

ROTEIRO 4 – LUCAS EDUARDO LEITE COSTA

1 -

```
C main.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  int main(){
6      Lista *l = criaLista();
7      int elemento_x = 2, elemento_y = 3;
8      insereInicio(l, elemento_x);
9      insereFim(l, elemento_y);
10     printf("Tamanho da lista : %d\n", tamanhoLista(l));
11     if(-1 == procura(l, 5))
12         printf("Elemento não encontrado na lista\n");
13     int x = 2;
14     printf("A posição do elemento %d é %d\n", x, procura(l, x));
15     insere_ordenada(l, 1);
16     remove_ocorrente(l, 3);
17     imprimeLista(l);
18     destroiLista(l);
19     return 0;
20 }
21
22
```

```
C lista.h > remove_ocorrente(Lista *, int)
1  #ifndef LISTA_H
2  #define LISTA_H
3  #define MAX 100
4  typedef struct {
5      int dados [MAX];
6      int tam;
7  }Lista;
8  Lista* criaLista();
9  void destroiLista ( Lista * li);
10 int tamanhoLista ( Lista * li);
11 int listaCheia ( Lista * li);
12 int listaVazia ( Lista * li);
13 int insereFim ( Lista *li, int elem );
14 int insereInicio ( Lista *li, int elem );
15 int removeFim ( Lista *li);
16 int removeInicio ( Lista *li);
17 int imprimeLista ( Lista * li);
18 int procura( Lista *li, int x);
19 int insere_ordenada(Lista *li, int x);
20 int remove_ocorrente(Lista *li, int x);
21
22 #endif
```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPURACÃO

```
[lucascosta@fedora roteiro_4]$ make
gcc -c lista.c
gcc main.o lista.o -o main
[lucascosta@fedora roteiro_4]$ ./main
Elemento inserido com sucesso!
Elemento inserido com sucesso!
Tamanho da lista : 2
Elemento não encontrado na lista
A posição do elemento 2 é 0
Elemento inserido com sucesso!
Elementos : 1 2
[lucascosta@fedora roteiro_4]$
```

```

C listac > @imprimeLista*
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 Lista *criaLista(){
6     Lista *li;
7     li = (Lista *)malloc( sizeof ( Lista ));
8     if(li != NULL )
9         li->tam = 0;
10    return li;
11 }
12
13 void destruiLista ( Lista * li){
14     if(li != NULL){
15         free(li);
16     }
17 }
18
19 int tamanhoLista ( Lista * li){
20     if(li == NULL ){
21         return -1;
22     }
23     return li -> tam;
24 }
25
26 int listaCheia ( Lista * li){
27     if(li == NULL )
28         return -1;
29     return (li -> tam == MAX);
30 }
31
32 int listaVazia ( Lista * li){
33     if(li == NULL ){
34         return -1;
35     }
36     return (li -> tam == 0);
37 }
38
39 int insereFim ( Lista *li , int elem ){
40     if(li == NULL )
41         return 0;
42     if (!listaCheia (li)){
43         li -> dados [li ->tam] = elem;
44         li ->tam++;
45         printf ("Elemento inserido com sucesso!\n"); return 1;
46     } else {
47         printf ("Elemento nao inserido - Lista Cheia!\n"); return 0;
48     }
49 }
50
51 int insereInicio ( Lista *li , int elem ){
52     if(li == NULL )
53         return 0;
54     if (!listaCheia (li)){
55         for(int i=li ->tam -1; i >=0; i--){
56             li -> dados [i+1] = li -> dados [i];
57             li -> dados [0] = elem;
58             li ->tam ++;
59             printf ("Elemento inserido com sucesso!\n"); return 1;
60         }
61     }
62 }

