

UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 4

Nome: Geraldo Arthur Detomi

OBS: Os exercícios que eram para serem feitos no caderno estão no final do arquivo.

1) Lista Sequencial Estática

roteiro_4/Lista.h

```
1  /*----- File: Lista.h -----+
2  |Lista Sequencial Estatica      |
3  | | | | Implementado por Guilherme C. Pena em 12/09/2023 |
4  +-----+ */
5  #ifndef LISTA_H
6  #define LISTA_H
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define MAX 100
12
13 typedef struct {
14     int qtd;
15     int dados[MAX];
16 } Lista;
17
18 Lista *criaLista();
19
20 void destroiLista(Lista *li);
21
22 int tamanhoLista(Lista *li);
23
24 int listaCheia(Lista *li);
25
26 int listaVazia(Lista *li);
27
28 int insereFim(Lista *li, int elem);
29
30 int insereIni(Lista *li, int elem);
31 /*
32     Exercício 1-3 - Insere um elemento na lista de forma ordenada
33 */
34 int insereOrdenado(Lista *li, int elem);
35
36 /*
```

```
37  Exercício 1.4 - Remove a primeira ocorrência do elemento caso exista
38  */
39  int removeElemento(Lista *li, int elem);
40
41  int imprimeLista(Lista *li);
42
43  int removeFim(Lista *li);
44
45  int removeIni(Lista *li);
46
47  // Exercício 1-1 busca determinado elemento e se encontra retorna seu índice
48  // caso contrário retorna -1
49  int procura(Lista *li, int x);
50
51  #endif
52
```

roteiro_4/Lista.c

```
1  #include "Lista.h"
2
3  Lista *criaLista() {
4      Lista *li;
5      li = (Lista *)malloc(sizeof(Lista));
6      if (li != NULL)
7          li->qtd = 0;
8      return li;
9  }
10
11 void destroiLista(Lista *li) {
12     if (li != NULL)
13         free(li);
14 }
15
16 int tamanhoLista(Lista *li) {
17     if (li == NULL)
18         return -1;
19     return li->qtd;
20 }
21
22 int listaCheia(Lista *li) {
23     if (li == NULL)
24         return -1;
25     return (li->qtd == MAX);
26 }
27
28 int listaVazia(Lista *li) {
29     if (li == NULL)
30         return -1;
31     return (li->qtd == 0);
32 }
33
34 int insereFim(Lista *li, int elem) {
35     if (li == NULL)
```

```
36     return 0;
37     if (!listaCheia(li)) {
38         li->dados[li->qtd] = elem;
39         li->qtd++;
40         return 1;
41     } else {
42         return 0;
43     }
44 }
45
46 int insereIni(Lista *li, int elem) {
47     if (li == NULL)
48         return 0;
49     if (!listaCheia(li)) {
50         int i;
51         for (i = li->qtd; i > 0; i--) {
52             li->dados[i] = li->dados[i - 1];
53         }
54         li->dados[0] = elem;
55         li->qtd++;
56         return 1;
57     } else {
58         return 0;
59     }
60 }
61
62 int imprimeLista(Lista *li) {
63     if (li == NULL)
64         return 0;
65     int i;
66     printf("Elementos:\n");
67     for (i = 0; i < li->qtd; i++) {
68         printf("%d ", li->dados[i]);
69     }
70     printf("\n");
71     return 1;
72 }
73
74 int removeFim(Lista *li) {
75     if (li == NULL)
76         return 0;
77     if (!listaVazia(li)) {
78         li->qtd--;
79         return 1;
80     } else
81         return 0;
82 }
83
84 int removeIni(Lista *li) {
85     if (li == NULL)
86         return 0;
87     if (!listaVazia(li)) {
88         int i;
89         for (i = 0; i < li->qtd - 1; i++)
```

```
90     li->dados[i] = li->dados[i + 1];
91     li->qtd--;
92     return 1;
93 } else {
94     return 0;
95 }
96 }
97
98 int procura(Lista *li, int x) {
99     if (li == NULL) {
100         return -1;
101     }
102
103     for (int i = 0; i < li->qtd; i++) {
104         if (li->dados[i] == x) {
105             return i;
106         }
107     }
108
109     return -1;
110 }
111
112 int insereOrdenado(Lista *li, int elem) {
113     if (li == NULL || listaCheia(li)) {
114         return 0;
115     }
116
117     int index = -1;
118
119     for (int i = 0; i < li->qtd; i++) {
120         if (li->dados[i] > elem) {
121             index = i;
122             break;
123         }
124     }
125
126     if (index == -1) {
127         return insereFim(li, elem);
128     }
129
130     for (int i = li->qtd; i > index; i--) {
131         li->dados[i] = li->dados[i - 1];
132     }
133
134     li->dados[index] = elem;
135     li->qtd++;
136
137     return 1;
138 }
139
140 int removeElemento(Lista *li, int elem) {
141     if (li == NULL || listaVazia(li)) {
142         return 0;
143     }
```

```
144     int index = -1;
145
146     for (int i = 0; i < li->qtd; i++) {
147         if (li->dados[i] == elem) {
148             index = i;
149             break;
150         }
151     }
152
153     if (index == -1) {
154         return 0;
155     }
156
157     for (int i = index; i < li->qtd - 1; i++) {
158         li->dados[i] = li->dados[i + 1];
159     }
160
161     li->qtd--;
162
163     return 1;
164 }
```

