

Listas

Prof. Bruno Travençolo
(com adaptações feitas por Paulo H. R. Gabriel)

1

Listas

- ▶ Estrutura de dados linear usada para armazenar e organizar dados
- ▶ Sequência de elementos do mesmo tipo
- ▶ Já foi visto alguma estrutura semelhante?

2

Listas

- ▶ Qual a diferença da ED “Sequência de elementos do mesmo tipo” para um vetor?
- ▶ Temos que ver a lista como um TAD
 - ▶ Se é um TAD devemos definir operações sobre a lista
- ▶ Quais operações são esperadas em uma lista?



3

Listas

- ▶ Operações de manipulação da lista
 - ▶ Inserir um elemento na lista
 - ▶ Remover um elemento da lista
 - ▶ Buscar um elemento da lista
 - ▶ Verificar o tamanho da lista
- ▶ Operações relacionadas a estrutura da lista
 - ▶ Criar a lista
 - ▶ Destruir a lista



4

Exemplo: Lista de alunos

- ▶ Suponha que queremos guardar diferentes listas de alunos que participam de diferentes atividades na universidade
 - ▶ Lista de alunos que participam do grupo PET
 - ▶ Lista de alunos que participam de maratonas de programação
 - ▶ Lista de alunos da atlética
 - ▶ Lista de alunos de grupo de estudo em empreendedorismo

```
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
```

5

Lista de alunos

- ▶ Devemos criar um TAD para armazenar essas listas
- ▶ Para criar o TAD vamos precisar definir os dados e as operações que serão suportadas pelo TAD

```
ListaSequencial.h
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
```

6

```
//Arquivo ListaSequencial.h
#define MAX 100
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct lista Lista;
Lista* cria_lista();
void libera_lista(Lista* li);
int consulta_lista_pos(Lista* li, int pos, struct aluno *al);
int consulta_lista_mat(Lista* li, int mat, struct aluno *al);
int insere_lista_final(Lista* li, struct aluno al);
int insere_lista_inicio(Lista* li, struct aluno al);
int insere_lista_ordenada(Lista* li, struct aluno al);
int remove_lista(Lista* li, int mat);
int remove_lista_inicio(Lista* li);
int remove_lista_final(Lista* li);
int tamanho_lista(Lista* li);
int lista_cheia(Lista* li);
int lista_vazia(Lista* li);
void imprime_lista(Lista* li);
int remove_lista_otimizado(Lista* li, int mat);
```

► Fonte: <https://programacaodescomplicada.wordpress.com/complementar/>

7

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "ListaSequencial.h" //inclui os Protótipos
4
5  //Definição do tipo lista
6  struct lista{
7      int qtd;
8      struct aluno dados[MAX];
9  };
10
```

8

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "ListaSequencial.h" //inclui os Protótipos
4
5  //Definição do tipo lista
6  struct lista{
7      int qtd;
8      struct aluno dados[MAX];
9  };
10

```

Variável quantidade (qtd) é quem diferencia a lista de um vetor. Com ela controlamos *em tempo de execução* quantos elementos nossa lista possui

9

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "ListaSequencial.h" //inclui os Protótipos
4
5  //Definição do tipo lista
6  struct lista{
7      int qtd;
8      struct aluno dados[MAX];
9  };
10

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;

```

Nossa lista suporta elementos do tipo *struct aluno*. Não é genérica para outros tipos (assunto a ser abordado em outros cursos)

10

```

ListaSequencial.h
#define MAX 100

//Arquivo ListaSequencial.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "ListaSequencial.h" //inclui os Protótipos
4
5  //Definição do tipo lista
6  struct lista{
7      int qtd;
8      struct aluno dados[MAX];
9  };
10

```

Nossa lista pode ter no máximo MAX elementos (ou seja MAX struct aluno)

11

```

ListaSequencial.c
struct lista{
    int qtd;
    struct aluno dados[MAX];
};

ListaSequencial.h
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

12

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```



```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista() {
12     Lista *li;
13     li = (Lista*) ma
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Parameters: sem entrada
Return value: endereço de memória da lista criada dinamicamente. Em caso de erro retorna NULL

13

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```



```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista() {
12     Lista *li;
13     li = (Lista*) ma
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Cria uma variável local para armazenar um ponteiro para a lista

14

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```



```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Aloca a estrutura Lista na memória para ser usada pelo TAD

15

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```



```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Quantos bytes são alocados na heap?

