Atividade 4 – Estrutura de Dados 1 – 2020.2
Professora Ana Luiza Bessa de Paula Barros
Ciência da computação - UECE
Aluno: Emerson Lucena dos Santos, 1538695

Lista

1. Quais as diferenças entre uma lista sequencial estática e uma encadeada dinâmica? Em quais situações se deve usar cada uma delas?

R - **Uma lista sequencial estática** é um array (vetor) com N posições do tipo lista (com atributos de quantidade e o vetor em si). Além disso, a lista sequencial estática é definida como um ponteiro para um struct do tipo lista, em que o usuário tem acesso somente ao ponteiro. Sendo assim, deve-se usar a lista sequencial estática quando se tem uma lista pequena ou existe a intenção de acesso rápido e direto aos elementos (levando em consideração os índices do array); quando se deseja um tempo constante para acessar um elemento e facilidade para modificar informações, removê-las do início ou fim da lista e a operação de busca é a mais utilizada. Em contrapartida, tem-se uma limitação do tamanho do array com uma definição prévia, além de oferecer mais dificuldade entre inserção e remoção de elementos no meio da lista.

Uma lista encadeada dinâmica é uma lista definida utilizando alocação dinâmica e acesso encadeado aos elementos, de modo que todos ficam interligados de forma definida. Cada elemento da lista é alocado de modo dinâmico pois são ponteiros, e cada ponteiro possui um campo de dado (informação inserida na lista), um campo de prox (próximo elemento da lista) e usa-se um ponteiro para ponteiro para guardar o endereço de memória do primeiro elemento, que é um ponteiro, da lista. Utilizar esse tipo de lista nos permite melhorar a utilização dos recursos de memória, tendo em vista que será utilizado e ocupado de forma dinâmica, por isso, não é preciso definir o tamanho prévio da lista. Além disso, não precisa realizar a movimentação de elementos nas operações de inserção e remoção, diferentemente da lista sequencial estática. Todavia, o acesso aos elementos acontece de forma indireta, sendo necessário percorrer toda a lista para acessar um elemento desejado.

2. Qual a diferença entre uma lista e um vetor?

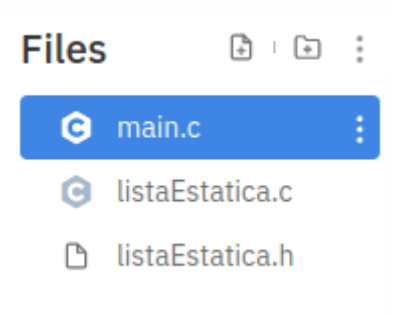
R - Uma lista é uma relação finita de itens, de modo que todos eles estejam em torno de um mesmo tema, como listas de itens em estoque em uma empresa, uma lista de compras domésticas, lista de convidados para uma festa, entre outros. De modo mais específico e técnico, uma lista é uma estrutura de dados linear utilizada para armazenar e organizar dados em um computador.

Quando se trata de um vetor, deve-se ter em mente que estamos nos referindo apenas a um arranjo de estrutura de dados responsável por armazenar elementos de modo que cada um possa ser identificado, pelo seu índice ou uma chave.

3. Implemente uma lista estática, uma encadeada circular e uma duplamente encadeada com as seguintes funções:

a. Criação;
b. Inserção de elemento;
c. Remoção de elemento;
d. Busca por um elemento;
e. Destruição da lista;
f. Tamanho da lista e
g. Verificar se a lista está cheia ou vazia.

LISTA SEQUENCIAL ESTÁTICA
ARQUIVOS CRIADOS:



file: listaEstatica.h (interface)

listaEstatica.h

```h
#ifndef LISTA_ESTATICA_H
#define LISTA_ESTATICA_H

#define MAX 100
struct pessoa {
  int idade;
  char nome[30];
  int cpf;
};

typedef struct pessoa Pessoa;
typedef struct lista Lista;

Lista* criar_lista();
void destruir_lista(Lista *);
int tamanho_lista(Lista *);
int lista_cheia(Lista* li);
int lista_vazia(Lista* li);
int inserir_inicio(Lista* li, struct pessoa dados);
int inserir_final(Lista* li, struct pessoa dados);
int inserir_meio(Lista* li, struct pessoa dados);
int remover_inicio(Lista* li);
int remover_meio(Lista* li, int);
int remover_final(Lista* li);
int consultar_lista_valor(Lista* li, int, struct pessoa *dados);
int consultar_lista_pos(Lista* li, int, struct pessoa *dados);

#endif
```

file: listaEstatica.c (implementação)

listaEstatica.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "listaEstatica.h"

struct lista {
  int qtd;
  struct pessoa dados[MAX];
};
//criando lista
Lista* criar_lista(){
  Lista *li;
  li = (Lista*) malloc(sizeof(struct lista));
  if(li != NULL){
    li->qtd = 0;
  }
  return li;
};
//destruindo lista
void destruir_lista(Lista* li){
  free(li);
};
//retornando o tamanho da lista
int tamanho_lista(Lista* li){
  if(li == NULL){
    return -1;
  } else {
    return li->qtd;
  }
}
//retornando se lista está cheia ou vazia
int lista_cheia(Lista* li){
```

listaEstatica.c

```
32    if(li == NULL){
33     return -1;
34    }
35    if(li->qtd == MAX) {
36      return 1;
37    } else {
38      return 0;
39    }
40  }
41  //retornando se lista está cheia ou vazia
42  int lista_vazia(Lista* li){
43    if(li == NULL){
44      return -1;
45    }
46    if(li->qtd == 0) {
47      return 1;
48    } else {
49      return 0;
50    }
51  }
52  //inserindo elemento no inicio
53  int inserir_inicio(Lista* li, struct pessoa dados){
54    if(li == NULL){
55      return 0;
56    } else if(lista_cheia(li)){
57      return 0;
58    }
59    if(lista_vazia(li)){
60      li->dados[0] = dados;
61      li->qtd++;
62      return 1;
```

listaEstatica.c

```
63    }
64    int aux = li->qtd;
65    while(aux != 0){
66      li->dados[aux] = li->dados[aux-1];
67      aux--;
68    }
69    li->dados[0] = dados;
70    li->qtd++;
71    return 1;
72  }
73  //inserindo elemento no fim
74  int inserir_final(Lista* li, struct pessoa dados){
75    if(li == NULL){
76      return 0;
77    } else if(lista_cheia(li)){
78      return 0;
79    }
80    li->dados[li->qtd] = dados;
81    li->qtd++;
82    return 1;
83  }
84  //inserindo no meio de forma ordenada
85  int inserir_meio(Lista* li, struct pessoa dados){
86    if(li == NULL){
87      return 0;
88    } else if(lista_cheia(li)){
89      return 0;
90    }
91    int i = 0;
92    while(i<li->qtd && li->dados[i].cpf < dados.cpf){
93      i++;
```

listaEstatica.c

```
94      }
95      for(int k = li->qtd-1; k>=i; k--){
96        li->dados[k+1] = li->dados[k];
97      }
98      li->dados[i] = dados;
99      li->qtd++;
100     return 1;
101   }
102   //removendo elemento do inicio
103   int remover_inicio(Lista* li){
104     if(li == NULL){
105       return 0;
106     } else if(lista_vazia(li)){
107       return 0;
108     }
109     int i = li->qtd;
110     for(int k = 0; k<i-1; k++){
111       li->dados[k] = li->dados[k+1];
112     }
113     li->qtd--;
114     return 1;
115   }
116   //removendo elemento do inicio
117   int remover_final(Lista* li){
118     if(li == NULL){
119       return 0;
120     } else if(lista_vazia(li)){
121       return 0;
122     }
123     li->qtd--;
124     return 1;
```

listaEstatica.c

```
125   }
126   //removendo elemento do meio da lista
127   int remover_meio(Lista* li, int cpf){
128     if(li == NULL){
129       return 0;
130     } else if(lista_vazia(li)){
131       return 0;
132     }
133     int i = 0;
134     while(i<li->qtd && li->dados[i].cpf != cpf){
135       i++;
136     }
137     if(i == li->qtd){
138       return 0; //404 not found
139     }
140
141     for(int k = i; k < li->qtd-1; k++){
142       li->dados[k] = li->dados[k+1];
143     }
144     li->qtd--;
145     return 1;
146   }
147   //Consultando na lista por valor
148   int consultar_lista_valor(Lista* li, int cpf, struct pessoa *dados){
149     if(li == NULL){
150       return 0;
151     } else if(lista_vazia(li)){
152       return 0;
153     }
154     for(int i = 0; i < li->qtd; i++){
155       if(li->dados[i].cpf == cpf){
```

listaEstatica.c

```
156         *dados = li->dados[i];
157         return 1;
158       }
159     }
160     return 0;
161   };
162   //Consultando na lista por posição
163   int consultar_lista_pos(Lista* li, int pos, struct pessoa *dados){
164     if(li == NULL || pos > li->qtd || pos <= 0 ){
165       return 0;
166     }
167     if(lista_vazia(li)){
168       return 0;
169     }
170     *dados = li->dados[pos-1];
171     return 1;
172   };
```

file: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEstatica.h"

int main(void) {
  Lista *li;
  Pessoa alguem;
  Pessoa human;
  char nomeVar[30] = "Emerson";

  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 12345678;

  li = criar_lista();
  int x = inserir_inicio(li, human);
  printf("Inserção no inicio da lista: %d\n", x);
  int u = inserir_inicio(li, human); //inserção de dois elementos
  printf("Inserção no inicio da lista: %d\n", u);
  int y = inserir_final(li, human);
  printf("Inserção no fim da lista: %d\n", y);
  int z = inserir_meio(li, human);
  printf("Inserção no meio da lista: %d\n", z);
  printf("\n\n");

  int t = remover_inicio(li);
  printf("Remoção no inicio da lista: %d\n", t);
  int w = remover_final(li);
  printf("Remoção no fim da lista: %d\n", w);
  int k = remover_meio(li, human.cpf);
  printf("Remoção no meio da lista: %d\n", k);
```

main.c

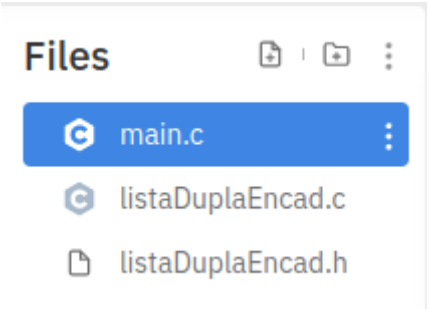
```
32    printf("\n\n");
33
34    int j = tamanho_lista(li);
35    printf("tamanho da lista: %d\n", j);
36    int n = lista_cheia(li);
37    printf("Lista cheia: %d\n", n);
38    int a = lista_vazia(li);
39    printf("Lista vazia: %d\n", a);
40    int h = consultar_lista_valor(li, human.cpf, &alguem);
41    printf("Consultar na lista valor: %d\n", h);
42    printf("%d\n", alguem.cpf);
43    int p = consultar_lista_pos(li, 1, &alguem);
44    printf("Consultar na lista posição: %d\n", p);
45    printf("%d\n", alguem.cpf);
46
47
48    destruir_lista(li);
49    return 0;
50  }
```

Resultados: 1 significa êxito.