roteiro_4/lista_estatica_test.c

```
1  #include "Lista.h"
2
3  int main() {
4      Lista *L;
5      L = criaLista();
6
7      insereOrdenado(L, 30);
8      insereOrdenado(L, 20);
9      insereOrdenado(L, 10);
10     insereOrdenado(L, 11);
11     insereOrdenado(L, 5);
12     insereOrdenado(L, 2);
13     imprimeLista(L);
14     if (removeElemento(L, 11)) {
15         printf("Elemento %d removido com sucesso\n", 11);
16     }
17
18     int index = procura(L, 30);
19
20     if (index != -1) {
21         printf("Elemento 30 encontrado na posição %d\n", index);
22     }
23
24     imprimeLista(L);
25
26     destroiLista(L);
27     return 0;
28 }
```

Saída do terminal:

```

arthurdetomi at arthurdetomi-System-Product-Name
• ./lista_estatica_test.out
Elementos:
2 5 10 11 20 30
Elemento 11 removido com sucesso
Elemento 30 encontrado na posição 4
Elementos:
2 5 10 20 30

```

2) Lista Simplesmente Encadeada

roteiro_4/LSE.h

```

1  /*----- File: LSE.h -----+
2  |Lista Simplesmente Encadeada      |
3  | | | | Implementado por Guilherme C. Pena em 14/09/2023      |
4  +-----+ */
5
6  #ifndef LISTASE_H
7  #define LISTASE_H
8
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 typedef struct NO {
13     int info;
14     struct NO *prox;
15 } NO;
16
17 typedef struct NO *Lista;
18
19 Lista *criaLista();
20
21 int listaVazia(Lista *li);
22
23 NO *alocarNO();
24
25 void liberarNO(NO *q);
26
27 int insereIni(Lista *li, int elem);
28
29 int insereFim(Lista *li, int elem);
30
31 int removeIni(Lista *li);
32
33 int removeFim(Lista *li);
34
35 void imprimeLista(Lista *li);
36
37 void recComplementar(NO *n);
38
39 void imprimeRevRec(Lista *li);
40
41 void imprimeRev(Lista *li);
42

```

```
43 void destroiLista(Lista *li);
44
45 /* Exercício 2.2 */
46 int tamanho(Lista *li);
47 int procura(Lista *li, int x);
48 int insereOrdenado(Lista *li, int elem);
49 int removePrimeiraOcorrencia(Lista *li, int elem);
50
51 #endif
52
```

roteiro_4/LSE.c

```
1  #include "LSE.h"
2
3  Lista *criaLista() {
4      Lista *li;
5      li = (Lista *)malloc(sizeof(Lista));
6      if (li != NULL) {
7          *li = NULL;
8      }
9      return li;
10 }
11
12 int listaVazia(Lista *li) {
13     if (li == NULL)
14         return 1;
15     if (*li == NULL)
16         return 1; // True - Vazia!
17     return 0;    // False - tem elemento!
18 }
19
20 NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
21
22 void liberarNO(NO *q) { free(q); }
23
24 int insereIni(Lista *li, int elem) {
25     if (li == NULL)
26         return 0;
27     NO *novo = alocarNO();
28     if (novo == NULL)
29         return 0;
30     novo->info = elem;
31     novo->prox = *li;
32     *li = novo;
33     return 1;
34 }
35
36 int insereFim(Lista *li, int elem) {
37     if (li == NULL)
38         return 0;
39     NO *novo = alocarNO();
40     if (novo == NULL)
41         return 0;
```

