4/18/25, 11:44 AM Selected files

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

4/18/25, 11:44 AM Selected files

```
Exercício 1.4 - Remove a primeira ocorrência do elemento caso exista
37
38
   */
39
   int removeElemento(Lista *li, int elem);
40
   int imprimeLista(Lista *li);
41
42
   int removeFim(Lista *li);
43
44
45
   int removeIni(Lista *li);
46
47
   // Exercicio 1-1 busca determinado elemento e se encontra retorna seu indice
48
   // caso contrário retorna -1
49
   int procura(Lista *li, int x);
50
51
   #endif
52
roteiro_4/Lista.c
    #include "Lista.h"
  2
  3
    Lista *criaLista() {
  4
      Lista *li;
  5
      li = (Lista *)malloc(sizeof(Lista));
       if (li != NULL)
  7
         li->qtd = 0;
  8
       return li;
  9
    }
 10
 11
    void destroiLista(Lista *li) {
       if (li != NULL)
 12
 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) {
       if (li == NULL)
```

```
36
        return 0;
37
      if (!listaCheia(li)) {
38
        li->dados[li->qtd] = elem;
39
        li->qtd++;
40
        return 1;
      } else {
41
42
        return 0;
43
      }
44
   }
45
46
   int insereIni(Lista *li, int elem) {
47
      if (li == NULL)
48
        return 0;
49
      if (!listaCheia(li)) {
        int i:
50
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
   int imprimeLista(Lista *li) {
      if (li == NULL)
63
64
        return 0;
65
      int i;
      printf("Elementos:\n");
66
      for (i = 0; i < li->qtd; i++) {
67
        printf("%d ", li->dados[i]);
68
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
        return 0;
81
82
   }
83
84
   int removeIni(Lista *li) {
85
      if (li == NULL)
        return 0;
86
87
      if (!listaVazia(li)) {
88
        int i;
89
        for (i = 0; i  qtd - 1; i++)
```

```
4/18/25, 11:44 AM
   90
             li->dados[i] = li->dados[i + 1];
   91
           li->qtd--;
           return 1;
   92
   93
         } else {
           return 0;
   94
   95
         }
   96
      }
   97
      int procura(Lista *li, int x) {
  98
  99
        if (li == NULL) {
  100
           return -1;
  101
  102
  103
        for (int i = 0; i < li->qtd; i++) {
           if (li->dados[i] == x) {
  104
  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
         for (int i = 0; i < li->qtd; i++) {
 119
  120
           if (li->dados[i] > elem) {
             index = i;
  121
  122
             break;
  123
           }
  124
         }
  125
  126
         if (index == -1) {
           return insereFim(li, elem);
  127
  128
         }
  129
  130
         for (int i = li->qtd; i > index; i--) {
           li->dados[i] = li->dados[i - 1];
  131
  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
         }
```

```
4/18/25, 11:44 AM
 144
        int index = -1;
 145
 146
        for (int i = 0; i qtd; i++) {
 147
           if (li->dados[i] == elem) {
             index = i;
 148
 149
             break;
 150
           }
 151
        }
 152
 153
        if (index == -1) {
 154
           return 0;
 155
 156
        for (int i = index; i < li->qtd - 1; i++) {
 157
           li->dados[i] = li->dados[i + 1];
 158
 159
        }
 160
        li->qtd--;
 161
 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);
     insereOrdenado(L, 20);
8
     insereOrdenado(L, 10);
9
     insereOrdenado(L, 11);
10
     insereOrdenado(L, 5);
11
     insereOrdenado(L, 2);
12
13
     imprimeLista(L);
     if (removeElemento(L, 11)) {
14
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:

4/18/25, 11:44 AM Selected files