Console Shell

```
> clang-7 -pthread -lm -o main listaEstatica.c main.c
> ./main
Inserção no inicio da lista: 1
Inserção no inicio da lista: 1
Inserção no fim da lista: 1
Inserção no meio da lista: 1


Remoção no inicio da lista: 1
Remoção no fim da lista: 1
Remoção no meio da lista: 1


tamanho da lista: 1
Lista cheia: 0
Lista vazia: 0
Consultar na lista valor: 1
12345678
Consultar na lista posição: 1
12345678
>
```

LISTA DUPLAMENTE ENCADEADA:
ARQUIVOS CRIADOS:



File: listaDuplaEncad.h (interface)

listaDuplaEncad.h

```h
#ifndef LISTA_DUPLA_ENCAD_H
#define LISTA_DUPLA_ENCAD_H

#define MAX 100
struct pessoa {
  int idade;
  char nome[30];
  int cpf;
};

typedef struct pessoa Pessoa;
typedef struct elemento* Lista;

Lista* criar_lista();
void destruir_lista(Lista* li);
int lista_cheia(Lista *li);
int lista_vazia(Lista *li);
int tamanho_lista(Lista *li);
int inserir_inicio(Lista* li, Pessoa dados);
int inserir_final(Lista* li, Pessoa dados);
int inserir_meio(Lista* li, Pessoa dados);
int remover_inicio(Lista* li);
int remover_fim(Lista* li);
int remover_meio(Lista* li, int);
int buscar_lista_valor(Lista* li, int, Pessoa *dados);
int buscar_lista_pos(Lista* li, int, Pessoa *dados);


#endif
```

File: listaDuplaEncad.c (implementação)

listaDuplaEncad.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "listaDuplaEncad.h"

struct elemento{
  struct elemento *ant;
  struct pessoa dados;
  struct elemento *prox;
};

typedef struct elemento Elem;

//Criando lista
Lista* criar_lista(){
  Lista* li = (Lista*) malloc(sizeof(Lista));
  if(li != NULL){
    *li = NULL;
  }
  return li;
}

//Excluindo lista
void destruir_lista(Lista* li){
  if(li != NULL){
    Elem* no;
    while((*li) != NULL){
      no = *li;
      *li = (*li)->prox;
      free(no);
    }
    free(li);
```

listaDuplaEncad.c

```c
  }
}

//Verificando se a lista está cheia
int lista_cheia(Lista *li){
  return 0;
}

//Verificando se a lista está vazia
int lista_vazia(Lista *li){
  if(li == NULL || *li == NULL ){
    return 1;
  }
  return 0;
}

//Exibindo o tamanho da lista
int tamanho_lista(Lista *li){
  if(li == NULL){
    return 0;
  }
  Elem* no = *li;
  int count = 0;
  while(no != NULL){
    count ++;
    no = no->prox;
  };
  return count;
};

//Inserindo elementos no inicio da lista
```

listaDuplaEncad.c

```c
int inserir_inicio(Lista* li, Pessoa dados){
    if(li == NULL){
      return 0; //se a lista nao existir
    }

    Elem* no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL){
      return 0;
    }

    no->ant = NULL;
    no->dados = dados;
    no->prox = (*li);
    if((*li) != NULL){
      (*li)->ant = no; //apontando o primeiro elemento
    }                  //depois do primeiro, depois
    *li = no;          //mudando o point do 1º elem.
    return 1;
}

//Inserindo elementos no final da lista
int inserir_final(Lista* li, Pessoa dados){
  if(li == NULL){
    return 0; //se a lista nao existir
  }
  Elem* no = (Elem*) malloc(sizeof(Elem));
  if(no == NULL){
    return 0;
  }
  no->dados = dados;
  no->prox = NULL;
```

listaDuplaEncad.c

```
94    if(lista_vazia(li)){
95      no->ant = NULL;
96      *li = no;
97      return 1;
98    }
99    Elem* aux = *li;
100   while(aux->prox != NULL){
101     aux = aux->prox;
102   }
103   aux->prox = no;
104   no->ant = aux;
105   return 1;
106 }
107
108 //Inserindo elementos no meio da lista de forma ordenada
109 int inserir_meio(Lista* li, Pessoa dados){
110   if(li == NULL){
111     return 0;
112   }
113   Elem* no = (Elem*) malloc(sizeof(Elem));
114   if(no == NULL){
115     return 0;
116   }
117   if(lista_vazia(li)){ //se for vazia
118     no->prox = NULL;
119     no->dados = dados;
120     no->ant = NULL;
121     *li = no;
122     return 1;
123   } else {
124     Elem *ant, *atual = *li;
```

listaDuplaEncad.c

```
125       no->dados = dados;
126       while(atual != NULL && atual->dados.cpf < dados.cpf){
127         ant = atual;
128         atual = atual->prox;
129       }
130       if(atual->ant == NULL){
131         no->ant = NULL;
132         no->prox = (*li);
133         (*li)->ant = no;
134         *li = no;
135       } else {
136         no->ant = ant;
137         no->prox = atual;
138         ant->prox = no;
139         if(atual != NULL){
140           atual->ant = no;
141         }
142       }
143       return 1;
144     }
145   }
146
147   //Removendo elementos do inicio da lista
148   int remover_inicio(Lista* li){
149     if(li == NULL){
150       return 0;
151     }
152     if(lista_vazia(li)){
153       return 0;
154     }
155       Elem* no = (*li);
```

listaDuplaEncad.c

```
156       *li = no->prox;
157     if((*li) != NULL){
158       no->prox->ant = NULL;
159     }
160
161     free(no);
162     return 1;
163   }
164
165   //Removendo elementos do fim da lista
166   int remover_fim(Lista* li){
167     if(li == NULL){
168       return 0;
169     }
170     if(lista_vazia(li)){
171       return 0;
172     }
173     Elem* no = *li;
174     while(no->prox != NULL){
175       no = no->prox;
176     }
177
178     if(no->ant == NULL){
179       *li = NULL; //só um elemento existe, logo, removemos ele
180     } else {
181       no->ant->prox = NULL; //dizendo que o elemento anterior ao ultimo,
          agora é o ultimo.
182     }
183
184     free(no);
185     return 1;
```

listaDuplaEncad.c

```c
}

//Removendo elementos do meio da lista de forma ordenada
int remover_meio(Lista* li, int cpf){
  if(li == NULL){
    return 0;
  }
  if(lista_vazia(li)){
    return 0;
  }
  Elem* no = *li;
  while(no != NULL && no->dados.cpf != cpf){
    no = no->prox;
  }
  if(no == NULL){
    return 0; //not found
  }
  if(no->ant == NULL){
    *li = no->prox;
    (*li)->ant = NULL;
  } else {
    no->ant->prox = no->prox;
  }
  if(no->prox == NULL){
    no->ant->prox = NULL;
  } else {
    no->prox->ant = no->ant;
  }

  free(no);
  return 1;
```

listaDuplaEncad.c

```c
}

//Buscando elemento na lista por valor
int buscar_lista_valor(Lista* li, int cpf, struct pessoa *dados){
  if(li == NULL){
    return 0;
  }
  if(lista_vazia(li)){
    return 0;
  }
  Elem* no = *li;
  while(no != NULL && no->dados.cpf != cpf){
    no = no->prox;
  }
  if(no == NULL){
    return 0;
  }
  *dados = no->dados;
  return 1;
}

//Buscando elemento na lista por posição
int buscar_lista_pos(Lista* li, int pos, struct pessoa *dados){
  if(li == NULL || pos <= 0){
    return 0;
  }
  if(lista_vazia(li)){
    return 0;
  }
  Elem* no = *li;
  int i = 1;
```

listaDuplaEncad.c