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



```
96     }
97     printf("\n");
98 }
99
100 void recComplementar(NO *n) {
101     if (n == NULL)
102         return;
103     recComplementar(n->prox);
104     printf("%d ", n->info);
105 }
106
107 void imprimeRevRec(Lista *li) {
108     if (li == NULL)
109         return;
110     if (listaVazia(li)) {
111         printf("Lista Vazia!\n");
112         return;
113     }
114     printf("Elementos:\n");
115     recComplementar(*li);
116     printf("\n");
117
118     // imprimeRevRec(&(*li)->prox);
119     // printf("%d ", (*li)->info);
120 }
121
122 void imprimeRev(Lista *li) {
123     if (li == NULL)
124         return;
125     if (listaVazia(li)) {
126         printf("Lista Vazia!\n");
127         return;
128     }
129     printf("Elementos REV:\n");
130     NO *ant, *aux;
131     NO *fim = NULL;
132     do {
133         aux = *li;
134         while (aux != fim) {
135             ant = aux;
136             aux = aux->prox;
137         }
138         printf("%d ", ant->info);
139         fim = ant;
140     } while (fim != *li);
141     printf("\n");
142 }
143
144 int tamanho(Lista *li) {
145     if (li == NULL)
146         return -1;
147     if (listaVazia(li)) {
148         return 0;
149     }
```

```
150
151     NO *aux = *li;
152
153     int tamanho = 0;
154
155     while (aux != NULL) {
156         aux = aux->prox;
157         tamanho++;
158     }
159
160     return tamanho;
161 }
162
163 int procura(Lista *li, int x) {
164     if (li == NULL || listaVazia(li)) {
165         return -1;
166     }
167
168     NO *aux = *li;
169
170     while (aux != NULL) {
171         if (aux->info == x) {
172             return 1;
173         }
174
175         aux = aux->prox;
176     }
177
178     return 0;
179 }
180
181 int removePrimeiraOcorrencia(Lista *li, int elem) {
182     if (li == NULL || listaVazia(li)) {
183         return 0;
184     }
185
186     NO *anterior = NULL, *atual = *li;
187
188     while (atual != NULL && atual->info != elem) {
189         anterior = atual;
190         atual = atual->prox;
191     }
192
193     if (atual == NULL) {
194         return 0;
195     }
196
197     if (anterior == NULL) {
198         *li = atual->prox;
199     } else {
200         anterior->prox = atual->prox;
201     }
202
203     liberarNO(atual);
```

```
204
205     return 1;
206 }
207
208 int insereOrdenado(Lista *li, int elem) {
209     if (li == NULL) {
210         return -1;
211     }
212
213     NO *anterior = NULL, *atual = *li;
214
215     while (atual != NULL && atual->info < elem) {
216         anterior = atual;
217         atual = atual->prox;
218     }
219
220     if (anterior == NULL) {
221         return insereIni(li, elem);
222     }
223
224     NO *novo_no = alocarNO();
225     novo_no->info = elem;
226
227     anterior->prox = novo_no;
228     novo_no->prox = atual;
229
230     return 1;
231 }
232
233 void destroiLista(Lista *li) {
234     if (li != NULL) {
235         NO *aux;
236         while ((*li) != NULL) {
237             aux = *li;
238             *li = (*li)->prox;
239             liberarNO(aux);
240         }
241         free(li);
242     }
243 }
244
```

roteiro_4/LSE_test.c

```
1  #include "LSE.h"
2  #include <stdio.h>
3
4  int main() {
5      Lista *L;
6      L = criaLista();
7
8      insereOrdenado(L, 20);
9      insereOrdenado(L, 30);
10     insereOrdenado(L, 40);

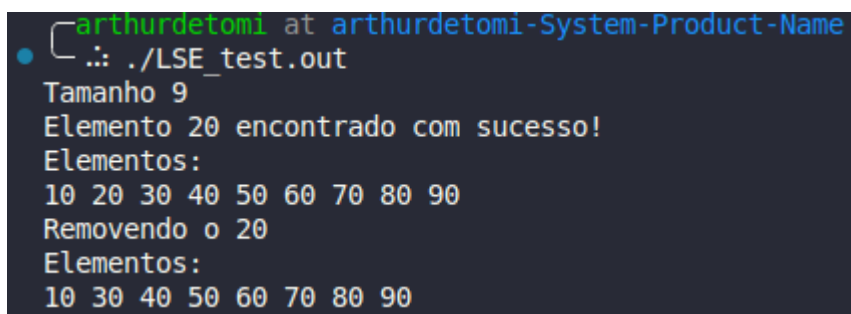
```

```

11     insereOrdenado(L, 50);
12     insereOrdenado(L, 60);
13     insereOrdenado(L, 70);
14     insereOrdenado(L, 80);
15     insereOrdenado(L, 90);
16     insereOrdenado(L, 10);
17
18     printf("Tamanho %d\n", tamanho(L));
19
20     if (procura(L, 20)) {
21         printf("Elemento 20 encontrado com sucesso!\n");
22     }
23     imprimeLista(L);
24
25     printf("Removendo o 20\n");
26     removePrimeiraOcorrencia(L, 20);
27
28     imprimeLista(L);
29
30     destroiLista(L);
31     return 0;
32 }
33