16


```

13      li = (Lista*) malloc(sizeof(struct lista));

```

ListaSequencial.h

```

#define MAX 100

```

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```

Quantos bytes são alocados na heap?

qtd: 4 bytes
dados: 100xsizeof(struct aluno)

matricula: 4 bytes
n1, n2, n3: 12 bytes
nome: 30 bytes
Total: 46
** obs: assumindo que não há alinhamento. Com alinhamento o tamanho será 48 bytes

Total malloc: 4+100x46 = 4604 bytes

17

```

13      li = (Lista*) malloc(sizeof(struct lista));

```

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

Total malloc: 4+100x46 = 4604 bytes

Qual o valor de 'li'?

Resposta: é o endereço retornado pelo malloc. Neste exemplo é 48. Se não houvesse alocação li receberia NULL

Endereço	Nome variável	Tipo	bytes	Conteúdo
48	li→qtd	int	4	lixo
52	li→dados[0]	struct aluno	46	lixo
98	li→dados[1]	struct aluno	46	lixo
144	li→dados[2]	struct aluno	46	lixo
190	li→dados[3]	struct aluno	46	lixo
236	li→dados[4]	struct aluno	46	lixo
....
4652	li→dados[99]	struct aluno	46	lixo

18

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```



```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Testa se o malloc funcionou. Caso ele não funcione ele retornará NULL

19

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```



```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Sabendo que a alocação funcionou, devemos iniciar o TAD indicando que a lista possui 0 elementos

20

Iniciando a Lista

```

14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

Endereço	Nome variável	Tipo	bytes	Conteúdo
48	li->qtd	int	4	0
52	li->dados[0]	struct aluno	46	lixo
98	li->dados[1]	struct aluno	46	lixo
144	li->dados[2]	struct aluno	46	lixo
190	li->dados[3]	struct aluno	46	lixo
236	li->dados[4]	struct aluno	46	lixo
....
4652	li->dados[99]	struct aluno	46	lixo

21

ListaSequencial.c

```

struct lista{
    int qtd;
    struct aluno dados[MAX];
};

```

ListaSequencial.h

```

struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();

```

```

//Arquivo ListaSequencial.c
// cria_lista - realiza a inicialização do TAD lista
// Parameters:
// Return value: endereço de memória da lista criada dinamicamente
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }

```

Retornamos o ponteiro para a Lista criada; ou NULL em caso de falha (vem do próprio malloc)

22

No programa principal (que usa o TAD)

```
#include <stdio.h>
#include <stdlib.h>
#include "ListaSequencial.h"

int main() {
    Lista* lista_alunos_facom;
    lista_alunos_facom = cria_lista();
```

23

No programa principal (que usa o TAD)

```
#include <stdio.h>
#include <stdlib.h>
#include "ListaSequencial.h"
```

```
int main() {
    Lista* lista_a
    lista_alunos_facom = cria_lista();
```

Incluir o .h do TAD para ter acesso aos tipos de dados e operações do TAD

24

No programa principal (que usa o TAD)

```
#include <stdio.h>
#include <stdlib.h>
#include "ListaSequencial.h"
```

```
int main() {
    Lista* lista_alunos_facom;
    lista_alunos_facom = cria_lista();
}
```

Cria uma ponteiro para o TAD. Note que ainda não houve alocação de memória para o TAD em si

25

No programa principal (que usa o TAD)


```
#include <stdio.h>
#include <stdlib.h>
#include "ListaSequencial.h"
```

```
int main() {
    Lista* lista_alunos_facom;
    lista_alunos_facom = cria_lista();
}
```

Cria a lista, alocando a memória pré-definida e retorna o ponteiro (endereço) para a região alocada para o main.c


26

```
10
11 Lista* cria_lista(){
12     Lista *li;
13     li = (Lista*) malloc(sizeof(struct lista));
14     if(li != NULL)
15         li->qtd = 0;
16     return li;
17 }
18
19 void libera_lista(Lista* li){
20     free(li);
21 }
```



27