```c

  while(no != NULL && i < pos){
    no = no->prox;
    i++;
  }

  if(no == NULL){
    return 0;
  }

  *dados = no->dados;
  return 1;
}
```

File: main.c (chamada das funções)

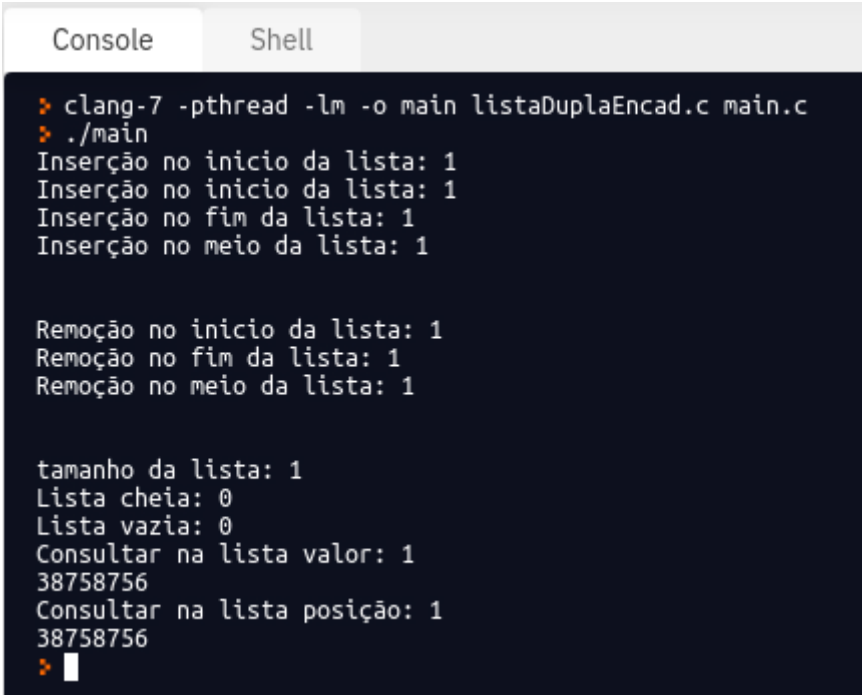
main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaDuplaEncad.h"

int main(void) {
  Lista *li;
  Pessoa alguem;
  Pessoa human;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 38758756;

  li = criar_lista();
  int x = inserir_inicio(li, human);
  printf("Inserção no inicio da lista: %d\n", x);
  int u = inserir_inicio(li, human); //inserção de dois elementos
  printf("Inserção no inicio da lista: %d\n", u);
  int y = inserir_final(li, human);
  printf("Inserção no fim da lista: %d\n", y);
  int z = inserir_meio(li, human);
  printf("Inserção no meio da lista: %d\n", z);
  printf("\n\n");

  int t = remover_inicio(li);
  printf("Remoção no inicio da lista: %d\n", t);
  int w = remover_fim(li);
  printf("Remoção no fim da lista: %d\n", w);
  int k = remover_meio(li, human.cpf);
  printf("Remoção no meio da lista: %d\n", k);
```

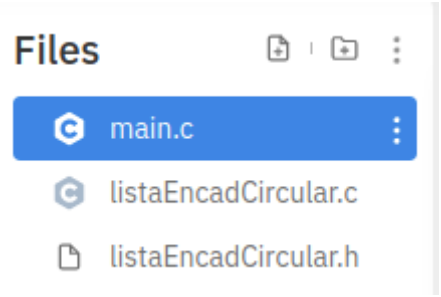
main.c

```c
  printf("\n\n");

  int j = tamanho_lista(li);
  printf("tamanho da lista: %d\n", j);
  int n = lista_cheia(li);
  printf("Lista cheia: %d\n", n);
  int a = lista_vazia(li);
  printf("Lista vazia: %d\n", a);
  int h = buscar_lista_valor(li, human.cpf, &alguem);
  printf("Consultar na lista valor: %d\n", h);
  printf("%d\n", alguem.cpf);
  int p = buscar_lista_pos(li, 1, &alguem);
  printf("Consultar na lista posição: %d\n", p);
  printf("%d\n", alguem.cpf);


  destruir_lista(li);
  return 0;
}
```

RESULTADOS:


Console
Shell
clang-7 -pthread -lm -o main listaDuplaEncad.c main.c
./main
Inserção no inicio da lista: 1
Inserção no inicio da lista: 1
Inserção no fim da lista: 1
Inserção no meio da lista: 1
Remoção no inicio da lista: 1
Remoção no fim da lista: 1
Remoção no meio da lista: 1
tamanho da lista: 1
Lista cheia: 0
Lista vazia: 0
Consultar na lista valor: 1
38758756
Consultar na lista posição: 1
38758756

LISTA ENCADEADA CIRCULAR:
ARQUIVOS CRIADOS:



File: listaEncadCircular.h (interface)

listaEncadCircular.h

```
1   #ifndef LISTA_ENCAD_CIRCULAR_H
2   #define LISTA_ENCAD_CIRCULAR_H
3
4   #define MAX 100
5   struct pessoa {
6     int idade;
7     char nome[30];
8     int cpf;
9   };
10
11  typedef struct pessoa Pessoa;
12  typedef struct elemento* Lista;
13
14  Lista* criar_lista();
15  void destruir_lista(Lista* li);
16  int lista_cheia(Lista *li);
17  int lista_vazia(Lista *li);
18  int tamanho_lista(Lista *li);
19  int inserir_inicio(Lista* li, Pessoa dados);
20  int inserir_final(Lista* li, Pessoa dados);
21  int inserir_meio(Lista* li, Pessoa dados);
22  int remover_inicio(Lista* li);
23  int remover_fim(Lista* li);
24  int remover_meio(Lista* li, int);
25  int buscar_lista_valor(Lista* li, int, Pessoa *dados);
26  int buscar_lista_pos(Lista* li, int, Pessoa *dados);
27
28
29  #endif
```

File: listaEncadCircular.c (implementação)

listaEncadCircular.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "listaEncadCircular.h"

struct elemento{
  struct pessoa dados;
  struct elemento *prox;
};

typedef struct elemento Elem;

//Criando lista
Lista* criar_lista(){
  Lista* li = (Lista*) malloc(sizeof(Lista));
  if(li != NULL){
    *li = NULL;
  }
  return li;
}

//Excluindo lista
void destruir_lista(Lista* li){
  if(!lista_vazia(li)){
    Elem *aux, *no = *li;
    while((*li) != no->prox){
      aux = no;
      no = no->prox;
      free(aux);
    }
    free(no);
    free(li);
```

listaEncadCircular.c

```
32    }
33  }
34
35  //Verificando se a lista está cheia
36  int lista_cheia(Lista *li){
37    return 0;
38  }
39
40  //Verificando se a lista está vazia
41  int lista_vazia(Lista *li){
42    if(li == NULL || (*li) == NULL ){
43      return 1;
44    }
45    return 0;
46  }
47
48  //Exibindo o tamanho da lista
49  int tamanho_lista(Lista *li){
50    if(li == NULL){
51      return 0;
52    }
53    if(lista_vazia(li)){
54      return 0;
55    }
56    Elem* no = *li;
57    int count = 0;
58    do{
59      count++;
60      no = no->prox;
61    } while(no != (*li));
62
```

listaEncadCircular.c

```
  return count;
};

//Inserindo elementos no inicio da lista
int inserir_inicio(Lista* li, Pessoa dados){
    if(li == NULL){
      return 0; //se a lista nao existir
    }
    Elem* no = (Elem*) malloc(sizeof(Elem));
    if(no == NULL){
      return 0;
    }
    no->dados = dados;
    if(lista_vazia(li)){
      *li = no;
      no->prox = no;
    } else {
      no->prox = *li;
      Elem* aux = *li;
      while(aux->prox != (*li)){
        aux = aux->prox;
      }
      aux->prox = no;
      (*li) = no;
    }
    return 1;
}

//Inserindo elementos no final da lista
int inserir_final(Lista* li, Pessoa dados){
  if(li == NULL){
```

listaEncadCircular.c

```
94      return 0; //se a lista nao existir
95    }
96    Elem* no = (Elem*) malloc(sizeof(Elem));
97    if(no == NULL){
98      return 0;
99    }
100   no->dados = dados;
101   if(lista_vazia(li)){
102     *li = no;
103     no->prox = no;
104   } else {
105     no->prox = *li;
106     Elem* aux = *li;
107     while(aux->prox != (*li)){
108       aux = aux->prox;
109     }
110     aux->prox = no;
111   }
112   return 1;
113 }
114
115 //Inserindo elementos no meio da lista de forma ordenada
116 int inserir_meio(Lista* li, Pessoa dados){
117   if(li == NULL){
118     return 0;
119   }
120   Elem* no = (Elem*) malloc(sizeof(Elem));
121   if(no == NULL){
122     return 0;
123   }
124   no->dados = dados;
```

listaEncadCircular.c

```
125     if(lista_vazia(li)){ //se for vazia
126       no->prox = no;
127       *li = no;
128     } else {
129       //verificando se será inserido no começo
130       if((*li)->dados.cpf > dados.cpf){
131         Elem* aux = *li;
132         while(aux->prox != (*li)){
133           aux = aux->prox;
134         }
135         aux->prox = no;
136         no->prox = (*li);
137         *li = no;
138         return 1;
139       }
140       Elem *ant = *li, *atual = (*li)->prox;
141       while(atual != (*li) && atual->dados.cpf < dados.cpf){
142         ant = atual;
143         atual = atual->prox;
144       }
145       no->prox = atual;
146       ant->prox = no;
147     }
148     return 1;
149   }
150
151   //Removendo elementos do inicio da lista
152   int remover_inicio(Lista* li){
153     if(li == NULL){
154       return 0;
155     }
```

listaEncadCircular.c

```c
  if(lista_vazia(li)){
    return 0;
  }
  if((*li)->prox == *li){ //verificando se temos apenas 1 elemento.
    free(*li);
    *li = NULL;
    return 1;
  }
  Elem *no = *li, *atual = *li;
  while(atual->prox != (*li)){
    atual = atual->prox;
  }
  atual->prox = no->prox;
  *li = no->prox;
  free(no);
  return 1;
}

//Removendo elementos do fim da lista
int remover_fim(Lista* li){
  if(li == NULL){
    return 0;
  }
  if(lista_vazia(li)){
    return 0;
  }
  if((*li)->prox == *li){ //verificando se temos apenas 1 elemento.
    free(*li);
    *li = NULL;
```

listaEncadCircular.c