```
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 /*----+
2
  |Lista Simplesmente Encadeada
  | | | | Implementado por Guilherme C. Pena em 14/09/2023
3
   +-----+ */
4
5
  #ifndef LISTASE H
6
7
  #define LISTASE H
8
  #include <stdio.h>
9
  #include <stdlib.h>
10
11
12
  typedef struct NO {
13
    int info;
     struct NO *prox;
14
   } NO;
15
16
17
   typedef struct NO *Lista;
18
19
  Lista *criaLista();
20
  int listaVazia(Lista *li);
21
22
   NO *alocarNO();
23
24
  void liberarNO(NO *q);
25
26
27
   int insereIni(Lista *li, int elem);
28
29
   int insereFim(Lista *li, int elem);
30
   int removeIni(Lista *li);
31
32
  int removeFim(Lista *li);
33
34
35
   void imprimeLista(Lista *li);
36
   void recComplementar(N0 *n);
37
38
39
  void imprimeRevRec(Lista *li);
40
   void imprimeRev(Lista *li);
41
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);
   int removePrimeiraOcorrencia(Lista *li, int elem);
49
50
51
   #endif
52
roteiro_4/LSE.c
  1 #include "LSE.h"
  2
    Lista *criaLista() {
  3
  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!
       return 0; // False - tem elemento!
 17
    }
 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
    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;
      if (listaVazia(li)) {
44
45
        *li = novo;
      } else {
46
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) {
      if (li == NULL)
56
        return 0;
57
58
      if (listaVazia(li))
59
        return 0;
60
      NO *aux = *li;
61
      *li = aux->prox;
      liberarNO(aux);
62
63
      return 1;
64
   }
65
   int removeFim(Lista *li) {
66
67
      if (li == NULL)
68
        return 0:
69
      if (listaVazia(li))
70
        return 0;
      NO *ant, *aux = *li;
71
72
     while (aux->prox != NULL) {
73
        ant = aux;
74
        aux = aux->prox;
75
76
      if (aux == *li)
77
        *li = aux->prox;
78
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)) {
        printf("Lista Vazia!\n");
88
89
        return;
90
      printf("Elementos:\n");
91
92
      N0 *aux = *li;
93
      while (aux != NULL) {
94
        printf("%d ", aux->info);
95
        aux = aux->prox;
```

```
4/18/25, 11:44 AM
   96
   97
         printf("\n");
   98
      }
  99
      void recComplementar(NO *n) {
 100
         if (n == NULL)
 101
 102
           return;
         recComplementar(n->prox);
 103
         printf("%d ", n->info);
 104
 105
      }
 106
 107
      void imprimeRevRec(Lista *li) {
        if (li == NULL)
 108
 109
           return;
 110
        if (listaVazia(li)) {
 111
           printf("Lista Vazia!\n");
 112
           return;
 113
        }
 114
         printf("Elementos:\n");
 115
        recComplementar(*li);
         printf("\n");
 116
 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)) {
           printf("Lista Vazia!\n");
 126
 127
           return;
 128
         }
 129
        printf("Elementos REV:\n");
 130
        NO *ant, *aux;
 131
        NO *fim = NULL;
        do {
 132
 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);
         printf("\n");
 141
 142
      }
 143
      int tamanho(Lista *li) {
 144
 145
         if (li == NULL)
 146
           return -1;
 147
         if (listaVazia(li)) {
 148
           return 0;
 149
```

```
4/18/25, 11:44 AM
 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
      int procura(Lista *li, int x) {
 163
         if (li == NULL || listaVazia(li)) {
 164
 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
       int removePrimeiraOcorrencia(Lista *li, int elem) {
 181
 182
         if (li == NULL || listaVazia(li)) {
 183
           return 0;
 184
         }
 185
        NO *anterior = NULL, *atual = *li;
 186
 187
        while (atual != NULL && atual->info != elem) {
 188
 189
           anterior = atual;
 190
           atual = atual->prox;
 191
         }
 192
         if (atual == NULL) {
 193
 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);
```

```
4/18/25, 11:44 AM
 204
 205
        return 1;
 206
      }
 207
      int insereOrdenado(Lista *li, int elem) {
 208
 209
        if (li == NULL) {
           return -1;
 210
 211
        }
 212
 213
        NO *anterior = NULL, *atual = *li;
 214
        while (atual != NULL && atual->info < elem) {</pre>
 215
 216
           anterior = atual;
 217
           atual = atual->prox;
 218
 219
 220
        if (anterior == NULL) {
 221
           return insereIni(li, elem);
 222
        }
 223
        NO *novo no = alocarNO();
 224
 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"
    #include <stdio.h>
   2
   3
     int main() {
  4
   5
       Lista *L;
   6
       L = criaLista();
  7
  8
       insereOrdenado(L, 20);
   9
       insereOrdenado(L, 30);
       insereOrdenado(L, 40);
 10
```

```
11
     insereOrdenado(L, 50);
12
     insereOrdenado(L, 60);
     insereOrdenado(L, 70);
13
14
     insereOrdenado(L, 80);
     insereOrdenado(L, 90);
15
     insereOrdenado(L, 10);
16
17
     printf("Tamanho %d\n", tamanho(L));
18
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 /*-----+
2
  |Lista Duplamente Encadeada
  | | | | Implementado por Guilherme C. Pena em 19/09/2023
3
4
5
  #ifndef LDE H
6
  #define LDE_H
7
8
  #include <stdio.h>
9
  #include <stdlib.h>
10
11
12
  typedef struct NO {
13
    int info;
14
    struct NO *prox;
    struct NO *ant;
15
```