```
42
43 int insere_lista_final(Lista* li, struct aluno al){
44     if(li == NULL)
45         return -1;
46     if(li->qtd == MAX) //lista cheia
47         return -1;
48     li->dados[li->qtd] = al;
49     li->qtd++;
50     return 0;
51 }
```



28

Lista Sequencial

- ▶ Insere um elemento no final da lista
- ▶ Inicialmente vazia

MAX = 6

0	1	2	3	4	5

qtd = 0



29

Lista Sequencial

- ▶ Insere um elemento no final da lista
- ▶ Insere "A" - `insere_lista_final(A)`
 - ▶ insere no final da lista

A					
0	1	2	3	4	5

qtd = 1



30

Lista Sequencial

- ▶ Insere um elemento no final da lista
- ▶ Insere "A"
- ▶ Insere "B"

A	B				
0	1	2	3	4	5

qtd = 2



31

Lista Sequencial

- ▶ Insere um elemento no final da lista
- ▶ Insere "A"
- ▶ Insere "B"
- ▶ Insere "C"

A	B	C			
0	1	2	3	4	5

qtd = 3



32


```
53 int insere_lista_inicio(Lista* li, struct aluno al){
54     if(li == NULL)
55         return -1;
56     if(li->qtd == MAX)//lista cheia
57         return -1;
58     int i;
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
61     li->dados[0] = al;
62     li->qtd++;
63     return 0;
64 }
```

33

```
53 int insere_lista_inicio(Lista* li, struct aluno al){
54     if(li == NULL)
55         return -1;
56     if(li->qtd == MAX)//lista cheia
57         return -1;
58     int i;
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
61     li->dados[0] = al;
62     li->qtd++;
63     return 0;
64 }
```

34

Lista Sequencial

- ▶ `int` insere_lista_inicio(Lista* li, struct aluno al) {
- ▶ Insere "D"

A	B	C			
0	1	2	3	4	5

qtd = 3



35

Lista Sequencial

- ▶ `int` insere_lista_inicio(Lista* li, struct aluno al) {
- ▶ Insere "D"

i = 2

A	B	C			
0	1	2	3	4	5

qtd = 3

```
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
```

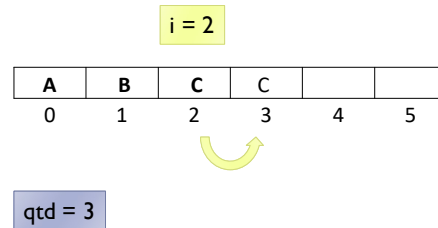


36

Lista Sequencial

► **int** insere_lista_inicio(Lista* li,
struct aluno al) {

► Insere "D"



```
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
```

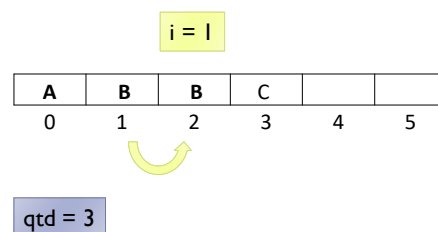


37

Lista Sequencial

► **int** insere_lista_inicio(Lista* li,
struct aluno al) {

► Insere "D"



```
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
```

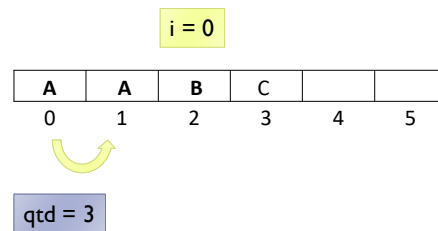


38

Lista Sequencial

► `int` insere_lista_inicio(Lista* li, struct aluno al) {

► Insere "D"



```
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
```

39

```
53 int insere_lista_inicio(Lista* li, struct aluno al){
54     if(li == NULL)
55         return -1;
56     if(li->qtd == MAX) //lista cheia
57         return -1;
58     int i;
59     for(i=li->qtd-1; i>=0; i--)
60         li->dados[i+1] = li->dados[i];
61     li->dados[0] = al;
62     li->qtd++;
63     return 0;
64 }
```

40

Lista Sequencial

► **int** insere_lista_inicio(Lista* li,
 struct aluno al) {

► Insere "D"

Insere D na primeira posição e aumenta qtd

D	A	B	C		
0	1	2	3	4	5

qtd = 4

```
61     li->dados[0] = al;
62     li->qtd++;
```

41

```
22
23  int consulta_lista_pos(Lista* li, int pos, struct aluno *al){
24      if(li == NULL || pos <= 0 || pos > li->qtd)
25          return -1;
26      *al = li->dados[pos-1];
27      return 0;
28  }
29
```

42