```
185       return 1;
186     }
187     Elem* no = *li, *ant = *li;
188     while(no->prox != (*li)){
189       ant = no;
190       no = no->prox;
191     }
192     ant->prox = *li;
193     free(no);
194     return 1;
195   }
196
197   //Removendo elementos do meio da lista de forma ordenada
198   int remover_meio(Lista* li, int cpf){
199     if(li == NULL){
200       return 0;
201     }
202     if(lista_vazia(li)){
203       return 0;
204     }
205     Elem* no = *li;
206     //verificando se será removido o inicio
207     if(no->dados.cpf == cpf){
208       if((*li)->prox == *li){ //verificando se temos apenas 1
       elemento.
209         free(no);
210         *li = NULL;
211         return 1;
212       }
213       Elem* aux = *li;
214       while(aux->prox != (*li)){
```

listaEncadCircular.c

```
215         aux = aux->prox;
216       }
217       aux->prox = (*li)->prox;
218       *li = (*li)->prox;
219       free(no);
220       return 1;
221     }
222     Elem* ant;
223     no = no->prox;
224     while(no != (*li) && no->dados.cpf != cpf){
225       ant = no;
226       no = no->prox;
227     }
228     if(no == *li){
229       return 0; //not found
230     }
231     ant->prox = no->prox;
232     free(no);
233     return 1;
234   }
235
236   //Buscando elemento na lista por valor
237   int buscar_lista_valor(Lista* li, int cpf, struct pessoa
      *dados){
238     if(li == NULL){
239       return 0;
240     }
241     if(lista_vazia(li)){
242       return 0;
243     }
244     Elem* no = *li;
```

listaEncadCircular.c

```
275     *dados = no->dados;
276     return 1;
277   }
```

File: main.c (chamada das funções)

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEncadCircular.h"

int main(void) {
  Lista *li;
  Pessoa alguem;
  Pessoa human;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 38758756;

  li = criar_lista();
  int x = inserir_inicio(li, human);
  printf("Inserção no inicio da lista: %d\n", x);
  int u = inserir_inicio(li, human); //inserção de dois
  elementos
  printf("Inserção no inicio da lista: %d\n", u);
  int y = inserir_final(li, human);
  printf("Inserção no fim da lista: %d\n", y);
  int z = inserir_meio(li, human);
  printf("Inserção no meio da lista: %d\n", z);
  printf("\n\n");

  int t = remover_inicio(li);
  printf("Remoção no inicio da lista: %d\n", t);
  int w = remover_fim(li);
  printf("Remoção no fim da lista: %d\n", w);
  int k = remover_meio(li, human.cpf);
```

main.c

```c
    printf("Remoção no meio da lista: %d\n", k);
    printf("\n\n");

    int j = tamanho_lista(li);
    printf("tamanho da lista: %d\n", j);
    int n = lista_cheia(li);
    printf("Lista cheia: %d\n", n);
    int a = lista_vazia(li);
    printf("Lista vazia: %d\n", a);
    int h = buscar_lista_valor(li, human.cpf, &alguem);
    printf("Consultar na lista valor: %d\n", h);
    printf("%d\n", alguem.cpf);
    int p = buscar_lista_pos(li, 1, &alguem);
    printf("Consultar na lista posição: %d\n", p);
    printf("%d\n", alguem.cpf);


    destruir_lista(li);
    return 0;
}
```

RESULTADOS:

Console Shell

```
> clang-7 -pthread -lm -o main listaEncadCircular.c main.c
> ./main
Inserção no inicio da lista: 1
Inserção no inicio da lista: 1
Inserção no fim da lista: 1
Inserção no meio da lista: 1


Remoção no inicio da lista: 1
Remoção no fim da lista: 1
Remoção no meio da lista: 1


tamanho da lista: 1
Lista cheia: 0
Lista vazia: 0
Consultar na lista valor: 1
38758756
Consultar na lista posição: 1
38758756
>
```

Nota-se que os resultados obtidos são iguais em todos os tipos de listas, porém, a implementação é diferente para cada lista. Isso é uma demonstração de como o Tipo Abstrato de Dados (TAD) funciona.

4. Com as estruturas da questão anterior escreva uma função que:

a. Dada uma lista L1, inverta a lista e armazene em uma outra lista L2.

LISTA SEQUENCIAL ESTÁTICA

OBS: Note que a operação de inserir no início tem a seguinte complexidade **O(n) = n:**

```
54  //inserindo elemento no inicio
55  int inserir_inicio(Lista* li, struct pessoa dados){
56    if(li == NULL){1
57      return 0;
58    } else if(lista_cheia(li)){1
59      return 0;
60    }
61    if(lista_vazia(li)){1
62      li->dados[0] = dados;
63      li->qtd++;
64      return 1;
65    }
66    int aux = li->qtd;1
67    while(aux != 0){(n)
68      li->dados[aux] = li->dados[aux-1];1
69      aux--;1
70    }
71    li->dados[0] = dados;1
72    li->qtd++;1
73    return 1;
74  }
```

f(n) = 1 + 1 + 1 + 1 + ((n)*(1+1)) + 1 + 1 = **2n + 6**.

O(n) = n.

Operação de inversão

```
177  Lista* inverter_lista(Lista* li){
178    if(li == NULL){1
179      return 0;
180    }
181    if(lista_vazia(li)){1
182      return 0;
183    }
184    Lista* l2cp;1
185    l2cp = criar_lista();1
186    for(int i = 0; i < li->qtd; i++){(n)
187      inserir_inicio(l2cp, li->dados[i]);n
188    }
189    return l2cp;1
190  };
```

f(n) = 1 + 1 + 1 + 1 + (n*n) + 1 **= n^2 + 5.**

O(n) = n^2

File: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEstatica.h"

int main(void) {
  Lista *li;
  Lista *l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;


  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 38758756;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 999999999;

  li = criar_lista();
  inserir_inicio(li, human);
  inserir_meio(li, human2);
  inserir_meio(li, human2);

  //olhando os elementos da lista Li
  printf("Mostrando li:\n");
  consultar_lista_pos(li, 1, &alguem);
  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
```

main.c

```
32  consultar_lista_pos(li, 2, &alguem);
33  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);
34  consultar_lista_pos(li, 3, &alguem);
35  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);
36
37
38  //Olhando os elementos da lista l2
39  l2 = inverter_lista(li);
40  printf("Operação de inversão funcionou\n");
41  consultar_lista_pos(l2, 1, &alguem);
42  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
43  consultar_lista_pos(l2, 2, &alguem);
44  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
45  consultar_lista_pos(l2, 3, &alguem);
46  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);
47
48
49
50  destruir_lista(li);
```

RESULTADOS:

Console Shell

```
> clang-7 -pthread -lm -o main listaEstatica.c main
> ./main
Mostrando li:
Quem é o primeiro elemento: 38758756
Quem é o segundo elemento: 999999999
Quem é o terceiro elemento: 999999999

Operação de inversão funcionou
Quem é o primeiro elemento: 999999999
Quem é o segundo elemento: 999999999
Quem é o terceiro elemento: 38758756

>
```

COMPLEXIDADE O(n) = n^2.

LISTA DUPLAMENTE ENCADEADA CIRCULAR

OBS: Note que a operação de inserir no início tem a seguinte complexidade **O(n) = 1:**

```
//Inserindo elementos no inicio da lista
int inserir_inicio(Lista* li, Pessoa dados){
    if(li == NULL){1
      return 0; //se a lista nao existir
    }

    Elem* no = (Elem*) malloc(sizeof(Elem));1
    if(no == NULL){1
      return 0;
    }

    no->ant = NULL;1
    no->dados = dados;1
    no->prox = (*li);1
    if((*li) != NULL){1
      (*li)->ant = no;1//apontando o primeiro elemento
    }                  //depois do primeiro, depois
    *li = no;1         //mudando o point do 1º elem.
    return 1;
}
```

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 9.

O(n) = 1.

```
int inverter_lista(Lista* li, Lista *l2){
  if(li == NULL){1
    return 0;
  }
  if(lista_vazia(li)){1
    return 0;
  }
  Lista* l2cp;1
  l2cp = criar_lista();1
  Elem* atual = *li;1
  while(atual->prox != NULL){(n-1)
    inserir_inicio(l2cp , (atual)->dados);1
    atual = atual->prox;1
  }
  inserir_inicio(l2cp , (atual)->dados);1
  *l2 = *l2cp;1
  free(l2cp);1
  return 1;
};
```

f(n) = 1 + 1 + 1 + 1 + 1 + ((n-1)*(1+1)) + 1 + 1 + 1 = 2n - 2 + 8 = **2n + 6.**
O(n) = n.

File: main.c

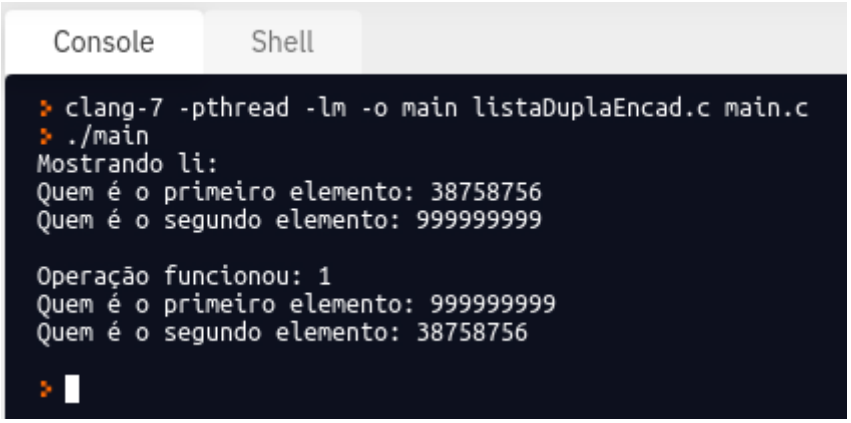
main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaDuplaEncad.h"

int main(void) {
  Lista *li;
  Lista l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 38758756;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 999999999;

  li = criar_lista();
  inserir_inicio(li, human);
  inserir_meio(li, human2);

  //olhando os elementos da lista Li
  printf("Mostrando li:\n");
  buscar_lista_pos(li, 1, &alguem);
  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
  buscar_lista_pos(li, 2, &alguem);
  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);
```

main.c

```c

  //Olhando os elementos da lista l2
  int x = inverter_lista(li, &l2);
  printf("Operação funcionou: %d\n", x);
  buscar_lista_pos(&l2, 1, &alguem);
  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
  buscar_lista_pos(&l2, 2, &alguem);
  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);



  destruir_lista(li);
```