```
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
   int insereFim(Lista *li, int elem);
30
31
32
   int removeIni(Lista *li);
33
34
   int removeFim(Lista *li);
35
   void imprimeLista(Lista *li);
36
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);
   int procura(Lista *li, int x);
46
47
   int insereOrdenado(Lista *li, int elem);
48
49
   #endif
50
roteiro_4/LDE.c
  1 #include "LDE.h"
  2
    Lista *criaLista() {
  3
  4
      Lista *li;
  5
      li = (Lista *)malloc(sizeof(Lista));
  6
      if (li != NULL) {
  7
         *li = NULL;
  8
      }
      return li;
  9
    }
 10
 11
 12
    int listaVazia(Lista *li) {
 13
      if (li == NULL)
 14
         return 1;
      if (*li == NULL)
 15
         return 1; // True - Vazia!
 16
```

```
17
      return 0; // False - tem elemento!
18
   }
19
20
   NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
21
   void liberarNO(NO *q) { free(q); }
22
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;
    novo->prox = *li;
31
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;
        *li = novo;
49
50
      } else {
51
       N0 *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
   int removeIni(Lista *li) {
60
      if (li == NULL)
61
62
        return 0;
      if (listaVazia(li))
63
64
        return 0;
      NO *aux = *li;
65
      *li = aux->prox;
66
67
      if (aux->prox != NULL)
68
        aux->prox->ant = NULL;
69
      liberarNO(aux);
70
      return 1;
```

```
4/18/25, 11:44 AM
   71 }
  72
   73
      int removeFim(Lista *li) {
   74
         if (li == NULL)
   75
           return 0;
   76
         if (listaVazia(li))
   77
           return 0;
   78
        N0 *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;
         liberarNO(aux);
   85
   86
         return 1;
   87
      }
   88
   89
      void imprimeLista(Lista *li) {
   90
         if (li == NULL)
   91
           return:
         if (listaVazia(li)) {
  92
  93
           printf("Lista Vazia!\n");
   94
           return;
   95
         }
        printf("Elementos:\n");
  96
  97
        N0 *aux = *li:
        while (aux != NULL) {
  98
  99
           printf("%d ", aux->info);
 100
           aux = aux -> prox;
 101
         printf("\n");
 102
 103
      }
 104
      void destroiLista(Lista *li) {
 105
 106
         if (li != NULL) {
           NO *aux;
 107
           while ((*li) != NULL) {
 108
 109
             aux = *li;
 110
             *li = (*li)->prox;
             // printf("Destruindo.. %d\n", aux->info);
 111
 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
         N0 *aux = *li;
 124
```

```
while (aux != NULL) {
125
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
      while (atual != NULL && atual->info < elem) {</pre>
160
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;
```

```
4/18/25, 11:44 AM
 179
 180
        atual->ant = novo no;
 181
 182
        anterior->prox = novo no;
 183
 184
       return 1;
 185
      }
 186
      int removeCasoExista(Lista *li, int elem) {
 187
 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
    int main() {
  4
   5
       Lista *L;
   6
       L = criaLista();
  7
       insereOrdenado(L, 20);
  8
   9
       insereOrdenado(L, 30);
 10
       insereOrdenado(L, 60);
```

```
11
     insereOrdenado(L, 40);
12
     insereOrdenado(L, 50);
     insereOrdenado(L, 80);
13
14
     insereOrdenado(L, 70);
15
     insereOrdenado(L, 90);
     insereOrdenado(L, 10);
16
17
18
     imprimeLista(L);
19
20
     printf("0 tamanho da lista é %d\n", tamanho(L));
21
     if (procura(L, 10)) {
22
        printf("Valor 10 encontrado na lista\n");
23
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
0 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 /*-----+
  |Lista Circular Simplesmente Encadeada
2
  | | | | Implementado por Guilherme C. Pena em 19/09/2023
3
  +----+ */
4
5
6 #ifndef LCSE_H
7
  #define LCSE_H
8
9
  #include <stdio.h>
  #include <stdlib.h>
10
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
   int insereFim(Lista *li, int elem);
29
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
    Exercício 4.2
40
41
   */
42 int tamanho(Lista *li);
   int procura(Lista *li, int elem);
43
44
45 #endif
roteiro_4/LCSE.c
  1 #include "LCSE.h"
  3
    Lista *criaLista() {
  4
      Lista *li;
  5
      li = (Lista *)malloc(sizeof(Lista));
      if (li != NULL) {
  6
  7
        *li = NULL;
  8
      }
  9
      return li;
 10
    }
 11
 12
    int listaVazia(Lista *li) {
 13
     if (li == NULL)
 14
        return 1;
      if (*li == NULL)
 15
         return 1; // True - Vazia!
 16
 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
   int insereIni(Lista *li, int elem) {
24
25
      if (li == NULL)
26
        return 0;
      NO *novo = alocarNO();
27
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
        N0 *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)
        return 0;
67
68
      if (listaVazia(li))
        return 0;
69
70
      NO *prim = *li;
71
      if (prim == prim->prox) {
72
        // apenas 1 elemento
73
        *li = NULL;
74
      } else {
```