```

Saída do terminal:



```

arthurdetomi at arthurdetomi-System-Product-Name
● .: ./LSE_test.out
Tamanho 9
Elemento 20 encontrado com sucesso!
Elementos:
10 20 30 40 50 60 70 80 90
Removendo o 20
Elementos:
10 30 40 50 60 70 80 90

```

3) Lista Duplamente Encadeada

roteiro_4/LDE.h

```

1  /*----- File: LDE.h -----+
2  |Lista Duplamente Encadeada      |
3  | | | | Implementado por Guilherme C. Pena em 19/09/2023      |
4  +-----+ */
5
6  #ifndef LDE_H
7  #define LDE_H
8
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 typedef struct NO {
13     int info;
14     struct NO *prox;
15     struct NO *ant;

```

```
16 } NO;
17
18 typedef struct NO *Lista;
19
20 Lista *criaLista();
21
22 int listaVazia(Lista *li);
23
24 NO *alocarNO();
25
26 void liberarNO(NO *q);
27
28 int insereIni(Lista *li, int elem);
29
30 int insereFim(Lista *li, int elem);
31
32 int removeIni(Lista *li);
33
34 int removeFim(Lista *li);
35
36 void imprimeLista(Lista *li);
37
38 void destroiLista(Lista *li);
39
40 int removeCasoExista(Lista *li, int elem);
41
42 /*
43     Seção 3 Exercícios
44 */
45 int tamanho(Lista *li);
46 int procura(Lista *li, int x);
47 int insereOrdenado(Lista *li, int elem);
48
49 #endif
50
```

roteiro_4/LDE.c

```
1 #include "LDE.h"
2
3 Lista *criaLista() {
4     Lista *li;
5     li = (Lista *)malloc(sizeof(Lista));
6     if (li != NULL) {
7         *li = NULL;
8     }
9     return li;
10 }
11
12 int listaVazia(Lista *li) {
13     if (li == NULL)
14         return 1;
15     if (*li == NULL)
16         return 1; // True - Vazia!
```

```
17     return 0;    // False - tem elemento!
18 }
19
20 NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
21
22 void liberarNO(NO *q) { free(q); }
23
24 int insereIni(Lista *li, int elem) {
25     if (li == NULL)
26         return 0;
27     NO *novo = alocarNO();
28     if (novo == NULL)
29         return 0;
30     novo->info = elem;
31     novo->prox = *li;
32     novo->ant = NULL;
33     if (!listaVazia(li))
34         (*li)->ant = novo;
35     *li = novo;
36     return 1;
37 }
38
39 int insereFim(Lista *li, int elem) {
40     if (li == NULL)
41         return 0;
42     NO *novo = alocarNO();
43     if (novo == NULL)
44         return 0;
45     novo->info = elem;
46     novo->prox = NULL;
47     if (listaVazia(li)) {
48         novo->ant = NULL;
49         *li = novo;
50     } else {
51         NO *aux = *li;
52         while (aux->prox != NULL)
53             aux = aux->prox;
54         aux->prox = novo;
55         novo->ant = aux;
56     }
57     return 1;
58 }
59
60 int removeIni(Lista *li) {
61     if (li == NULL)
62         return 0;
63     if (listaVazia(li))
64         return 0;
65     NO *aux = *li;
66     *li = aux->prox;
67     if (aux->prox != NULL)
68         aux->prox->ant = NULL;
69     liberarNO(aux);
70     return 1;
```

```
71 }
72
73 int removeFim(Lista *li) {
74     if (li == NULL)
75         return 0;
76     if (listaVazia(li))
77         return 0;
78     NO *aux = *li;
79     while (aux->prox != NULL)
80         aux = aux->prox;
81     if (aux->ant == NULL)
82         *li = aux->prox;
83     else
84         aux->ant->prox = NULL;
85     liberarNO(aux);
86     return 1;
87 }
88
89 void imprimeLista(Lista *li) {
90     if (li == NULL)
91         return;
92     if (listaVazia(li)) {
93         printf("Lista Vazia!\n");
94         return;
95     }
96     printf("Elementos:\n");
97     NO *aux = *li;
98     while (aux != NULL) {
99         printf("%d ", aux->info);
100         aux = aux->prox;
101     }
102     printf("\n");
103 }
104
105 void destroiLista(Lista *li) {
106     if (li != NULL) {
107         NO *aux;
108         while ((*li) != NULL) {
109             aux = *li;
110             *li = (*li)->prox;
111             // printf("Destruindo.. %d\n", aux->info);
112             liberarNO(aux);
113         }
114         free(li);
115     }
116 }
117
118 int procura(Lista *li, int elem) {
119     if (li == NULL || listaVazia(li)) {
120         return 0;
121     }
122
123     NO *aux = *li;
124
```

```
125     while (aux != NULL) {
126         if (aux->info == elem) {
127             return 1;
128         }
129
130         aux = aux->prox;
131     }
132
133     return 0;
134 }
135
136 int tamanho(Lista *li) {
137     if (li == NULL || listaVazia(li)) {
138         return 0;
139     }
140     int tamanho = 0;
141
142     NO *aux = *li;
143
144     while (aux != NULL) {
145         tamanho++;
146
147         aux = aux->prox;
148     }
149
150     return tamanho;
151 }
152
153 int insereOrdenado(Lista *li, int elem) {
154     if (li == NULL) {
155         return 0;
156     }
157
158     NO *atual = *li;
159
160     while (atual != NULL && atual->info < elem) {
161         atual = atual->prox;
162     }
163
164     if (atual == NULL) {
165         return insereFim(li, elem);
166     }
167
168     NO *anterior = atual->ant;
169
170     if (anterior == NULL) {
171         return insereIni(li, elem);
172     }
173
174     NO *novo_no = alocarNO();
175
176     novo_no->info = elem;
177     novo_no->prox = atual;
178     novo_no->ant = anterior;
```



```
179
180     atual->ant = novo_no;
181
182     anterior->prox = novo_no;
183
184     return 1;
185 }
186
187 int removeCasoExista(Lista *li, int elem) {
188     if (li == NULL || listaVazia(li)) {
189         return 0;
190     }
191
192     NO *atual = *li;
193
194     while (atual != NULL && atual->info != elem) {
195         atual = atual->prox;
196     }
197
198     if (atual == NULL) {
199         return 0;
200     }
201
202     NO *anterior = atual->ant, *proximo = atual->prox;
203
204     if (anterior == NULL) {
205         return removeIni(li);
206     }
207
208     if (proximo == NULL) {
209         return removeFim(li);
210     }
211
212     proximo->ant = anterior;
213     anterior->prox = proximo;
214
215     liberarNO(atual);
216
217     return 1;
218 }
219
```