RESULTADOS:



COMPLEXIDADE: O(n) = n

LISTA ENCADEADA CIRCULAR

OBS: Note que a operação de inserir no início tem a seguinte complexidade **O(n) = n:**

```
//Inserindo elementos no inicio da lista
int inserir_inicio(Lista* li, Pessoa dados){
    if(li == NULL){ 1
      return 0; //se a lista nao existir
    }
    Elem* no = (Elem*) malloc(sizeof(Elem));1
    if(no == NULL){ 1
      return 0;
    }
    no->dados = dados; 1
    if(lista_vazia(li)){1
      *li = no;
      no->prox = no;
    } else {
      no->prox = *li; 1
      Elem* aux = *li; 1
      while(aux->prox != (*li)){ (n)
        aux = aux->prox;1
      }
      aux->prox = no; 1
      (*li) = no; 1
    }
    return 1;
}
```

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + (n*1) + 1 + 1 = n + 9.

O(n) = n.

```
279  int inverter_lista(Lista* li, Lista *l2){
280    if(li == NULL){ 1
281      return 0;
282    }
283    if(lista_vazia(li)){ 1
284      return 0;
285    }
286    Lista* l2cp; 1
287    l2cp = criar_lista();1
288    Elem* atual = *li; 1
289    while(atual->prox != *li){ (n-1)
290      inserir_inicio(l2cp , (atual)->dados);n
291      atual = atual->prox;1
292    }
293    inserir_inicio(l2cp , (atual)->dados);n//INSERINDO O ÚLTIMO
294    *l2 = *l2cp;1
295    free(l2cp);1
296    return 1;
297  };
```

f(n) = 1 + 1 + 1 + 1 + 1 + ((n-1)*(n+1)) + n + 1 + 1 = n^2 + 1 + n + 7 = **n^2 + n + 8**.

O(n) = n^2.

File: main.c

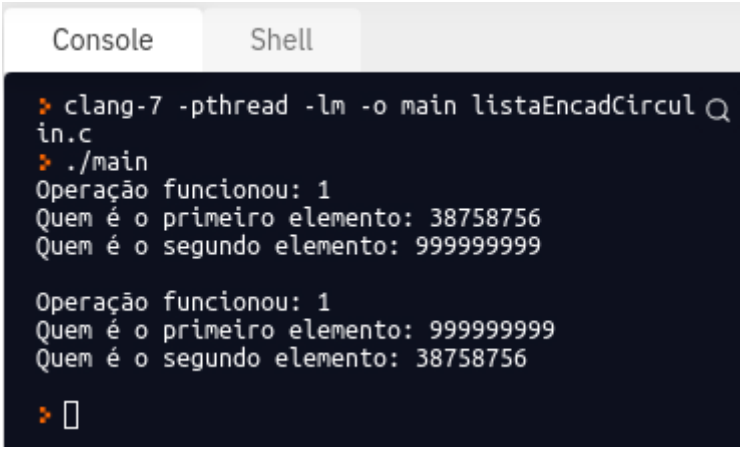
main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEncadCircular.h"

int main(void) {
  Lista *li;
  Lista l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;


  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 38758756;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 999999999;

  li = criar_lista();
  inserir_inicio(li, human);
  inserir_meio(li, human2);

  //olhando os elementos da lista Li
  printf("Mostrando li:\n");
  buscar_lista_pos(li, 1, &alguem);
  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
  buscar_lista_pos(li, 2, &alguem);
```

main.c

```c
  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);

  //Olhando os elementos da lista l2
  int x = inverter_lista(li, &l2);
  printf("Operação funcionou: %d\n", x);
  buscar_lista_pos(&l2, 1, &alguem);
  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
  buscar_lista_pos(&l2, 2, &alguem);
  printf("Quem é o segundo elemento: %d\n\n", alguem.cpf);



  destruir_lista(li);
```

RESULTADOS:



COMPLEXIDADE: O(n) = n²

---

b. Dada uma lista L1, crie uma cópia dela em L2 eliminando os valores repetidos.

LISTA SEQUENCIAL ESTÁTICA

OBS: Note que a operação de inserir elemento no final da lista tem a seguinte complexidade **O(1):**

```
//inserindo elemento no fim
int inserir_final(Lista* li, struct pessoa dados){
  if(li == NULL){1
    return 0;
  } else if(lista_cheia(li)){1
    return 0;
  }
  li->dados[li->qtd] = dados;1
  li->qtd++;1
  return 1;
}
//inserindo no meio de forma ordenada
```

f(n) = 1 + 1 + 1 + 1 = 4.

O(1) = 1.

OBS: Note que a operação de consulta de elemento na lista pelo valor tem a seguinte complexidade **O(n):**

```
219  //Buscando elemento na lista por valor
220  int buscar_lista_valor(Lista* li, int cpf, struct pessoa
     *dados){
221    if(li == NULL){1
222      return 0;
223    }
224    if(lista_vazia(li)){1
225      return 0;
226    }
227    Elem* no = *li;1
228    while(no != NULL && no->dados.cpf != cpf){(n)
229      no = no->prox;1
230    }
231    if(no == NULL){1
232      return 0;
233    }
234    *dados = no->dados;1
235    return 1;
236  }
```

f(n) = 1 + 1 + 1 + (n*1) + 1 + 1 = n+5.

O(n) = n.

Operação de remoção de repetidos:

```
192  //Removendo elementos repetidos e retornando outra lista sem eles
193  Lista* remover_repetidos(Lista* li){
194    if(li == NULL){1
195      return 0;
196    }
197    if(lista_vazia(li)){1
198      return 0;
199    }
200    Pessoa tem;1
201    Lista* l2cp = criar_lista();1
202    for(int i = 0; i < li->qtd; i++){(n)//passando pelos elementos da
       lista li.
203      if(!consultar_lista_valor(l2cp, (*li).dados[i].cpf, &tem)){(n+1)
204        //verificando se existe elemento já adicionado.
205        inserir_final(l2cp, (*li).dados[i]);1
206      };
207    }
208    return l2cp;
209  };
```

f(n) = 1 + 1 + 1 + 1 +((n)*(n+1)) = n^2 + n + 4.

O(n^2) = n^2

File: main.c

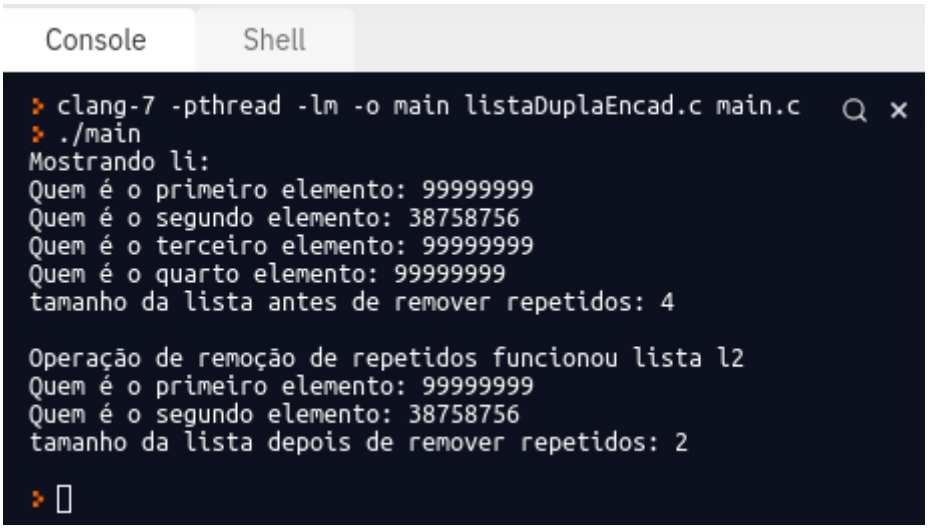
main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEstatica.h"

//PARA TESTAR CADA FUNCIONALIDADE, DESCOMENTE OS TRECHOS DE CÓDIGO ABAIXO DO COMENTÁRIO QUE O EXPLICA.

int main(void) {
  Lista *li;
  Lista *l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;
  Pessoa human3;
  Pessoa human4;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 99999999;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 38758756;

  char teste2[30] = "Lula";
  human3.nome[29] = nomeVar[29];
  human3.idade = 19;
  human3.cpf = 99999999;
  char teste3[30] = "Lula";
```

main.c

```
human4.nome[29] = nomeVar[29];
human4.idade = 19;
human4.cpf = 99999999;

li = criar_lista();
inserir_inicio(li, human);
inserir_final(li, human2);
inserir_final(li, human3);
inserir_final(li, human4);

//UTILIZADOS PARA TESTAR REMOÇÃO DE REPETIDOS

//olhando os elementos antes da remoção lista Li
printf("Mostrando li:\n");
consultar_lista_pos(li, 1, &alguem);
printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
consultar_lista_pos(li, 2, &alguem);
printf("Quem é o segundo elemento: %d\n", alguem.cpf);
consultar_lista_pos(li, 3, &alguem);
printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
consultar_lista_pos(li, 4, &alguem);
printf("Quem é o quarto elemento: %d\n", alguem.cpf);
printf("tamanho da lista antes de remover repetidos: %d\n\n",
tamanho_lista(li));

l2 = remover_repetidos(li);
//Olhando os elementos da lista l2
printf("Operação de remoção de repetidos funcionou lista l2\n");
consultar_lista_pos(l2, 1, &alguem);
printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
consultar_lista_pos(l2, 2, &alguem);
printf("Quem é o segundo elemento: %d\n", alguem.cpf);
int ok = consultar_lista_pos(l2, 3, &alguem);
if(ok)
  printf("Quem é o terceiro elemento (qtd inacessível): %d\n",
  alguem.cpf);
printf("tamanho da lista depois de remover repetidos: %d\n\n",
tamanho_lista(l2));
```