```
4/18/25, 11:44 AM
   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);
         return 1;
   82
  83
      }
   84
   85
      int removeFim(Lista *li) {
  86
         if (li == NULL)
  87
           return 0;
         if (listaVazia(li))
  88
           return 0:
   89
  90
        N0 *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
           }
           ant->prox = *li;
 100
 101
 102
         liberarNO(aux);
 103
         return 1;
 104
      }
 105
      void imprimeLista(Lista *li) {
 106
         if (li == NULL)
 107
 108
           return;
 109
         if (listaVazia(li)) {
 110
           printf("Lista Vazia!\n");
           return;
 111
 112
         printf("Elementos:\n");
 113
 114
        N0 *aux = *li;
 115
        while (aux->prox != *li) {
           printf("%d ", aux->info);
 116
 117
           aux = aux -> prox;
 118
         }
         printf("%d ", aux->info);
 119
 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);
         liberarNO(prim);
135
136
         free(li);
137
      }
138
    }
139
140
    int tamanho(Lista *li) {
141
       if (li == NULL || listaVazia(li)) {
142
         return 0;
143
144
145
      N0 *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)) {
         return 0;
160
161
      }
162
163
      N0 *aux = *li;
164
165
      do {
         if (aux->info == elem) {
166
167
           return 1;
168
         }
169
170
       aux = aux -> prox;
       } while (aux != *li);
171
172
173
       return 0;
174 }
roteiro_4/LCSE_test.c
 1 #include "LCSE.h"
 2
 3 int main() {
 4
      Lista *L;
 5
      L = criaLista();
```

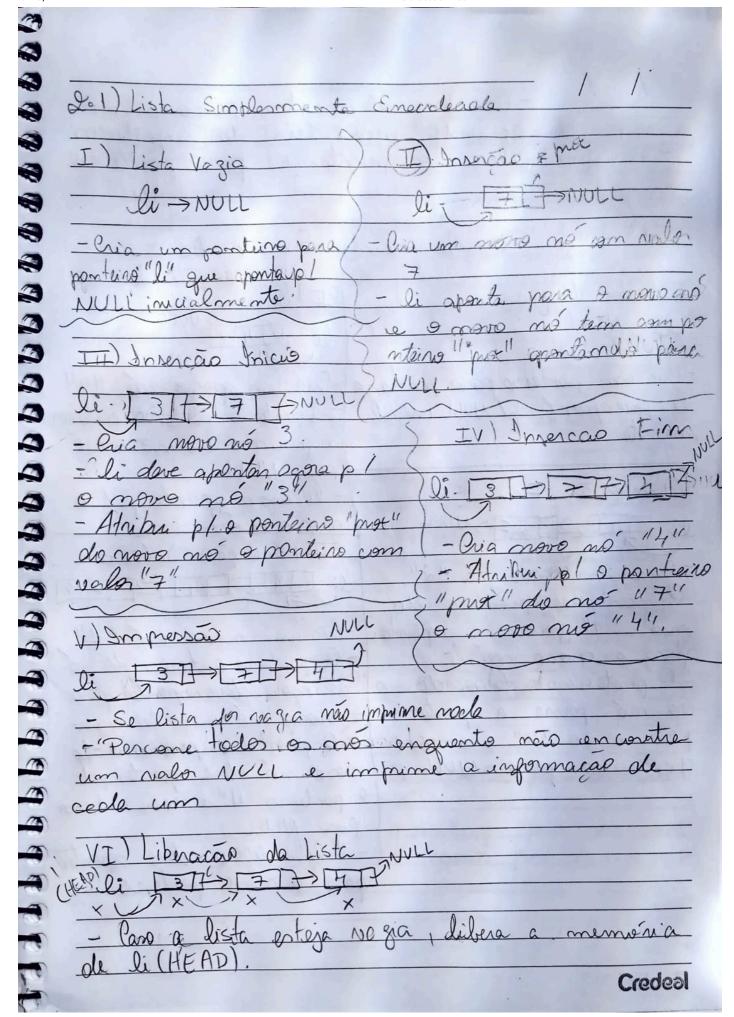
```
6
 7
     insereFim(L, 10);
 8
     insereFim(L, 20);
9
     insereFim(L, 30);
10
     insereFim(L, 40);
11
     insereFim(L, 50);
     insereFim(L, 60);
12
13
     insereFim(L, 70);
     insereFim(L, 80);
14
15
     insereFim(L, 90);
16
17
     imprimeLista(L);
18
19
     printf("%d tamanho da lista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