roteiro_4/LDE_test.c

```
1  #include "LDE.h"
2  #include <stdio.h>
3
4  int main() {
5      Lista *L;
6      L = criaLista();
7
8      insereOrdenado(L, 20);
9      insereOrdenado(L, 30);
10     insereOrdenado(L, 60);

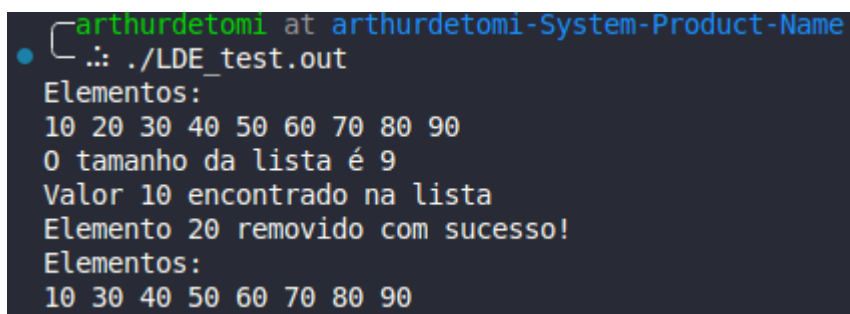
```

```

11     insereOrdenado(L, 40);
12     insereOrdenado(L, 50);
13     insereOrdenado(L, 80);
14     insereOrdenado(L, 70);
15     insereOrdenado(L, 90);
16     insereOrdenado(L, 10);
17
18     imprimeLista(L);
19
20     printf("O tamanho da lista é %d\n", tamanho(L));
21
22     if (procura(L, 10)) {
23         printf("Valor 10 encontrado na lista\n");
24     }
25
26     if (removeCasoExista(L, 20)) {
27         printf("Elemento 20 removido com sucesso!\n");
28     }
29
30     imprimeLista(L);
31
32     destroiLista(L);
33     return 0;
34 }

```

Saída do terminal:



```

arthurdetomi at arthurdetomi-System-Product-Name
❯ ./LDE_test.out
Elementos:
10 20 30 40 50 60 70 80 90
O tamanho da lista é 9
Valor 10 encontrado na lista
Elemento 20 removido com sucesso!
Elementos:
10 30 40 50 60 70 80 90