```

22
23 int consulta_lista_pos(Lista* li, int pos, struct aluno *al){
24     if(li == NULL || pos <= 0 || pos > li->qtd)
25         return -1;
26     *al = li->dados[pos-1];
27     return 0;
28 }
29

```

43

Lista Sequencial

- `consulta_lista_pos(Lista* li, int pos, struct aluno *al)`
- A lista inicia em 1 (um), mas o vetor em zero
- Exemplo:
 - Consulta o 2º elemento da lista

pos = 2

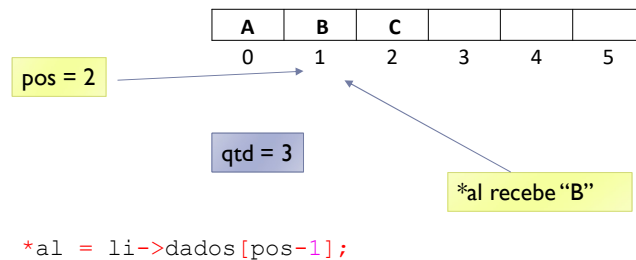
A	B	C			
0	1	2	3	4	5

qtd = 3

44

Lista Sequencial

- ▶ `consulta_lista_pos(Lista* li, int pos, struct aluno *al)`
- ▶ A lista inicia em 1 (um), mas o vetor em zero
- ▶ Exemplo:
 - ▶ Consulta o 2º elemento da lista



45

Lista Sequencial

- ▶ `consulta_lista_pos(Lista* li, int pos, struct aluno *al)`
- ▶ Cuidado com o parâmetro `struct aluno *al`
- ▶ O aluno já deve estar alocado no programa principal
 - ▶ Exemplo de chamada no programa principal

```
struct aluno aluno; // aloca variável no main()
consulta_lista_pos(li, 2, &aluno);
```

```
*al = li->dados[pos-1];
```

Passagem por referência de aluno

46

Lista Sequencial

- ▶ `consulta_lista_pos(Lista* li, int pos, struct aluno *al)`
- ▶ Cuidado com o parâmetro `struct aluno *al`
- ▶ O aluno já deve estar alocado no programa principal
 - ▶ Exemplo de chamada no programa principal

```
struct aluno aluno; // aloca variável no main()
consulta_lista_pos(li, 2, &aluno);
```

`*al` referencia a variável “aluno” que chamou a função consulta

```
*al = li->dados[pos-1];
```

47

```
30 int consulta_lista_mat(Lista* li, int mat, struct aluno *al) {
31     if(li == NULL)
32         return -1;
33     int i = 0;
34     while(i < li->qtd && li->dados[i].matricula != mat)
35         i++;
36     if(i == li->qtd) // elemento não encontrado
37         return -1;
38
39     *al = li->dados[i];
40     return 0;
41 }
```

48


```

30 int consulta_lista_mat(Lista* li, int mat, struct aluno *al){
31     if(li == NULL)
32         return -1;
33     int i = 0;
34     while(i < li->qtd && li->dados[i].matricula != mat)
35         i++;
36     if(i == li->qtd) //elemento nao encontrado
37         return -1;
38
39     *al = li->dados[i];
40     return 0;
41 }

```

49

- ▶ O loop faz a busca pelo número de matrícula desejado
- ▶ Exemplo:
 - ▶ Buscar aluno “B” (supor que a busca foi por número de matrícula)

```

33 int i = 0;
34 while(i < li->qtd && li->dados[i].matricula != mat)
35     i++;

```

i = 0

A	B	C			
0	1	2	3	4	5

qtd = 3

```

li->dados[0]
A != B ? True
Vai pro próximo

```

50

- ▶ O loop faz a busca pelo número de matrícula desejado
- ▶ Exemplo:
 - ▶ Buscar aluno “B” (supor que a busca foi por número de matrícula)

```

33  int i = 0;
34  while(i < li->qtd && li->dados[i].matricula != mat)
35      i++;

```

i = 1

A	B	C			
0	1	2	3	4	5

qtd = 3

li->dados[1]
B != B ? False
Achou o elemento, sai do loop

51

```

30  int consulta_lista_mat(Lista* li, int mat, struct aluno *al) {
31      if(li == NULL)
32          return -1;
33      int i = 0;
34      while(i < li->qtd && li->dados[i].matricula != mat)
35          i++;
36      if(i == li->qtd) // elemento não encontrado
37          return -1;
38
39      *al = li->dados[i];
40      return 0;
41  }

```

52