Exercícios 2.1, 3.1 e 4.1



/ / = 0 = 100i 0 to 100
a um blesalosendo memoria dos respectivos
- Par lim liker a montring para ranting
li li
STATE STATE OF THE
3.1) Lista Duplamente Encode ade
The state of the s
I) Lista Vogia II) Inveneas
li NULL
- Cuia um parteino para ponteino - Pria um momo mé 1/41/
- Cuia um parteino para pontano - Pria um mono mé 11411 li que aponta p/ NUL inicial - Aponto os pontano anterior
mente. La protimo do novo mó
paie NULL
II) Inserção Fim /-li ogora aponta p/o
1: MH TO DED NULL (movo no
- Pria um movo mo "2", li [1]
-O ponteinie práino do
mo "4" passa a apondar - lia um novo nó "1"
para o moro no, -O ponteiro on terior do nó
porteno anterior do mo "4" ogora deve operator pro
NO MO PONO A CHAMAS O MONO NO.
rare o mo "4" e o sen - O ponteiro motime do
aprotimo para NUL moro no dere aporto pl
Para MIII
- "bill agora deve aponto
pare a nova porteiro.
Credeal

- Por fins libera o ponteino p/ ponteino li 4.1) Lista lingular. Sim planomente encadecado I) Lista Vozia II) Imerção Li NULL Cuia um prenteino p/ ponteino li apantendo - lia mono mé "I" pona NULL - li apanta para o no mó - o ponteino "puet" do n	Di l'I print print Transme talos os elements impriminado seu doctos enquents diferente de NULL - Utiliza o ponteino "prot" de colo mó ponteino "prot" de colo mó ponteino ponteino VI) Liberação de Lista X. Qi 1/1 1/2 2 1/2 X. Qi 1/1 1/4 1/4 2 1/4 - Penene a lista desclorando memoria enquent diferente de NULL - Por fim libera o ponteino porteino le 4.1) Lista lingular Simplemente encodeodo I) Lista Vazia II) Smarção Li NULL lista nom pronteino por ponteino li apontendo - Ena mono mó "1" pona NULL - li aponta para o no		
Di 11 Print print print Pr	Di # 1 Parint - Rousne talos os elements impriminado reu doctos enquento diferente de NULL - Utiliza o pronteino "mot" de calo no p/ ponconer estre os ponteinos VI) Liberação do Lista - Recente a lista decelarando memaria enquent diferente de NULL - Per fim libera o ponteino p/ ponteiro li 4.1) Lista laquela. Sim plamento encoclocado I) Lista Vazia II) merços li NULL lia am pronteiro p/ ponteiro li apontendo "- lia novo mo "I" pona NULL - aponta para o no mo aponta para o no mo aponta para o la mes.		
Di 11 Print print print Pr	Di # 1 Parint - Rousne talos os elements impriminado reu doctos enquento diferente de NULL - Utiliza o pronteino "mot" de calo no p/ ponconer estre os ponteinos VI) Liberação do Lista - Recente a lista decelarando memaria enquent diferente de NULL - Per fim libera o ponteino p/ ponteiro li 4.1) Lista laquela. Sim plamento encoclocado I) Lista Vazia II) merços li NULL lia am pronteiro p/ ponteiro li apontendo "- lia novo mo "I" pona NULL - aponta para o no mo aponta para o no mo aponta para o la mes.	IV Ummessão	
impriminate ser doctor enquents differente de NULL - Utiliza o pronteiro "prot" de cola mó por remoner entre or ponteiros VI) Liberação do Lista X. Di U1 1 1 2 1 - Persone a lista decelarando anemaria enquent diferente de NULL - Por firm libera o ponteiro porteiro li 4.1) Lista Cincular Siam plaramente encadeado I) Lista Vozia II) Smerção Li NULL Cria um pronteiro por ponteiro li aponte para o no no - o ponteiro "prot" do no - o ponteiro "proteiro proteiro p	impriminate ser aboles enquents differente de NUIL Utiliza o parteiro "mot" de Calo mó pl renconer estre os ponteiros VI) Liberação do Lista X. li VII 4 7 2 1 Tx Tx Tx Tx - Persone a lista devalamado memmia enquent diferente de NULL - Por fim libera o ponteiro p/ ponteiro li 4. 1) Lista Circula. Simplemente encodeado I) Lista Vazia II) Imerção Li NULL luia um pronteiro p/ porteiro li aponte volo - Cia mono mó "11" pone NULL - a ponteiro "prot" do m má aponta para ele mes.	0: 11 5 14	2