```

4) Lista Circular Simplesmente Encadeada

roteiro_4/LCSE.h

```

1  /*----- File: LCSE.h -----+
2  |Lista Circular Simplesmente Encadeada |
3  | | | | Implementado por Guilherme C. Pena em 19/09/2023 |
4  +-----+ */
5
6  #ifndef LCSE_H
7  #define LCSE_H
8
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 typedef struct NO {
13     int info;
14     struct NO *prox;

```

```
15 } NO;  
16  
17 typedef struct NO *Lista;  
18  
19 Lista *criaLista();  
20  
21 int listaVazia(Lista *li);  
22  
23 NO *alocarNO();  
24  
25 void liberarNO(NO *q);  
26  
27 int insereIni(Lista *li, int elem);  
28  
29 int insereFim(Lista *li, int elem);  
30  
31 int removeIni(Lista *li);  
32  
33 int removeFim(Lista *li);  
34  
35 void imprimeLista(Lista *li);  
36  
37 void destroiLista(Lista *li);  
38  
39 /*  
40     Exercício 4.2  
41 */  
42 int tamanho(Lista *li);  
43 int procura(Lista *li, int elem);  
44  
45 #endif
```

roteiro_4/LCSE.c

```
1 #include "LCSE.h"  
2  
3 Lista *criaLista() {  
4     Lista *li;  
5     li = (Lista *)malloc(sizeof(Lista));  
6     if (li != NULL) {  
7         *li = NULL;  
8     }  
9     return li;  
10 }  
11  
12 int listaVazia(Lista *li) {  
13     if (li == NULL)  
14         return 1;  
15     if (*li == NULL)  
16         return 1; // True - Vazia!  
17     return 0;    // False - tem elemento!  
18 }  
19  
20 NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
```

```
21
22 void liberarNO(NO *q) { free(q); }
23
24 int insereIni(Lista *li, int elem) {
25     if (li == NULL)
26         return 0;
27     NO *novo = alocarNO();
28     if (novo == NULL)
29         return 0;
30     novo->info = elem;
31     if (listaVazia(li)) {
32         novo->prox = novo;
33         *li = novo;
34     } else {
35         NO *aux = *li;
36         while (aux->prox != (*li))
37             aux = aux->prox;
38         aux->prox = novo;
39         novo->prox = *li;
40         *li = novo;
41     }
42     return 1;
43 }
44
45 int insereFim(Lista *li, int elem) {
46     if (li == NULL)
47         return 0;
48     NO *novo = alocarNO();
49     if (novo == NULL)
50         return 0;
51     novo->info = elem;
52     if (listaVazia(li)) {
53         novo->prox = novo;
54         *li = novo;
55     } else {
56         NO *aux = *li;
57         while (aux->prox != (*li))
58             aux = aux->prox;
59         aux->prox = novo;
60         novo->prox = *li;
61     }
62     return 1;
63 }
64
65 int removeIni(Lista *li) {
66     if (li == NULL)
67         return 0;
68     if (listaVazia(li))
69         return 0;
70     NO *prim = *li;
71     if (prim == prim->prox) {
72         // apenas 1 elemento
73         *li = NULL;
74     } else {
```

```
75     NO *aux = *li;
76     while (aux->prox != (*li))
77         aux = aux->prox;
78     aux->prox = (*li)->prox;
79     *li = (*li)->prox;
80 }
81 liberarNO(prim);
82 return 1;
83 }
84
85 int removeFim(Lista *li) {
86     if (li == NULL)
87         return 0;
88     if (listaVazia(li))
89         return 0;
90     NO *aux = *li;
91     if (aux == aux->prox) {
92         // apenas 1 elemento
93         *li = NULL;
94     } else {
95         NO *ant;
96         while (aux->prox != (*li)) {
97             ant = aux; // anterior
98             aux = aux->prox;
99         }
100        ant->prox = *li;
101    }
102    liberarNO(aux);
103    return 1;
104 }
105
106 void imprimeLista(Lista *li) {
107     if (li == NULL)
108         return;
109     if (listaVazia(li)) {
110         printf("Lista Vazia!\n");
111         return;
112     }
113     printf("Elementos:\n");
114     NO *aux = *li;
115     while (aux->prox != *li) {
116         printf("%d ", aux->info);
117         aux = aux->prox;
118     }
119     printf("%d ", aux->info);
120     printf("\n");
121 }
122
123 void destroiLista(Lista *li) {
124     if (li != NULL && (*li) != NULL) {
125         NO *prim, *aux;
126         prim = *li;
127         *li = (*li)->prox;
128         while ((*li) != prim) {
```