RESULTADOS:



COMPLEXIDADE O(n) = n^2

LISTA DUPLAMENTE ENCADEADA

OBS: Note que a operação de inserir no final tem a seguinte complexidade **O(n) = n:**

```
83  //Inserindo elementos no final da lista
84  int inserir_final(Lista* li, Pessoa dados){
85    if(li == NULL){1
86      return 0; //se a lista nao existir
87    }
88    Elem* no = (Elem*) malloc(sizeof(Elem));1
89    if(no == NULL){ 1
90      return 0;
91    }
92    no->dados = dados;1
93    no->prox = NULL;1
94    if(lista_vazia(li)){ 1
95      no->ant = NULL;
96      *li = no;
97      return 1;
98    }
99    Elem* aux = *li;1
100   while(aux->prox != NULL){ n
101     aux = aux->prox;1
102   }
103   aux->prox = no;1
104   no->ant = aux;1
105   return 1;
106 }
```

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + n*1 + 1 + 1 = **n + 9**.

O(n) = n.

OBS: Note que a operação de busca por valor tem a seguinte complexidade **O(n) = n:**

```
219  //Buscando elemento na lista por valor
220  int buscar_lista_valor(Lista* li, int cpf, struct pessoa *dados){
221    if(li == NULL){1
222      return 0;
223    }
224    if(lista_vazia(li)){1
225      return 0;
226    }
227    Elem* no = *li;1
228    while(no != NULL && no->dados.cpf != cpf){n
229      no = no->prox;1
230    }
231    if(no == NULL){1
232      return 0;
233    }
234    *dados = no->dados;1
235    return 1;
236  }
```

f(n) = 1 + 1 + 1 + n*1 + 1 + 1 = **n + 5**.

O(n) = n.

Operação de remoção de repetidos:

```
282  int remover_repetidos(Lista* li, Lista *l2){
283    if(li == NULL){1
284      return 0;
285    }
286    if(lista_vazia(li)){1
287      return 0;
288    }
289    Lista* l2cp;1
290    Pessoa tem;1
291    l2cp = criar_lista();1
292    Elem* atual = (*li);1
293    while(atual->prox != NULL){n
294      if(!buscar_lista_valor(l2cp, atual->dados.cpf, &tem)){(n+1)
295        inserir_final(l2cp, atual->dados);n//verificando se o elemento
           existe antes de inserir na nova lista.
296      }
297      atual = atual->prox;1
298    }
299    //verificando se o último valor está na lista para inseri-lo na
       lista. Caso já esteja, não é inserido.
300    if(!buscar_lista_valor(l2cp, atual->dados.cpf, &tem)){(n+1)
301      inserir_final(l2cp, atual->dados);n
302    }
303    *l2 = *l2cp;1
304    return 1;
305  };
```

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + ((n)*((n+1) + n + 1) + (n+1) + n + 1 = 6 + $2n^2$ + 2n + 2n + 2 **= $2n^2$ + 4n + 8.**

O(n^2) = n^2

File: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaDuplaEncad.h"

//PARA TESTAR CADA FUNCIONALIDADE, DESCOMENTE OS TRECHOS DE CÓDIGO ABAIXO DO COMENTÁRIO QUE O EXPLICA.

int main(void) {
  Lista *li;
  Lista l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;
  Pessoa human3;
  Pessoa human4;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 99999999;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 38758756;

  char teste2[30] = "Lula";
  human3.nome[29] = nomeVar[29];
  human3.idade = 19;
  human3.cpf = 99999999;
  char teste3[30] = "Lula";
```

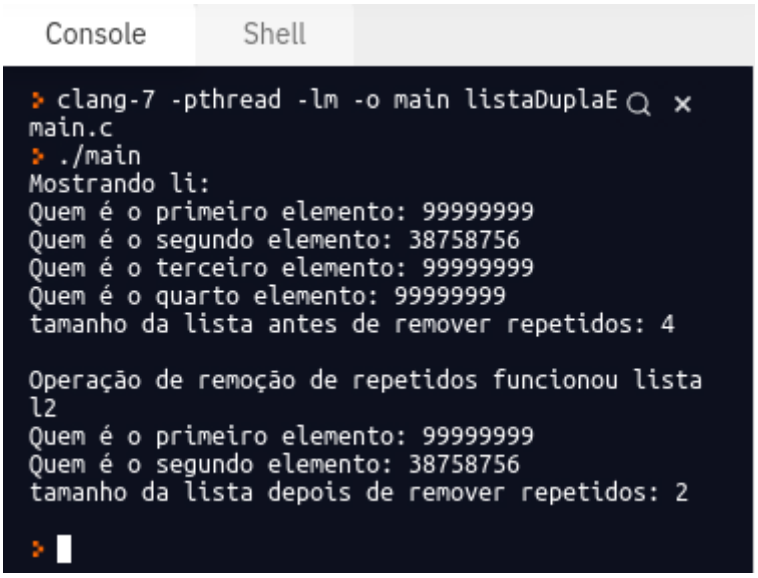
main.c

```
31  human4.nome[29] = nomeVar[29];
32  human4.idade = 19;
33  human4.cpf = 99999999;
34
35  li = criar_lista();
36  inserir_inicio(li, human);
37  inserir_final(li, human2);
38  inserir_final(li, human3);
39  inserir_final(li, human4);
40
41  //UTILIZADOS PARA TESTAR REMOÇÃO DE REPETIDOS
42
43  //olhando os elementos antes da remoção lista Li
44  printf("Mostrando li:\n");
45  buscar_lista_pos(li, 1, &alguem);
46  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
47  buscar_lista_pos(li, 2, &alguem);
48  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
49  buscar_lista_pos(li, 3, &alguem);
50  printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
51  buscar_lista_pos(li, 4, &alguem);
52  printf("Quem é o quarto elemento: %d\n", alguem.cpf);
53  printf("tamanho da lista antes de remover repetidos: %d\n\n",
    tamanho_lista(li));
54
55  remover_repetidos(li, &l2);
56  //Olhando os elementos da lista l2
57  printf("Operação de remoção de repetidos funcionou lista l2\n");
58  buscar_lista_pos(&l2, 1, &alguem);
59  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
60  buscar_lista_pos(&l2, 2, &alguem);
```

main.c

```
61  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
62  int ok = buscar_lista_pos(&l2, 3, &alguem);
63  if(ok)
64    printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
65  printf("tamanho da lista depois de remover repetidos: %d\n\n",
    tamanho_lista(&l2));
```

RESULTADOS:



COMPLEXIDADE $O(n^2) = n^2$.

LISTA ENCADEADA CIRCULAR

OBS: Note que a operação de inserir no final tem a seguinte complexidade **O(n) = n:**

```
91  //Inserindo elementos no final da lista
92  int inserir_final(Lista* li, Pessoa dados){
93    if(li == NULL){1
94      return 0; //se a lista nao existir
95    }
96    Elem* no = (Elem*) malloc(sizeof(Elem));1
97    if(no == NULL){1
98      return 0;
99    }
100   no->dados = dados;1
101   if(lista_vazia(li)){1
102       *li = no;
103       no->prox = no;
104   } else {
105       no->prox = *li;1
106       Elem* aux = *li;1
107       while(aux->prox != (*li)){n
108         aux = aux->prox;1
109       }
110       aux->prox = no;1
111   }
112   return 1;
113 }
```

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + n*1 + 1 = **n + 8**.

O(n) = n.

OBS: Note que a operação de busca por valor na lista tem a seguinte complexidade **O(n) = n:**

```
236  //Buscando elemento na lista por valor
237  int buscar_lista_valor(Lista* li, int cpf, struct pessoa *dados){
238    if(li == NULL){1
239      return 0;
240    }
241    if(lista_vazia(li)){1
242      return 0;
243    }
244    Elem* no = *li;1
245    while(no->prox != (*li) && no->dados.cpf != cpf){n
246      no = no->prox;1
247    }
248    if(no->dados.cpf != cpf){1
249      return 0;
250    }
251    *dados = no->dados;1
252    return 1;
253  }
```

f(n) = 1 + 1 + 1 + n*1 + 1 + 1 = **n + 5.**

O(n) = n.

Operação de remoção de repetidos:

```
299  int remover_repetidos(Lista* li, Lista *l2){
300    if(li == NULL){1
301      return 0;
302    }
303    if(lista_vazia(li)){1
304      return 0;
305    }
306    Lista* l2cp;1
307    Pessoa tem;1//Variavel apenas para chamar a função de consulta
308    l2cp = criar_lista();1
309    Elem* atual = (*li);1
310    while(atual->prox != *li){n
311      if(!buscar_lista_valor(l2cp, atual->dados.cpf, &tem)){ n+1
312        inserir_final(l2cp, atual->dados);n//verificando se o
           elemento existe antes de inserir na nova lista.
313      }
314      atual = atual->prox;1
315    }
316    //verificando se o último valor é igual ao primeiro para inserir
       o último elemento na lista. Caso seja, não é inserido.
317    if(!buscar_lista_valor(l2cp, atual->dados.cpf, &tem)){ n+1
318        inserir_final(l2cp, atual->dados); n
319    }
320    *l2 = *l2cp;1
321    return 1;
322  };
```

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + ((n)*((-n+1) + n + 1) + (n + 1) + n + 1 = 6 + $2n^2$ + 2n + 2n + 2 **=**

$2n^2$ + 4n + 8.

O(n^2) = n^2.

File: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEncadCircular.h"

//PARA TESTAR CADA FUNCIONALIDADE, DESCOMENTE OS TRECHOS DE CÓDIGO ABAIXO DO COMENTÁRIO QUE O EXPLICA.

int main(void) {

  Lista *li;
  Lista l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;
  Pessoa human3;
  Pessoa human4;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 99999999;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 38758756;

  char teste2[30] = "Lula";
  human3.nome[29] = nomeVar[29];
  human3.idade = 19;
  human3.cpf = 99999999;
```