```

```

C listac > ...
49
50 int insereInicio ( Lista *li , int elem ){
51     if(li == NULL )
52         return 0;
53     if (!listaCheia (li)){
54         for(int i=li ->tam -1; i >=0; i--){
55             li -> dados [i+1] = li -> dados [i];
56             li -> dados [0] = elem;
57             li ->tam ++;
58             printf ("Elemento inserido com sucesso!\n"); return 1;
59         } else {
60             printf ("Elemento nao inserido - Lista Cheia!\n"); return 0;
61         }
62     }
63 }
64
65 int removeFim ( Lista *li){
66     if(li == NULL )
67         return 0;
68     if(! listaVazia (li)){
69         li ->tam--;
70         printf (" Elemento removido com sucesso!\n");
71         return 1;
72     } else {
73         printf (" Nenhum elemento removido - Lista Vazia!\n");
74         return 0;
75     }
76 }
77
78 int removeInicio ( Lista *li){
79     if(li == NULL )
80         return 0;
81     if (! listaVazia (li)){
82         for(int i=0; i<li ->tam -1; i++){
83             li -> dados [i] = li -> dados [i+1];
84             li ->tam --;
85             printf (" Elemento removido com sucesso!\n");
86             return 1;
87         } else {
88             printf (" Nenhum elemento removido - Lista Vazia!\n");
89             return 0;
90         }
91     }
92 }
93
94 int imprimeLista ( Lista * li){
95     if(li == NULL )
96         return 0;
97     if(listaVazia (li)){
98         printf ("Lista vazia!\n"); return 0;
99     }
100    printf ("Elementos : ");
101    for(int i = 0; i<li->tam; i++) {
102        printf ("%d ", li -> dados [i] );
103    }
104    printf ("\n");
105    return 1;
106 }
107
108 int removeLista (Lista *li, int elem)

```

C lista.c > imprimeLista(Lista *)

```
104 }
105
106 int procura( Lista *li, int x){
107     if (li == NULL) {
108         return 0;
109     }
110     for(int i = 0; i< li->tam; i++){
111         if(li -> dados[i] == x)
112             return i;
113     }
114     return -1;
115 }
116
117 int insere_ordenada(Lista *li, int x) {
118     if (li == NULL)
119         return 0;
120
121     if (listaCheia(li)){
122         printf("Elemento não inserido - Lista Cheia!\n");
123         return 0;
124     }
125
126     int i;
127     for (i = 0; i < li->tam && x > li->dados[i]; i++){
128     }
129     for (int j = li->tam; j > i; j--) {
130         li->dados[j] = li->dados[j-1];
131     }
132
133     li->dados[i] = x;
134     li->tam++;
135     printf("Elemento inserido com sucesso!\n");
136     return 1;
137 }
138
139 int remove_ocorrente( Lista *li, int x){
140     if (li == NULL)
141         return 0;
142
143     for(int i = 0; i<li->tam; i++){
144         if(x == li->dados[i]){
145             for(int j = i; j<li->tam-1; j++){
146                 li->dados[j] = li->dados[j+1];
147             }
148             li->tam --;
149             return 0;
150         }
151     }
152 }
```

```
2 > C main.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  int main(){
6      Lista li = criaLista();
7      insere_ordenado(li,5);
8      insere_ordenado(li,7);
9      insere_ordenado(li,2);
10     insere_ordenado(li,8);
11     remove_ocorrente(li,8);
12     if(!procura(li,15))
13         printf("Elemento não pertence a lista\n");
14     if(procura(li,5))
15         printf("Elemento encontrado!\n");
16     imprimeLista(li);
17     destroiLista(li);
18     return 0;
19
20 }

2 > C lista.h > insere_ordenado(Lista, int)
1  #ifndef LISTA_H
2  #define LISTA_H
3
4  typedef struct NO {
5      int info;
6      struct NO* prox;
7  } NO;
8
9  typedef struct NO* Lista;
10 NO* alocarNO();
11 int procura(Lista li, int x);
12 int tamanho(Lista li);
13 Lista criaLista();
14 int listaVazia(Lista li);
15 void imprimeLista(Lista li);
16 void remove_ocorrente(Lista lista, int x);
17 void insere_ordenado(Lista lista, int x);
18 void destroiLista(Lista * li);
19 void liberarNO (NO* atual);
20 #endif
21
```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPURAÇÃO

```
[lucascosta@fedora 2]$ ./main
Elemento não pertence a lista
Elemento encontrado!
Elementos: 2 5 7
[lucascosta@fedora 2]$
```

```

2 > C lista.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 Lista criaLista() {
6     Lista li = (Lista)malloc(sizeof(struct NO));
7     if (li != NULL)
8         li->prox = NULL;
9     return li;
10 }
11
12 NO* alocarNO() {
13     return (NO*)malloc(sizeof(NO));
14 }
15
16 void insere_ordenado(Lista lista, int n) {
17     if (lista == NULL)
18         return;
19
20     NO* novo = alocarNO();
21     if (novo == NULL)
22         return;
23
24     novo->info = n;
25
26     NO* atual = lista->prox;
27     NO* anterior = NULL;
28
29     while (atual != NULL && n > atual->info) {
30         anterior = atual;
31         atual = atual->prox;
32     }
33
34     if (anterior == NULL) {
35         novo->prox = lista->prox;
36         lista->prox = novo;
37     } else {
38         novo->prox = anterior->prox;
39         anterior->prox = novo;
40     }
41 }
42
43 void liberarNO (NO* atual){
44     free(atual);
45 }
46
47 void remove_ocorrente(Lista lista, int n) {
48     if (lista == NULL || lista->prox == NULL)