```

-----
i = 1 ; qtd = 3
36     if(i == li->qtd) //elemento nao encontrado
37         return -1;
38
39     *al = li->dados[i];

```

i = 1

A	B	C			
0	1	2	3	4	5

qtd = 3

►

53

```

30 int consulta_lista_mat(Lista* li, int mat, struct aluno *al){
31     if(li == NULL)
32         return -1;
33     int i = 0;
34     while(i < li->qtd && li->dados[i].matricula != mat)
35         i++;
36     if(i == li->qtd) //elemento nao encontrado
37         return -1;
38
39     *al = li->dados[i];
40     return 0;
41 }

```

Passagem por referência de aluno

al é uma passagem por referência de uma variável já alocada

```

struct aluno aluno;
consulta_lista_mat(li, "B", &aluno)

```

►

54

```

65
66  int insere_lista_ordenada(Lista* li, struct aluno al){
67      if(li == NULL)
68          return -1;
69      if(li->qtd == MAX)//lista cheia
70          return -1;
71      int k,i = 0;
72      while(i<li->qtd && li->dados[i].matricula < al.matricula)
73          i++;
74
75      for(k=li->qtd-1; k >= i; k--)
76          li->dados[k+1] = li->dados[k];
77
78      li->dados[i] = al;
79      li->qtd++;
80      return 0;
81  }

```

55

Lista Sequencial

- ▶ Remover um elemento
- ▶ `int remove_lista_inicio(Lista* li) {`
- ▶ Como remover do início?

A	B	C			
0	1	2	3	4	5

qtd = 3

```

122      for(k=0; k< li->qtd-1; k++)
123          li->dados[k] = li->dados[k+1];
124      li->qtd--;

```

56

Lista Sequencial

- ▶ Remover um elemento
- ▶ `int` remove_lista_inicio(Lista* li) {
- ▶ Como remover do início?

A	B	C			
0	1	2	3	4	5

qtd = 3

```

122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;

```

k = 0



57

Lista Sequencial

- ▶ Remover um elemento
- ▶ `int` remove_lista_inicio(Lista* li) {
- ▶ Como remover do início?

B	B	C			
0	1	2	3	4	5

qtd = 3

```

122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;

```

k = 0

0

1



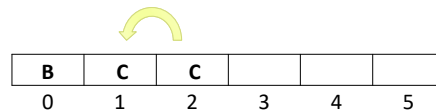
58

Lista Sequencial

- ▶ Remover um elemento

▶ **int** remove_lista_inicio(Lista* li) {

- ▶ Como remover do início?



qtd = 3

```

122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;

```

k = 1

1

2

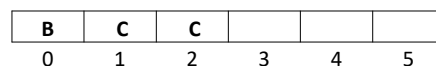
59

Lista Sequencial

- ▶ Remover um elemento

▶ **int** remove_lista_inicio(Lista* li) {

- ▶ Como remover do início?



qtd = 2

2

```

122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;

```

k = 2

qtd = 2

60

Lista Sequencial

► Remover um elemento

► **int** remove_lista_inicio(Lista* li) {

► Como remover do início?

B	C	C			
0	1	2	3	4	5

qtd = 2

O conteúdo não é apagado, somente é feito o deslocamento. A posição [2] do vetor nunca será lida pois qtd = 2

```
122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;
```

61

Lista Sequencial

► Remover um elemento

► **int** remove_lista_inicio(Lista* li) {

► Como remover do início?

B	C				
0	1	2	3	4	5

qtd = 2

A posição [2] fica disponível

```
122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;
```

62

Lista Sequencial

- ▶ Remover um elemento
- ▶ `int remove_lista_inicio(Lista* li) {`
- ▶ Como remover do início?

B	C				
0	1	2	3	4	5

qtd = 2

```

122     for(k=0; k< li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;

```

Custo computacional alto para uma operação simples (um remoção gera o deslocamento de todo o vetor)

63

Lista Sequencial

- ▶ Remover um elemento
- ▶ `int remove_lista_inicio(Lista* li) {`
- ▶ Como remover do início?

B	C				
0	1	2	3	4	5

qtd = 2

Observe que não utilizamos FREE !

Remover um elemento da lista sequencial não implica em deslocar memória para o elemento. Toda memória já foi pré-alocada na criação da lista

64