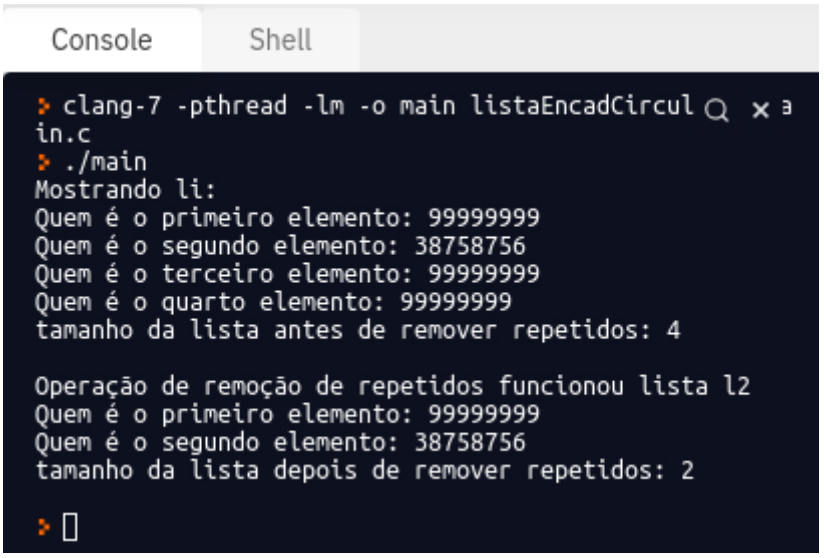
main.c

```
31  char teste3[30] = "Lula";
32  human4.nome[29] = nomeVar[29];
33  human4.idade = 19;
34  human4.cpf = 99999999;
35
36  li = criar_lista();
37  inserir_inicio(li, human);
38  inserir_final(li, human2);
39  inserir_final(li, human3);
40  inserir_final(li, human4);
41
42  //UTILIZADOS PARA TESTAR REMOÇÃO DE REPETIDOS
43
44  //olhando os elementos antes da remoção lista Li
45  printf("Mostrando li:\n");
46  buscar_lista_pos(li, 1, &alguem);
47  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
48  buscar_lista_pos(li, 2, &alguem);
49  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
50  buscar_lista_pos(li, 3, &alguem);
51  printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
52  buscar_lista_pos(li, 4, &alguem);
53  printf("Quem é o quarto elemento: %d\n", alguem.cpf);
54  printf("tamanho da lista antes de remover repetidos: %d\n\n",
    tamanho_lista(li));
55
56  remover_repetidos(li, &l2);
57  //Olhando os elementos da lista l2
58  printf("Operação de remoção de repetidos funcionou lista l2\n");
59  buscar_lista_pos(&l2, 1, &alguem);
60  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
```

main.c

```
61  buscar_lista_pos(&l2, 2, &alguem);
62  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
63  int ok = buscar_lista_pos(&l2, 3, &alguem);
64  if(ok)
65    printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
66  printf("tamanho da lista depois de remover repetidos: %d\n\n",
    tamanho_lista(&l2));
```

RESULTADOS:



COMPLEXIDADE $O(n^2) = n^2$.

---

c. Dadas duas listas, L1 e L2, verificar se as duas são iguais.

LISTA SEQUENCIAL ESTÁTICA
Operação de comparação

```
210  //verificando se as listas são iguais
211  int verificar_igualdade(Lista* li, Lista* l2){
212    if(li == NULL){1
213      return 0;
214    };
215    if(l2 == NULL){1
216      return 0;
217    };
218    int count = 0;1
219    if(li->qtd == l2->qtd){1
220      for(int i = 0; i < li->qtd; i++){n
221        if(li->dados[i].cpf == l2->dados[i].cpf){1
222          count ++;1
223        }
224      }
225      if(count == li->qtd){1
226        return 1;
227      } else {
228        return 0;
229      }
230    } else {
231      return 0;
232    }
233  };
```

f(n) = 1 + 1 + 1 + 1 + n*(1 + 1) + 1 = 2n + 5.
O(n) = n.

File: listaEstatica.h (interface).

```
30   int verificar_igualdade(Lista* li, Lista* l2);
```

File: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEstatica.h"

//PARA TESTAR CADA FUNCIONALIDADE, DESCOMENTE OS TRECHOS DE CÓDIGO ABAIXO DO COMENTÁRIO QUE O EXPLICA.

int main(void) {
  Lista *li;
  Lista *l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;
  Pessoa human3;
  Pessoa human4;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 99999999;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 38758756;

  char teste2[30] = "Lula";
  human3.nome[29] = nomeVar[29];
  human3.idade = 19;
  human3.cpf = 99999999;
  char teste3[30] = "Lula";
```

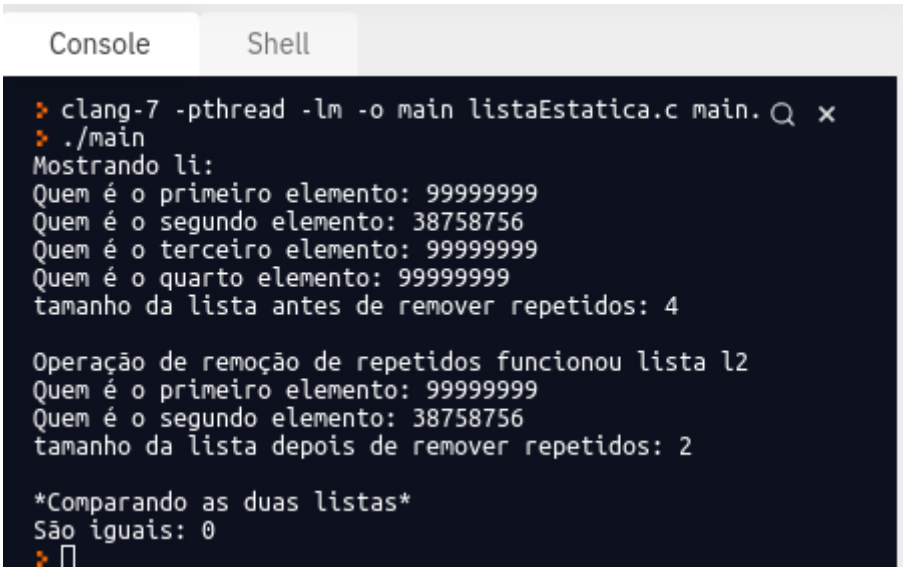
main.c

```
31  human4.nome[29] = nomeVar[29];
32  human4.idade = 19;
33  human4.cpf = 99999999;
34
35  li = criar_lista();
36  inserir_inicio(li, human);
37  inserir_final(li, human2);
38  inserir_final(li, human3);
39  inserir_final(li, human4);
40
41  //UTILIZADOS PARA TESTAR REMOÇÃO DE REPETIDOS
42
43  //olhando os elementos antes da remoção lista Li
44  printf("Mostrando li:\n");
45  consultar_lista_pos(li, 1, &alguem);
46  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
47  consultar_lista_pos(li, 2, &alguem);
48  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
49  consultar_lista_pos(li, 3, &alguem);
50  printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
51  consultar_lista_pos(li, 4, &alguem);
52  printf("Quem é o quarto elemento: %d\n", alguem.cpf);
53  printf("tamanho da lista antes de remover repetidos: %d\n\n",
    tamanho_lista(li));
54
55  l2 = remover_repetidos(li);
56  //Olhando os elementos da lista l2
57  printf("Operação de remoção de repetidos funcionou lista l2\n");
58  consultar_lista_pos(l2, 1, &alguem);
59  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
60  consultar_lista_pos(l2, 2, &alguem);
```

main.c

```
61  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
62  int ok = consultar_lista_pos(l2, 3, &alguem);
63  if(ok)
64    printf("Quem é o terceiro elemento (qtd inacessível): %d\n",
    alguem.cpf);
65  printf("tamanho da lista depois de remover repetidos: %d\n\n",
    tamanho_lista(l2));
66
67  int verif = verificar_igualdade(li, l2);
68  printf("*Comparando as duas listas*\n");
69  printf("São iguais: %d\n", verif);
70
```

RESULTADOS:



COMPLEXIDADE O(n) = n.

LISTA DUPLAMENTE ENCADEADA

Operação de comparação

```
307  //verificando se as listas são iguais
308  int verificar_igualdade(Lista* li, Lista l2){
309  if(li == NULL){ 1
310      return 0;
311    };
312    if(l2 == NULL){1
313      return 0;
314    };
315    Elem* no = *li, *aux = l2; //pegando as duas listas 1
316    if(tamanho_lista(li) != tamanho_lista(&l2)){n//verificando se são do
       mesmo tamanho, se nao, já sabemos que são diferentes.
317      return 0;
318    }
319    while(no->prox != NULL){ n//percorrendo toda a lista até o último
       elemento
320      if(no->dados.cpf != aux->dados.cpf){1
321        return 0;
322      }
323      no = no->prox; 1
324      aux = aux->prox; 1
325    }
326    //fazendo a ultima verificação para o ultimo elemento.
327    if(no->dados.cpf != aux->dados.cpf){1
328        return 0;
329    }
330    return 1;
331  };
```

f(n) = 1 + 1 + 1 + n + (n)*(1 + 1 + 1) + 1 = 4n + 4.

O(n) = n.

File: listaDuplaEncad.h