```

```

46
47 void remove_ocorrente(Lista lista, int n) {
48     if (lista == NULL || lista->prox == NULL)
49         return;
50
51     NO* atual = lista->prox;
52     NO* anterior = NULL;
53
54     while (atual != NULL) {
55         if (atual->info == n){
56
57             if (anterior == NULL) {
58                 lista->prox = atual->prox;
59             } else {
60                 anterior->prox = atual->prox;
61             }
62             free(atual);
63             return;
64         }
65         anterior = atual;
66         atual = atual->prox;
67     }
68 }
69
70
71
72 int listaVazia(Lista li) {
73     if (li == NULL || li->prox == NULL)
74         return 1;
75     return 0;
76 }
77
78 void imprimeLista(Lista li) {
79     if (li == NULL)
80         return;
81     if (listaVazia(li)) {
82         printf("Lista vazia!\n");
83         return;
84     }
85     printf("Elementos: ");
86     NO* aux = li->prox;
87     while (aux != NULL) {
88         printf("%d ", aux->info);
89         aux = aux->prox;
90     }
91     printf("\n");
92 }
93

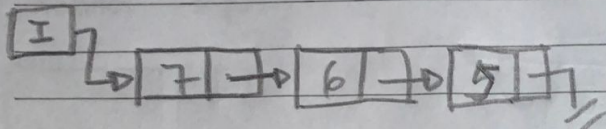
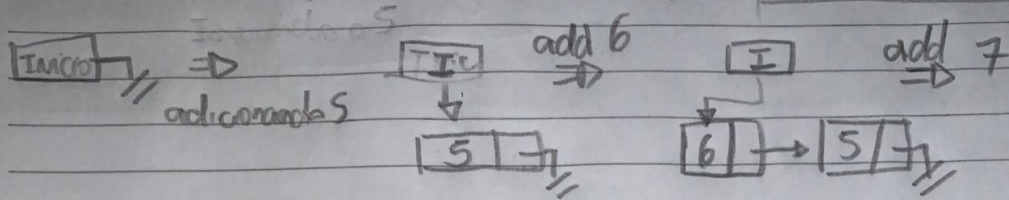
```

2 > C lista.c > ...

```
90     }
91     printf("\n");
92 }
93
94
95 int tamanho(Lista li) {
96     int tam = 0;
97     if (li == NULL) {
98         return tam;
99     }
100     NO* aux = li->prox;
101     while (aux != NULL) {
102         tam++;
103         aux = aux->prox;
104     }
105     return tam;
106 }
107
108 int procura(Lista li, int x) {
109     if (li == NULL)
110         return 0;
111     NO* aux = li->prox;
112     while (aux != NULL) {
113         if (aux->info == x) {
114             return 1;
115         }
116         aux = aux->prox;
117     }
118     return 0;
119 }
120
121 void destroiLista( Lista * li){
122     if(li != NULL ){
123         NO* aux;
124         while ((* li) != NULL ){
125             aux = *li;
126             *li = (* li) ->prox ;
127             liberarNO (aux);
128         }
129         free(li);
130     }
131 }
```

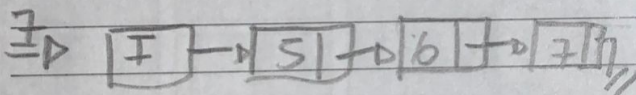
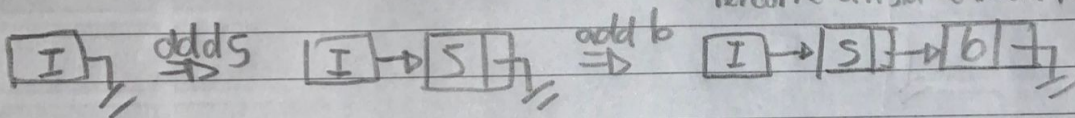