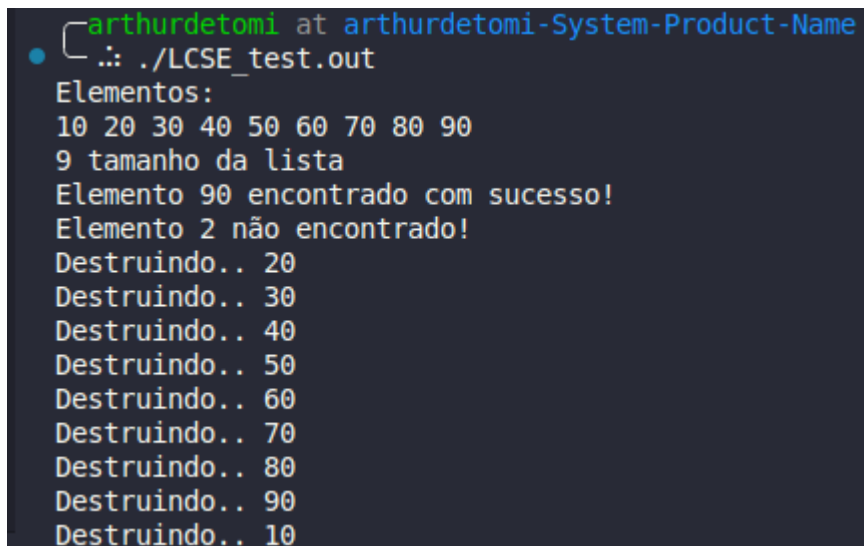
```
129     aux = *li;
130     *li = (*li)->prox;
131     printf("Destruindo.. %d\n", aux->info);
132     liberarNO(aux);
133 }
134 printf("Destruindo.. %d\n", prim->info);
135 liberarNO(prim);
136 free(li);
137 }
138 }
139
140 int tamanho(Lista *li) {
141     if (li == NULL || listaVazia(li)) {
142         return 0;
143     }
144
145     NO *aux = *li;
146
147     int tamanho = 0;
148
149     do {
150         tamanho++;
151
152         aux = aux->prox;
153     } while (aux != *li);
154
155     return tamanho;
156 }
157
158 int procura(Lista *li, int elem) {
159     if (li == NULL || listaVazia(li)) {
160         return 0;
161     }
162
163     NO *aux = *li;
164
165     do {
166         if (aux->info == elem) {
167             return 1;
168         }
169
170         aux = aux->prox;
171     } while (aux != *li);
172
173     return 0;
174 }
```

roteiro_4/LCSE_test.c

```
1 #include "LCSE.h"
2
3 int main() {
4     Lista *L;
5     L = criaLista();
```

```
6
7   insereFim(L, 10);
8   insereFim(L, 20);
9   insereFim(L, 30);
10  insereFim(L, 40);
11  insereFim(L, 50);
12  insereFim(L, 60);
13  insereFim(L, 70);
14  insereFim(L, 80);
15  insereFim(L, 90);
16
17  imprimeLista(L);
18
19  printf("%d tamanho da lista \n", tamanho(L));
20
21  if (procura(L, 90)) {
22      printf("Elemento 90 encontrado com sucesso!\n");
23  }
24
25  if (!procura(L, 2)) {
26      printf("Elemento 2 não encontrado!\n");
27  }
28
29  destroiLista(L);
30
31  return 0;
32 }
```

Saída do terminal:



```
arthurdetomi at arthurdetomi-System-Product-Name
❯ ./LCSE_test.out
Elementos:
10 20 30 40 50 60 70 80 90
9 tamanho da lista
Elemento 90 encontrado com sucesso!
Elemento 2 não encontrado!
Destruindo.. 20
Destruindo.. 30
Destruindo.. 40
Destruindo.. 50
Destruindo.. 60
Destruindo.. 70
Destruindo.. 80
Destruindo.. 90
Destruindo.. 10
```

Exercícios 2.1, 3.1 e 4.1

2.1) Lista Simplesmente Encadeada

I) Lista Vazia

li \rightarrow NULL

- Cria um ponteiro para ponteiro "li" que aponta para NULL inicialmente.

II) Inserção no fim

li \rightarrow [7] \rightarrow NULL

- Cria um novo nó com valor 7
- li aponta para o novo nó e o novo nó tem como ponteiro "next" apontando para NULL.

III) Inserção Início

li \rightarrow [3] \rightarrow [7] \rightarrow NULL

- Cria novo nó 3.
- li deve apontar agora para o novo nó "3".
- Atribui p/ o ponteiro "next" do novo nó o ponteiro com valor "7".