```
int verificar_igualdade(Lista* li, Lista l2);
```

File: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaDuplaEncad.h"

//PARA TESTAR CADA FUNCIONALIDADE, DESCOMENTE OS TRECHOS DE CÓDIGO
ABAIXO DO COMENTÁRIO QUE O EXPLICA.

int main(void) {
  Lista *li;
  Lista l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;
  Pessoa human3;
  Pessoa human4;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 99999999;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 38758756;

  char teste2[30] = "Lula";
  human3.nome[29] = nomeVar[29];
  human3.idade = 19;
  human3.cpf = 99999999;

```

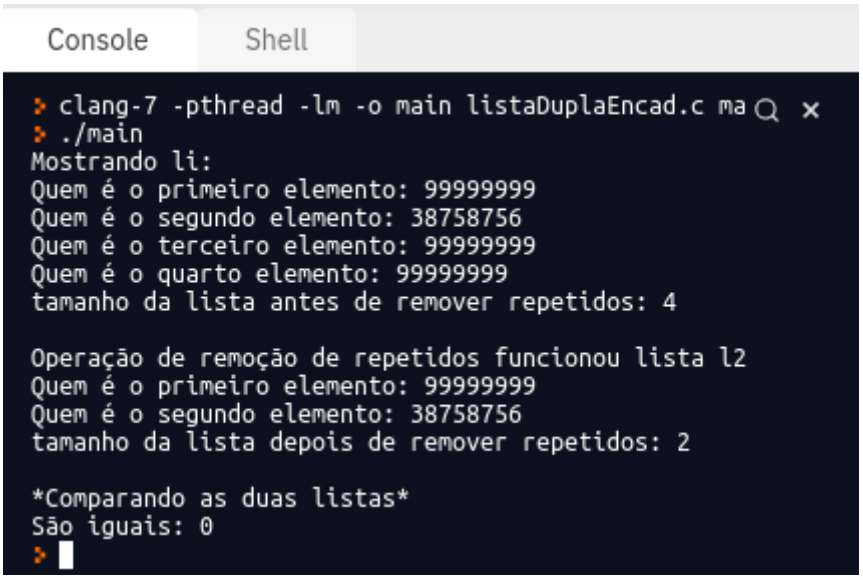
main.c

```c
char teste3[30] = "Lula";
human4.nome[29] = nomeVar[29];
human4.idade = 19;
human4.cpf = 99999999;

li = criar_lista();
inserir_inicio(li, human);
inserir_final(li, human2);
inserir_final(li, human3);
inserir_final(li, human4);

//UTILIZADOS PARA TESTAR REMOÇÃO DE REPETIDOS

//olhando os elementos antes da remoção lista Li
printf("Mostrando li:\n");
buscar_lista_pos(li, 1, &alguem);
printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
buscar_lista_pos(li, 2, &alguem);
printf("Quem é o segundo elemento: %d\n", alguem.cpf);
buscar_lista_pos(li, 3, &alguem);
printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
buscar_lista_pos(li, 4, &alguem);
printf("Quem é o quarto elemento: %d\n", alguem.cpf);
printf("tamanho da lista antes de remover repetidos: %d\n\n",
tamanho_lista(li));

remover_repetidos(li, &l2);

//Olhando os elementos da lista l2
printf("Operação de remoção de repetidos funcionou lista l2\n");
buscar_lista_pos(&l2, 1, &alguem);
```

main.c

```c
buscar_lista_pos(&l2, 2, &alguem);
printf("Quem é o segundo elemento: %d\n", alguem.cpf);
int ok = buscar_lista_pos(&l2, 3, &alguem);
if(ok)
  printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
printf("tamanho da lista depois de remover repetidos: %d\n\n",
tamanho_lista(&l2));

int verif = verificar_igualdade(li, l2);
printf("*Comparando as duas listas*\n");
printf("São iguais: %d\n", verif);
```

RESULTADOS:



COMPLEXIDADE O(n) = n.

LISTA ENCADEADA CIRCULAR
Operação de comparação

```
324  //verificando se as listas são iguais
325  int verificar_igualdade(Lista* li, Lista l2){
326  if(li == NULL){1
327      return 0;
328    };
329    if(l2 == NULL){1
330      return 0;
331    };
332    Elem* no = *li, *aux = l2;1//pegando as duas listas
333    if(tamanho_lista(li) != tamanho_lista(&l2)){n//verificando se
       são do mesmo tamanho, se nao, já sabemos que são diferentes.
334      return 0;
335    }
336    while(no->prox != *li){n//percorrendo toda a lista até o último
       elemento
337      if(no->dados.cpf != aux->dados.cpf){1
338        return 0;
339      }
340      no = no->prox;1
341      aux = aux->prox;1
342    }
343    //fazendo a ultima verificação para o ultimo elemento.
344    if(no->dados.cpf != aux->dados.cpf){1
345        return 0;
346    }
347    return 1;
348  };
```

f(n) = 1 + 1 + 1 + n + n*(1 + 1 + 1) + 1 = 4n + 4.
O(n) = n.

File: listaEncadCircular.h (interface)

```
28   int verificar_igualdade(Lista* li, Lista l2);
```

File: main.c

main.c

```c
#include <stdio.h>
#include <stdio.h>
#include "listaEncadCircular.h"

//PARA TESTAR CADA FUNCIONALIDADE, DESCOMENTE OS TRECHOS DE CÓDIGO ABAIXO DO COMENTÁRIO QUE O EXPLICA.

int main(void) {
  Lista *li;
  Lista l2;
  Pessoa alguem;
  Pessoa human;
  Pessoa human2;
  Pessoa human3;
  Pessoa human4;

  char nomeVar[30] = "Emerson";
  human.nome[29] = nomeVar[29];
  human.idade = 19;
  human.cpf = 99999999;

  char teste[30] = "Ana Luiza";
  human2.nome[29] = nomeVar[29];
  human2.idade = 19;
  human2.cpf = 38758756;

  char teste2[30] = "Lula";
  human3.nome[29] = nomeVar[29];
  human3.idade = 19;
  human3.cpf = 99999999;
  char teste3[30] = "Lula";
```

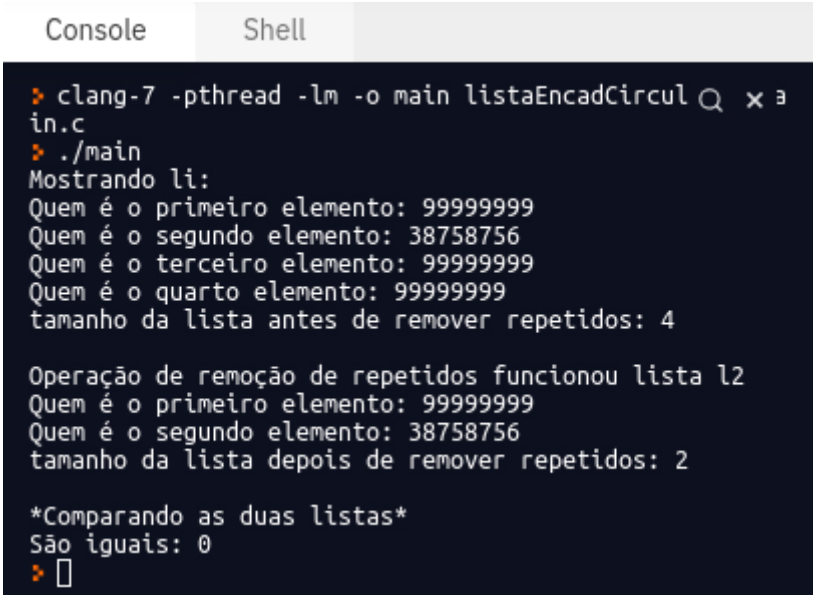
main.c

```
31  human4.nome[29] = nomeVar[29];
32  human4.idade = 19;
33  human4.cpf = 99999999;
34
35  li = criar_lista();
36  inserir_inicio(li, human);
37  inserir_final(li, human2);
38  inserir_final(li, human3);
39  inserir_final(li, human4);
40
41  //UTILIZADOS PARA TESTAR REMOÇÃO DE REPETIDOS
42
43  //olhando os elementos antes da remoção lista Li
44  printf("Mostrando li:\n");
45  buscar_lista_pos(li, 1, &alguem);
46  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
47  buscar_lista_pos(li, 2, &alguem);
48  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
49  buscar_lista_pos(li, 3, &alguem);
50  printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
51  buscar_lista_pos(li, 4, &alguem);
52  printf("Quem é o quarto elemento: %d\n", alguem.cpf);
53  printf("tamanho da lista antes de remover repetidos: %d\n\n",
    tamanho_lista(li));
54
55  remover_repetidos(li, &l2);
56  //Olhando os elementos da lista l2
57  printf("Operação de remoção de repetidos funcionou lista l2\n");
58  buscar_lista_pos(&l2, 1, &alguem);
59  printf("Quem é o primeiro elemento: %d\n", alguem.cpf);
60  buscar_lista_pos(&l2, 2, &alguem);
```

main.c

```
61  printf("Quem é o segundo elemento: %d\n", alguem.cpf);
62  int ok = buscar_lista_pos(&l2, 3, &alguem);
63  if(ok)
64    printf("Quem é o terceiro elemento: %d\n", alguem.cpf);
65  printf("tamanho da lista depois de remover repetidos: %d\n\n",
    tamanho_lista(&l2));
66
67  int verif = verificar_igualdade(li, l2);
68  printf("*Comparando as duas listas*\n");
69  printf("São iguais: %d\n", verif);
```

RESULTADOS:



COMPLEXIDADE O(n) = n.