Inserção de 3 elementos no início
Inserido 5, 6 e 7 numa lista vazia

Notação = [I] = Início
Elemento = [Info/Prox]

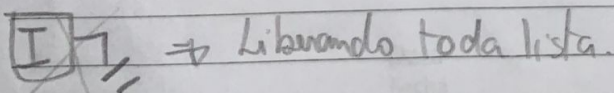
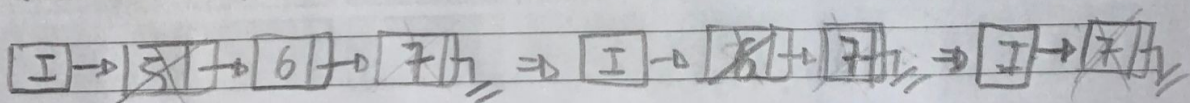


As mesmas inserções no fim

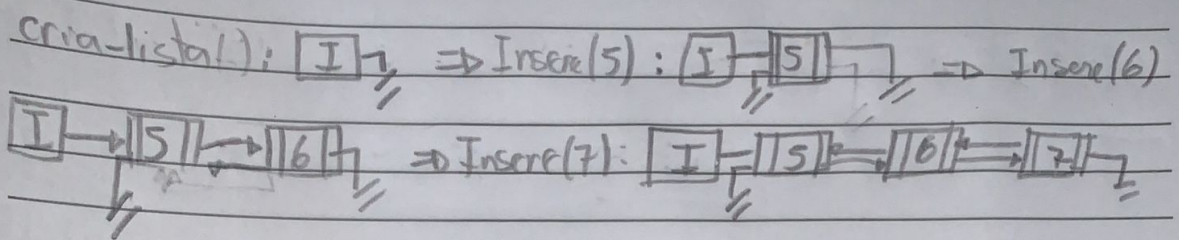
Percorre a lista e insere no final



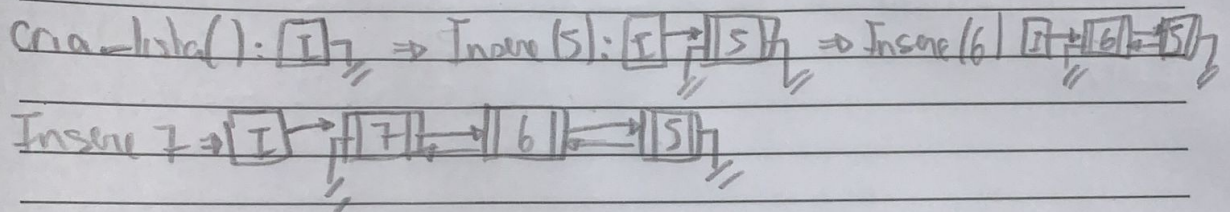
Destruindo - numa lista, liberando cada nó



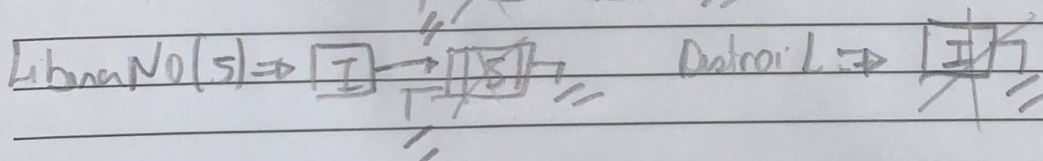
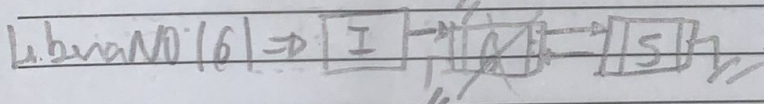
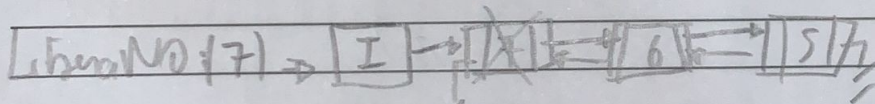
Inserção no Final