IV) Inserção Fim

li \rightarrow [3] \rightarrow [7] \rightarrow [4] \rightarrow NULL

- Cria novo nó "4"
- Atribui p/ o ponteiro "next" do nó "7" o novo nó "4".

V) Impressão

li \rightarrow [3] \rightarrow [7] \rightarrow [4] \rightarrow NULL

- Se lista for vazia não imprime nada
- Percorre todos os nós enquanto não encontrar um valor NULL e imprime a informação de cada um

VI) Liberação da lista

(HEAD) li \rightarrow [3] \rightarrow [7] \rightarrow [4] \rightarrow NULL

- Como a lista esteja vazia, libera a memória de li (HEAD).

Credeal

- 1 / 1 - Para criar elementos precisa um a um desalojando memória dos respectivos
- Por fim libera o ponteiro para ponteiro li

3.1) Lista Duplamente Encadeada

I) Lista Vazia

li \rightarrow NULL

- Cria um ponteiro para ponteiro li que aponta p/ NULL inicialmente.

II) Inserção Fim

li \rightarrow [4] \leftrightarrow [2] \rightarrow NULL

- Cria um novo nó "2".
- O ponteiro próximo do nó "4" passa a apontar para o novo nó.
- O ponteiro anterior do novo nó passa a apontar para o nó "4" e o seu próximo para NULL

(II) Inserção

li \rightarrow [4] \rightarrow NULL

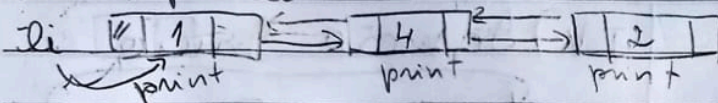
- Cria um novo nó "4".
- Aponta os ponteiros anteriormente próximo do novo nó para NULL
- li agora aponta p/ o novo nó.

III) Inserção Início

li \rightarrow [1] \leftrightarrow [4] \leftrightarrow [2]

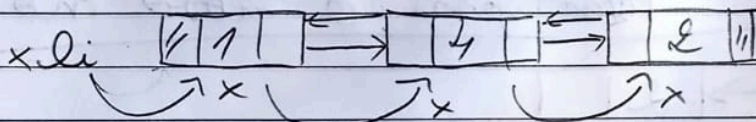
- Cria um novo nó "1".
- O ponteiro anterior do nó "4" agora deve apontar para o novo nó.
- O ponteiro próximo do novo nó deve apontar p/ o ponteiro "4" e seu anterior para NULL
- "li" agora deve apontar para o novo ponteiro.

IV) Impressão



- Percorre todos os elementos imprimindo seu valor enquanto diferente de NULL
- Utiliza o ponteiro "next" de cada nó p/ percorrer entre os ponteiros

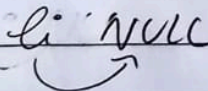
VI) Liberação de Lista



- Percorre a lista desalocando memória enquanto diferente de NULL
- Por fim libera o ponteiro p/ ponteiro li.

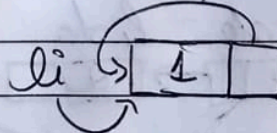
4.1) Lista Circular Simplesmente encadeada

I) Lista Vazia



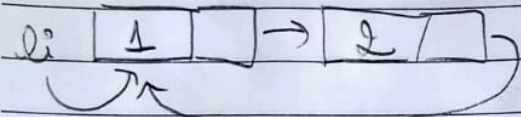
- Cria um ponteiro p/ ponteiro li apontando para NULL

II) Inserção



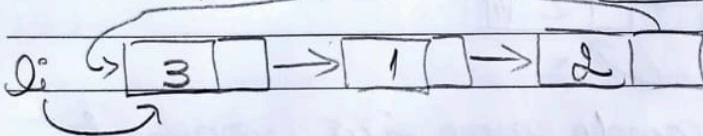
- Cria novo nó "1"
- li aponta para o novo nó
- o ponteiro "next" do novo nó aponta para ele mesmo

III) Inserção Fim



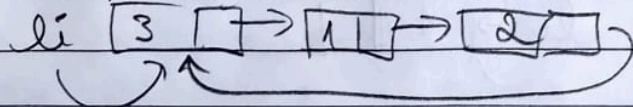
- Cria um novo nó "2".
- O nó "1" agora deve apontar para o "2" e o novo nó "2" aponta para o nó "1" formando um círculo.

IV) Impressão



- Se lista vazia não imprime
- Percorre a lista nó por nó imprimindo a informação de cada um até que chegue novamente por li

V) Liberação da lista



- Percorre cada nó desalocando memória até que retorne a li pela 2ª vez
- Por fim libera o ponteiro p/ ponteiro li.