impriminate seu doctor enquentre diferente de NWI - Utiliza o ponteino "prot" de cola mó p/ pronconer entre os ponteinos VI) Liberação de Lista X. li VII 4 - L 1 - Persone a lista develocando memoria enquent diferente de NULL - Por fins libera o ponteino p/ ponteino li 4.1) Lista Cinquela. Sins plasomente encoleodo I) Lista Vazia II) Smenção Li NULL li SI I Cuia um pronteiro p/ ponteiro li aponteado - Cua novo mó "I" pona NULL - Li aponta para o no no - o ponteiro "prot" do n	impriminate ser doctor enquento di ferente de NVII Liberação de Lista - Persone a lista desclarando mensaria enquento diferente de NVII - Por fina libera o pontano p/ ponteiro li 4.1) Lista linguala. Sim planmente encocleado I) Lista Vagia II) Smarção Li VIII Lia non pronteiro p/ ponteiro li oponteiro p/ ponteiro p/ p/ p/ do m	Ligarint point	bunt,
impriminate seu doctor enquente diferente de NVII - Utiliza o ponteino "prot" de colo mó p/ pronconer entre os ponteinos VI) Liberação do Lista Xli VII 4 - & - Persona lista devalorando memoria enquent diferente de NULL - Por fins libera o ponteino p/ ponteino li 1.1) Lista Cinquela. Simplemente encoleodo I) Lista Vazia II) Smenção li NULL lia um pronteiro p/ ponteiro li aponteiro p/ ponteiro l'III	impriminate ser doctor enquento di ferente de NVII Liberação de Lista - Persone a lista desclarando mensaria enquento diferente de NVII - Por fina libera o pontano p/ ponteiro li 4.1) Lista linguala. Sim planmente encocleado I) Lista Vagia II) Smarção Li VIII Lia non pronteiro p/ ponteiro li oponteiro p/ ponteiro p/ p/ p/ do m	- Persone todas os	elementes
differente de NVIL - Viliza o parteiro "prot" de coola mó p / manconer entre os ponteiros VI) Liberação do Lista X li VII 4 4 5 2 11 - Persone a lista desalacando memoria enquant diferente de NVILL - Por fins libera o ponteiros p / ponteiro li 4.1) Lista Cinarla. Sim planmento encoolado I) Lista Vozia II) Smarção Li NVILL lia um prenteiro p/ ponteiro li apontendo - Cia moro mó "I" pona NVILL - Li aponta para o no nó - o ponteiro "prot" do n	differente de NVII - Otiliza o pronteino "prot" de cola no p / pronconer estre os ponteinos VI) Liberação de Lista X. Qi VII JI JI E II X. Qi VII JI JI JI JI Y. A JI Lista de NULL Lista Lingular Lista Lingular Lista Vozia JI Januare no Ponteiro JI JI Lista um pronteira p Ponteiro JI Lista um pronteira p Ponteiro JI Ponteiro JI Lista um pronteira p Ponteiro JI Ponteiro Ji		
Coole no pronteino prot de Coole no pronteino prot de Ponteino VI) Liberação de Lista Xeli 1/1 4 5 2 1 - Persona lista devalorando memoria enquent diferente de NULL - Por fim libera o ponteino p' ponteino li 4.1) Lista Cinquela. Simplemente encooleado I) Lista Vazia II) Smerção Li NULL li SI I ponteino li apontendo - Cua moro mó "I" ponteino li apontendo - Cua moro mó "I" pono NULL - Li aponta para o no no - o ponteino "puet" do n	Cole no pl prenconer estre or ponteiros VI) Liberação do Lista X. Que 1/1 - 1/2 1/2	diferente de NULL	Marie Marie Company
ronteinos VI) Liberação de Lista X. li VII - LI	ponteiros VI) Liberação do Lista X. li 11 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	- Utiliza o ponteino	"mot" de
NI) Liberação de LB = X. Di 1	ponteiros VI) Liberação do Lista X. Di 1	coole no p/ mono	oner entre or
- Penenca lista desclorando mempia enquant disperente de NULC - Por sim libera o ponteiro p' ponteiro li 4.1) lista liquila. Sim plesomente encooleado I) lista Vazia II) Smerção Li NULC lia um prenteiro p' ponteiro li aponteado - lua novo mó "1" pona NUL - li aponto para o no nó - a ponteiro "puet" do n	- Penene a lista develorando memoria enquante dispente de NULC - Por sim libera o ponteiro p' ponteiro li 4.1) Lista lingula. Sim planmente encadeado I) Lista Vazia II) Smarças Li NULC lia um pronteiro pl ponteiro li apontendo - lua moro mó "s" pona NUL - a ponta para o no - a ponta para elo mes.	ponteino	man of the same