Inserção no Início



Destruição da lista




```
3 > C main.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 int main(){
6     Lista *li = criaLista();
7     insere_ordenado(li,6);
8     insere_ordenado(li,1);
9     insere_ordenado(li,5);
10    insere_ordenado(li,9);
11    insere_ordenado(li,7);
12    imprimeLista(li);
13    remove_ocorrente(li,5);
14    imprimeLista(li);
15    printf("Tamanho da lista: %d\n", tamanhoLista(li));
16    if(procura(li,9))
17        printf("0 elemento está na lista!\n");
18    destroiLista(li);
19    return 0;
20 }
```

```
3 > C lista.h > ...
1 # ifndef LISTA_H
2 # define LISTA_H
3
4 typedef struct NO{
5     int info ;
6     struct NO* ant;
7     struct NO* prox ;
8 }NO;
9 typedef struct NO* Lista ;
10 #endif
11
12 Lista *criaLista();
13 int listaVazia(Lista * li);
14 NO* alocarNO();
15 void liberarNO(NO* q);
16 int insereIni(Lista * li , int elem );
17 int insereFim(Lista * li , int elem );
18 int removeIni(Lista * li);
19 int removeFim(Lista * li);
20 void imprimeLista(Lista * li);
21 void destroiLista(Lista * li);
22 int tamanhoLista (Lista * li);
23 int procura(Lista *li, int elem);
24 int insere_ordenado(Lista * li , int elem );
25 int remove_ocorrente(Lista * li, int elem);
26
27
```

PROBLEMAS SAÍDA **TERMINAL** PORTAS CONSOLE DE DEPUÇÃO

```
[lucascosta@fedora 3]$ ./main
Elementos: 1 5 6 7 9
Elementos: 1 6 7 9
Tamanho da lista: 4
0 elemento está na lista!
[lucascosta@fedora 3]$
```

```
3 > C lista.c > remove_ocorrente(Lista *, int)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5
6 Lista * criaLista () {
7     Lista *li;
8     li = ( Lista *) malloc ( sizeof ( Lista ));
9     if(li != NULL )
10         *li = NULL ;
11     return li;
12 }
13
14
15 int listaVazia ( Lista * li){
16     if(li == NULL )
17         return 1;
18     if ( * li == NULL )
19         return 1;
20     return 0;
21 }
22
23 NO* alocarNO () {
24     return (NO *) malloc ( sizeof (NO));
25 }
26
27 void liberarNO (NO* q){
28     free (q);
29 }
30
31
32 int insereIni( Lista * li , int elem ){
33     if(li == NULL ) return 0;
34     NO* novo = alocarNO () ;
35     if( novo == NULL ) return 0;
36     novo -> info = elem ;
37     novo -> prox = *li;
38     novo -> ant = NULL ;
39     if (! listaVazia (li))
40         (* li) -> ant = novo ;
41     *li = novo ;
42     return 1;
43 }
44
45 int insereFim ( Lista * li , int elem ){
46     if(li == NULL ) return 0;
47     NO* novo = alocarNO () ;
48
```

```
3 > C lista.c > remove_ocorrente(Lista *, int)
45
46 int insereFim ( Lista * li , int elem ){
47     if(li == NULL ) return 0;
48     NO* novo = alocarNO () ;
49     if( novo == NULL ) return 0;
50     novo -> info = elem ;
51     novo -> prox = NULL ;
52     if( listaVazia (li)){
53         novo -> ant = NULL ;
54         *li = novo ;
55     } else {
56         NO* aux = *li;
57         while (aux -> prox != NULL )
58             aux = aux -> prox ;
59         aux -> prox = novo ;
60         novo -> ant = aux;
61     }
62     return 1;
63 }
64
65
66 int removeIni ( Lista * li){
67     if(li == NULL ) return 0;
68     if( listaVazia (li)) return 0;
69     NO* aux = *li;
70     *li = aux -> prox ;
71     if(aux -> prox != NULL )
72         aux -> prox -> ant = NULL ;
73     liberarNO (aux);
74     return 1;
75 }
76
77
78 int removeFim ( Lista * li){
79     if(li == NULL ) return 0;
80     if( listaVazia (li)) return 0;
81     NO * aux = *li;
82     while (aux -> prox != NULL )
83         aux = aux -> prox ;
84     if(aux -> ant == NULL )
85         *li = aux -> prox ;
86     else
87         aux -> ant -> prox = NULL ;
88     liberarNO (aux);
89     return 1;
90 }
91
92
```

