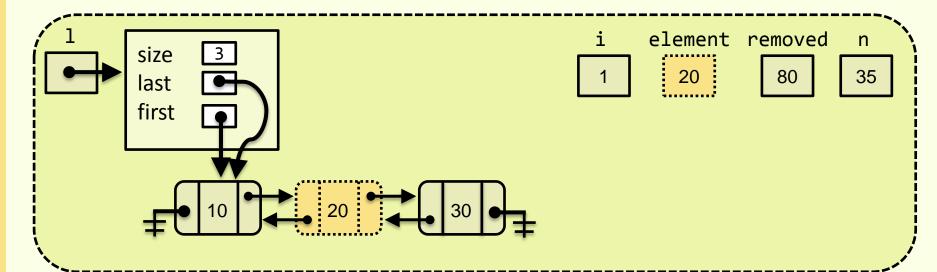
n







```
List *l = createList();
                                      ItemType element = 20;
                                      removeList(1,5,&removed);
addLastList(1,10);
                                      removeList(1,2,&removed);
addLastList(1,20);
                                      removeList(1,0,&removed);
addLastList(1,30);
addLastList(1,40);
                                      int i = indexOfList(1,&element);
addList(1,70,1);
                                      setList(1,0,&n);
addList(1,80,0);
                                      removeList(1,&element);
ItemType removed, n = 35;
                                      removeList(1,0,&removed);
```

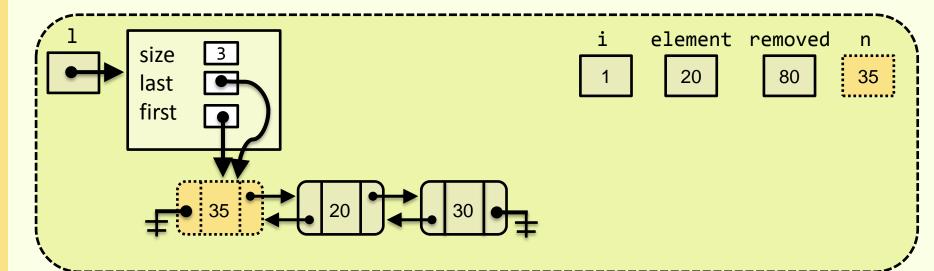








```
List *l = createList();
                                      ItemType element = 20;
                                      removeList(1,5,&removed);
addLastList(1,10);
                                      removeList(1,2,&removed);
addLastList(1,20);
                                      removeList(1,0,&removed);
addLastList(1,30);
addLastList(1,40);
                                      int i = indexOfList(1,&element);
addList(1,70,1);
                                      setList(1,0,&n);
addList(1,80,0);
                                      removeList(1,&element);
ItemType removed, n = 35;
                                      removeList(1,0,&removed);
```



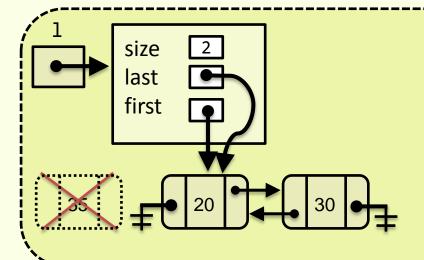


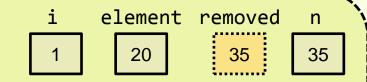




```
List *l = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
removeList(1,0,&removed);
```





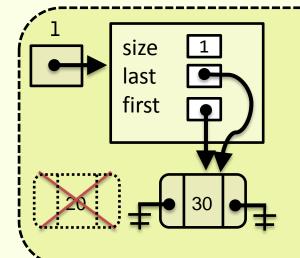


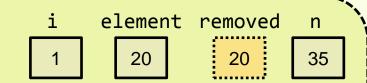




```
List *l = createList();
addLastList(1,10);
addLastList(1,20);
addLastList(1,30);
addLastList(1,40);
addList(1,70,1);
addList(1,80,0);
ItemType removed, n = 35;
```

```
ItemType element = 20;
removeList(1,5,&removed);
removeList(1,2,&removed);
removeList(1,0,&removed);
int i = indexOfList(1,&element);
setList(1,0,&n);
removeList(1,&element);
```





Implementação





Implementação

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

```
typedef struct node{
List *createList ();
                                                        ItemType
                                                                   data;
                                                        struct node *prev;
void initializeList(List *1);
                                                        struct node *next;
int addLastList(List *1, ItemType e);
                                                     }Node;
int addList(List* 1, ItemType e, int index);
int removeList(List* 1, int index, ItemType *e);
                                                     typedef struct{
                                                        Node *first;
int removeElementList(List* 1, ItemType* e);
                                                        Node *last;
int getList(List* 1, int index, ItemType* e);
                                                        int
                                                             size;
int setList(List* 1, int index, ItemType* e);
                                                     }List;
int indexOfList(List* 1, ItemType* e);
                                                     list
int containsList(List* 1, ItemType *e);
                                                             size
int sizeList(List* 1);
                                                             last
int isEmptyList(List* 1);
                                                             first
void printList(List* 1);
```

Implementação

LET'S DO IT





Referências