- Penenca lista desclorando mempia enquant disperente de NULC - Por sim libera o ponteiro p' ponteiro li 4.1) lista liquila. Sim plesomente encooleado I) lista Vazia II) Smerção Li NULC lia um prenteiro p' ponteiro li aponteado - lua novo mó "1" pona NUL - li aponto para o no nó - a ponteiro "puet" do n	- Penene a lista develorando memoria enquante dispente de NULC - Por sim libera o ponteiro p' ponteiro li 4.1) Lista lingula. Sim planmente encadeado I) Lista Vazia II) Smarças Li NULC lia um pronteiro pl ponteiro li apontendo - lua moro mó "s" pona NUL - a ponta para o no - a ponta para elo mes.	- Williams	The state of the s
- Penenca lista desclorando mempia enquant disperente de NULC - Por sim libera o ponteiro p' ponteiro li 4.1) lista liquila. Sim plesomente encooleado I) lista Vazia II) Smerção Li NULC lia um prenteiro p' ponteiro li aponteado - lua novo mó "1" pona NUL - li aponto para o no nó - a ponteiro "puet" do n	- Penene a lista develorando memoria enquante dispente de NULC - Por sim libera o ponteiro p' ponteiro li 4.1) Lista lingula. Sim planmente encadeado I) Lista Vazia II) Smarças Li NULC lia um pronteiro pl ponteiro li apontendo - lua moro mó "s" pona NUL - a ponta para o no - a ponta para elo mes.	VI Liberação de Li	36
- Persone a lista deselamendo memoria emquante diferente de NULC - Por fim libera o ponteino por ponteino li 4.1) Lista lingula Simplemente encadeado I) Lista Vazia II) Smerças Li NULC li SII ponteino li apantembo - Pira moro mó "1" pona NUL - Di apante para o no no - o ponteiro "puet" do n	- Persone a lista deselvando meninia enquante disperente de NULL - Per sim libera o ponteino pel ponteino li 4.1) Lista "liquilmi Simplemente" encocleado I) Lista Vazia II) Imercas li NULL lia um pronteino pel ponteino li apontendo - lua mono mó "1". pona NULL - Di aponta para o no nó - o prenteino "prot" do m mó aponta para ele mes.	ar contract	
- Persone a lista deselamendo memoria emquante diferente de NULC - Por fim libera o ponteino por ponteino li 4.1) Lista lingula Simplemente encadeado I) Lista Vazia II) Smerças Li NULC li SII ponteino li apantembo - Pira moro mó "1" pona NUL - Di apante para o no no - o ponteiro "puet" do n	- Persone a lista deselvando meninia enquante disperente de NULL - Per sim libera o ponteino pel ponteino li 4.1) Lista "liquilmi Simplemente" encocleado I) Lista Vazia II) Imercas li NULL lia um pronteino pel ponteino li apontendo - lua mono mó "1". pona NULL - Di aponta para o no nó - o prenteino "prot" do m mó aponta para ele mes.	xli (1) 4	
- Per fins libera o ponteino p/ ponteino li 4.1) Lista lingular. Sim planomente encadendo I) Lista Vozia II) Imerção Li NUL Cuia um prenteino p/ ponteino li apantendo - lia mono mé "I" pona NUL - li apanta para o no mó - o ponteino "puet" do n	dispende de NUCC - Par Jim libera a ponteira p' ponteira li 4.1) Lista lingula. Simples amente en a alco de I) Lista Vazia II) Smrenças Li NUCC li JIII Cuia um pronteira p' ponteira li apanteado - ena mono mó "I" pona NUCC - Li aponta para o no mó aponta para ele mes.		XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
- Per fins libera o ponteino p/ ponteino li 4.1) Lista lingular. Sim planomente encadendo I) Lista Vozia II) Imerção Li NUL Cuia um prenteino p/ ponteino li apantendo - lia mono mé "I" pona NUL - li apanta para o no mó - o ponteino "puet" do n	dispende de NUCC - Par Jim libera a ponteira p' ponteira li 4.1) Lista lingula. Simples amente en a alco de I) Lista Vazia II) Smrenças Li NUCC li JIII Cuia um pronteira p' ponteira li apanteado - ena mono mó "I" pona NUCC - Li aponta para o no mó aponta para ele mes.	- Tenenca lista de	alocando memoria enquant