3 > C lista.c > remove_ocorrente(Lista*,int)

```
91
92
93 void imprimeLista ( Lista * li){
94     if(li == NULL ) return ;
95     if( listaVazia (li)){
96         printf (" Lista vazia!\n");
97         return ;
98     }
99     printf (" Elementos: ");
100     NO* aux = *li;
101     while ( aux != NULL ){
102         printf ("%d ", aux->info );
103         aux = aux -> prox ;
104     }
105     printf ("\n");
106 }
107
108
109 void destroiLista ( Lista * li){
110     if(li != NULL ){
111         NO* aux;
112         while ((* li) != NULL ){
113             aux = *li;
114             *li = (* li) ->prox ;
115             liberarNO (aux);
116         }
117         free (li);
118     }
119 }
120
121
122 int tamanhoLista ( Lista * li){
123     int tam = 0;
124     if(li == NULL )
125         return 0;
126     if(listaVazia(li))
127         return 0;
128     NO* aux = *li;
129     while (aux->prox != NULL ){
130         aux = aux->prox ;
131         tam++;
132     }
133     tam++;
134     return tam;
135 }
136
137
138 int procura(Lista * li, int elem){
```

3 > C lista.c > procura(Lista*,int)

```
138 int procura(Lista * li, int elem){
139     if(li == NULL ) return 0;
140     if( listaVazia (li)) return 0;
141     NO * aux = *li;
142     while (aux->prox != NULL ){
143         if(aux->info == elem){
144             return 1;
145         }
146         aux = aux->prox ;
147     }
148     if(aux->info == elem){
149         return 1;
150     }
151     return 0;
152 }
153
154
155 int insere_ordenado ( Lista * li , int x ){
156     if (insereIni(li, x)) {
157         NO *atual = *li;
158         NO *anterior = NULL;
159         int trocou = 0;
160
161         while (atual->prox != NULL && atual->info > atual->prox->info) {
162             int guardaElem = atual->info;
163             atual->info = atual->prox->info;
164             atual->prox->info = guardaElem;
165             anterior = atual;
166             atual = atual->prox;
167             trocou = 1;
168         }
169         if (trocou && anterior != NULL) {
170             anterior = NULL;
171             atual = *li;
172             while (atual->prox != NULL && atual->info >
173                 atual->prox->info){
174                 int guardaElem = atual->info;
175                 atual->info = atual->prox->info;
176                 atual->prox->info = guardaElem;
177                 anterior = atual;
178                 atual = atual->prox;
179             }
180         }
181         return 1;
182     } else {
183         return 0;
184     }
```

```
int remove_ocorrente(Lista * li, int elem){
    if(procura(li, elem)){
        NO * aux = *li;
        if(aux->info == elem){
            removeIni(li);
            return 1;
        }
        while (aux->prox != NULL ){
            if(aux->info == elem){
                aux->ant->prox = aux->prox;
                aux->prox->ant = aux->ant;
                liberarNO(aux);
                return 1;
            }
            aux = aux->prox ;
        }
        if(aux->info == elem){
            removeFim(li);
            return 1;
        }
        else{
            return 0;
        }
    }
    else{
        return 0;
    }
}
```