```
127
128 int remove_lista_final(Lista* li){
129     if(li == NULL)
130         return -1;
131     if(li->qtd == 0)
132         return -1;
133     li->qtd--;
134     return 0;
135 }
```

65

```
127
128 int remove_lista_final(Lista* li){
129     if(li == NULL)
130         return -1;
131     if(li->qtd == 0)
132         return -1;
133     li->qtd--;
134     return 0;
135 }
```

66

Lista Sequencial

- ▶ Insere um elemento na fila
- ▶ Insere "A"
- ▶ Insere "B"
- ▶ Insere "C"

A	B	C			
0	1	2	3	4	5

qtd = 3



67

Lista Sequencial

▶ **int** remove_lista_final(Lista* li) {

A	B	C			
0	1	2	3	4	5

qtd = 2

O conteúdo não é apagado, somente é diminuído o valor de qte. Após isso, a posição [2] do vetor nunca será lida pois qtd = 2

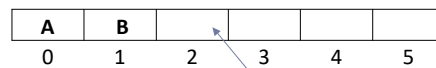
li->qtd--;



68

Lista Sequencial

```
► int remove_lista_final(Lista* li){
```



```
    qtd = 2
```


```
    li->qtd--;
```

69

```
82
83 int remove_lista(Lista* li, int mat){
84     if(li == NULL)
85         return -1;
86     if(li->qtd == 0)
87         return -1;
88     int k,i = 0;
89     while(i<li->qtd && li->dados[i].matricula != mat)
90         i++;
91     if(i == li->qtd) //elemento nao encontrado
92         return 0;
93
94     for(k=i; k< li->qtd-1; k++)
95         li->dados[k] = li->dados[k+1];
96     li->qtd--;
97     return 0;
98 }
```


70

```
100 int remove_lista_otimizado(Lista* li, int mat){
101     if(li == NULL)
102         return -1;
103     if(li->qtd == 0)
104         return -1;
105     int i = 0;
106     while(i < li->qtd && li->dados[i].matricula != mat)
107         i++;
108     if(i == li->qtd) //elemento nao encontrado
109         return 0;
110
111     li->qtd--;
112     li->dados[i] = li->dados[li->qtd];
113     return 0;
114 }
```



71

```
115
116 int remove_lista_inicio(Lista* li){
117     if(li == NULL)
118         return -1;
119     if(li->qtd == 0)
120         return -1;
121     int k = 0;
122     for(k=0; k < li->qtd-1; k++)
123         li->dados[k] = li->dados[k+1];
124     li->qtd--;
125     return 0;
126 }
```



72

```

136
137 int tamanho_lista(Lista* li){
138     if(li == NULL)
139         return -1;
140     else
141         return li->qtd;
142 }
143
144 int lista_cheia(Lista* li){
145     if(li == NULL)
146         return -1;
147     return (li->qtd == MAX);
148 }
149
150 int lista_vazia(Lista* li){
151     if(li == NULL)
152         return -1;
153     return (li->qtd == 0);
154 }

```

73

```

155
156 int imprime_lista(Lista* li){
157     if(li == NULL)
158         return -1;
159     int i;
160     for(i=0; i< li->qtd; i++){
161         printf("Matricula: %d\n",li->dados[i].matricula);
162         printf("Nome: %s\n",li->dados[i].nome);
163         printf("Notas: %f %f %f\n",li->dados[i].n1,
164                                     li->dados[i].n2,
165                                     li->dados[i].n3);
166         printf("-----\n");
167     } return 0;
168 }

```

74

```
#include <stdio.h>
#include <stdlib.h>
#include "ListaSequencial.h"
int main(){
    struct aluno a[4] = {{2,"Andre",9.5,7.8,8.5},
                        {4,"Ricardo",7.5,8.7,6.8},
                        {1,"Bianca",9.7,6.7,8.4},
                        {3,"Ana",5.7,6.1,7.4}};

    Lista* li = cria_lista();
    int i;
    for(i=0; i < 4; i++){
        insere_lista_ordenada(li,a[i]);

        imprime_lista(li);
        printf("\n\n\n\n");

        for(i=0; i < 5; i++){
            if (remove_lista_otimizado(li,i)==-1)
                printf("Erro\n");

            imprime_lista(li);
            printf("\n\n\n\n");
        }

        libera_lista(li);
        system("pause");
        return 0;
    }
```