4.1) Lista Pingula Simplemente encodeodo I) Lista Vazia II) Imerção Li NULC lia um pronteiro p/ ponteiro li apantendo - Pia novo mó "1" para NUL - Li apanta para o no no - a panteiro "puet" do n	4.1) Lista Pingula. Simplemente encode de I) Lista Vozia II) Imperçae Li VUIC li SII Cuia um pronteiro pol ponteiro li apanteardo - Pira novo mó "I". pana NUL - Di aponte para o mo nó - a ponteiro "pust" do m mó aponta para ele mes.	differente de NUCC	Total Court of March
Li Vozia Li Voll Lia um monteiro pl ponteiro li apontendo - cua novo mó "1". pona NUL - li aponta para o no nó - o ponteiro "pue" do n	Lista Vozia II) Imerças Li VUII Lia um pronteiro pol ponteiro li aponteado - era moro mó "1". pona NUII - li aponta para o no nó - a ponteiro "puet" do m mó aponta para ele mes.	- Par Sim libera	o ponteino ponteino le.
Li Vozia Li Voll Lia um monteiro pl ponteiro li apontendo - cua novo mó "1". pona NUL - li aponta para o no nó - o ponteiro "pue" do n	Lista Vozia II) Imerças Li VUII Lia um pronteiro pol ponteiro li aponteado - era moro mó "1". pona NUII - li aponta para o no nó - a ponteiro "puet" do m mó aponta para ele mes.	1 1) 1 1 -0	the state of the s
Li NULL Cia um monteiro pl ponteiro li aponteado - cua novo mó "1" pona NULL - Li aponta para o no nó - o ponteiro "puet" do n	Cià um pronteiro pl ponteiro li aponteado - la moro mó "1". pona NUL - li aponta para o no mó - o ponteiro "puet" do m mó aponta para ele mes.		Simplesmente encoledde
Cuia um monteiro pl ponteiro li apontendo - ena moro mó "1" para NUU - li aponta para o no nó - o ponteiro "puet" do n	Cira um pronteira pl ponteiro li aponte nolo - Cia novo mó "1" pana NUL - Di aponta pana o no nó - o ponteiro "puet" do n mó aponta para ele mes.	1) Lista Vagia	II) Immercan
ponteino li apontendo - eia novo mó "1". para NUL - li aponta para o no nó - o ponteiro "puet" do n	ponteinali apantando - Eria moro mó "1". pana NUL - li apanta para o no no - a panteiro "puet" do n mó apanta para ele mes.	- Li NULC	0:-611
ponteino li apontendo - eia novo mó "1". para NUL - li aponta para o no nó - o ponteiro "puet" do n	ponteinali apantando - Eria moro mó "1". pana NUL - li apanta para o no no - a panteiro "puet" do n mó apanta para ele mes.	- 1 1 1	St. 72
para NULL - li aponta para o no no - o ponte jo "puet" do n	pona NUL - li aponta para o no no' - a ponta para olo mes. nó aponta para ele mes.		- 0 : 11/h
- o ponte jo "puet" do n	- 9 ponteis "put" do n nó aponta paro ele mes.	V	O' and one of
- o ponting "pust" do n	- o parte par ele mes.	pona MUCC	
mó aponta paro ele mes	mó aponta paro ele mes.	and or seemed to	
mo apronta piro ele mes			- o pontano mot do m
			mo agranta piro de meso

/ / _	TO SELECTION OF THE PARTY OF TH
	IV) In serces Inicio
III) Inserção Fim	IV) In sergio Inicio
	10: 12 1211 1217
	14 3 1 1 2 1 1 2 1
	2: 2: 2: 2:11
Cra com more no "D".	- Bia um mores nos 3
O no "1" ogora deve aponto	- 9 ultimo mo 121 para
are a "2" o mous mo	Shows boro of which
2" apronta para o mo	- O moro no aponto pera
formando um cinculo	9 ponteino "1"
	- li ogna passa a span
IV I Impressão	for para o novo con.
3 > 1	2
	- Taxan of Girls de conde
-Se lista vogia mais im	prime
Percone a lista mó /	ho no
informação a informação	de cools um
te que oparse monome	nte por li
9	ATT TO THE PARTY OF THE PARTY O
V) liberação do lista	
li 3 77 177 2	
Percone ado mó des	alocando memorio ate que
etome a li pela	Le vez
Por Jum litera &	montaine of montaine Oi
4	The following see
	AND ASSESSMENT OF THE PARTY OF