.4

```
4 > C main.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 int main(){
6     Lista* l = criaLista();
7     insereFim(l,7);
8     insereFim(l,9);
9     insereFim(l,1);
10    insereFim(l,8);
11    if(!procura(l,25))
12        printf("Elemento não encontrado!\n");
13    if(procura(l,8))
14        printf("Elemento pertence a lista\n");
15    printf("Tamanho da lista: %d\n", tamanho(l));
16    imprimeLista(l);
17    destroiLista(l);
18 }
```

```
4 > C lista.h > LISTA
1 #ifndef LISTA_H
2 #define LISTA_H
3 #include <stdio.h>
4 #include <stdlib.h>
5 #define MAX 100
6
7 typedef struct NO {
8     int info;
9     struct NO* prox;
10 } NO;
11 typedef struct NO* Lista;
12 Lista* criaLista();
13 int tamanho(Lista* lista);
14 int procura(Lista* lista, int n);
15 void imprimeLista(Lista* li);
16 NO* alocarNO(int n);
17 int insereFim(Lista* li, int n);
18 void destroiLista(Lista* *li);
19 void liberarNO(NO* no);
20 #endif
21
22
```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPUÇÃO

```
[lucascosta@fedora 4]$ ./main
Elemento não encontrado!
Elemento pertence a lista
Tamanho da lista: 4
Elementos na lista: 7 9 1 8
[lucascosta@fedora 4]$
```

```
4 > C lista.c > procura(Lista*, int)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 Lista* criaLista () {
6     Lista *li;
7     li = (Lista *) malloc(sizeof(Lista ));
8     if(li != NULL )
9         *li = NULL ;
10    return li;
11 }
12
13 void imprimeLista(Lista* li) {
14     if (*li == NULL) {
15         printf("Lista vazia!\n");
16         return;
17     }
18     printf("Elementos na lista: ");
19     NO* aux = *li;
20     while(aux->prox != *li){
21         printf("%d ", aux->info);
22         aux = aux->prox;
23     }
24     printf("%d ", aux->info);
25     printf("\n");
26 }
27
28 NO* alocarNO(int n) {
29     NO* novo = (NO*)malloc(sizeof(NO));
30     if (novo == NULL) {
31         return NULL;
32     }
33
34     novo->info = n;
35     novo->prox = NULL;
36     return novo;
37 }
38
```

```
4 > C lista.c > ...
38
39 int insereFim(Lista* li, int n) {
40     NO* novo = alocarNO(n);
41     if (novo == NULL) {
42         return 0;
43     }
44
45     if (*li == NULL) {
46         *li = novo;
47         novo->prox = novo;
48     } else {
49         NO* aux = *li;
50         while (aux->prox != *li) {
51             aux = aux->prox;
52         }
53         aux->prox = novo;
54         novo->prox = *li;
55     }
56
57     return 1;
58 }
59
60 int tamanho(Lista* lista) {
61     if (*lista == NULL) {
62         return 0;
63     }
64     int tam = 1;
65     NO* atual = *lista;
66     atual = atual->prox;
67     while (atual != *lista){
68         tam++;
69         atual = atual->prox;
70     }
71
72     return tam;
73 }
74
```

```
4 > C lista.c > procura(Lista *, int)
72     return tam;
73 }
74
75 int procura(Lista* lista, int x) {
76     if (*lista == NULL) {
77         return 0;
78     }
79
80     NO* atual = *lista;
81     do {
82         if (atual->info==x) {
83             return 1;
84         }
85         atual = atual->prox;
86     } while (atual != *lista);
87
88     return 0;
89 }
90
91 void destroiLista(Lista *li){
92     if(li != NULL && (* li) != NULL ){
93         NO* prim , *aux ;
94         prim = *li;
95         *li = (* li) ->prox ;
96         while ((* li) != prim ){
97             aux = *li;
98             *li = (* li) ->prox ;
99             liberarNO (aux);
100         }
101         liberarNO(prim);
102         free(li);
103     }
104 }
105
106 void liberarNO(NO* no){
107     free (no);
108 }
```


Inserindo no início

Cria lista: $[I] \rightarrow$ \Rightarrow Inserir(5): $[I] \rightarrow [5] \rightarrow$ \Rightarrow I(6): $[I] \rightarrow [6] \rightarrow$
 $[5]$

Inserir(7): $[I] \rightarrow [7] \rightarrow [6] \rightarrow [5] \rightarrow$

Inserindo no final

Cria lista: $[I] \rightarrow$ \Rightarrow Inserir(5): $[I] \rightarrow [5] \rightarrow$ \Rightarrow I(6): $[I] \rightarrow [5] \rightarrow$
 $[6]$

Inserir(7): $[I] \rightarrow [5] \rightarrow [6] \rightarrow [7] \rightarrow$

Destruindo lista

LiberaNo(5): $[I] \rightarrow [5] \rightarrow [6] \rightarrow [7] \rightarrow$ \Rightarrow LiberaNo(6): \rightarrow

$[I] \rightarrow [5] \rightarrow [7] \rightarrow$ \Rightarrow LiberaNo(7): $[I] \rightarrow [5] \rightarrow$ \Rightarrow Destroi L

~~[I]~~, lista destruída.