Atividade 4

```
/*****************************************************************************
//     listaLigadaD2.c
// Este programa gerencia listas lineares duplamente ligadas
//   (implementacao dinamica).
// As listas gerenciadas podem ter um numero arbitrario de elementos.
// Não usaremos sentinela nesta estrutura.
*****************************************************************************/
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0

typedef int bool;
typedef int TIPOCHAVE;

typedef struct {
  TIPOCHAVE chave;
  // outros campos...
} REGISTRO;


typedef struct aux{
  REGISTRO reg;
  struct aux *ant, *prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
  PONT inicio;
} LISTA;

/* Inicializando da lista ligada (a lista jah esta criada e eh apontada
pelo endereco em l) */
void inicializarLista(LISTA* l){
  l->inicio = NULL;
} /* inicializarLista */

/* Exibindo da lista sequencial */
void exibirLista(LISTA* l){
  PONT end = l->inicio;
  printf("Lista: \" ");
  while (end != NULL){
    printf("%d ", end->reg.chave); // soh lembrando TIPOCHAVE = int
    end = end->prox;
  }
  printf("\"\n");
```

```
} /* exibirLista */


/* Verifica consistencia da lista duplamente ligada */
bool verificarListaDuplamenteLigada(LISTA* l){
  bool res = true;
  if (!l->inicio) return res;
  PONT ant;
  PONT pos = l->inicio;
  if (pos->ant){
    res = false;
    printf("Problema na verificacao (1): endereco anterior do primeiro elemento difere d
  }
  while (pos != NULL){
    ant = pos;
    pos = pos->prox;
    if (pos && pos->ant != ant){
      printf("TESTE %p x %p.\n", pos->ant, ant);
      printf("Problema na verificacao (1): endereco anterior do elemento %i difere do en
      res = false;
    }
  }
  return res;
} /* verificarListaDuplamenteLigada */

/* Retornar o tamanho da lista (numero de elementos) */
int tamanho(LISTA* l) {
  PONT end = l->inicio;
  int tam = 0;
  while (end != NULL){
    tam++;
    end = end->prox;
  }
  return tam;
} /* tamanho */

/* Retornar o tamanho em bytes da lista. Neste caso, isto depende do numero
   de elementos que estao sendo usados.   */
int tamanhoEmBytes(LISTA* l) {
  return(tamanho(l)*sizeof(ELEMENTO))+sizeof(LISTA); // sizeof(LISTA) = sizeof(PONT) poi
} /* tamanhoEmBytes */

/* Busca sequencial (lista ordenada ou nao) */
PONT buscaSequencial(LISTA* l, TIPOCHAVE ch){
  PONT pos = l->inicio;
  while (pos != NULL){
    if (pos->reg.chave == ch) return pos;
    pos = pos->prox;
  }
  return NULL;
```

```
} /* buscaSequencial */


/* Busca sequencial (lista ordenada) */
PONT buscaSeqOrd(LISTA* l, TIPOCHAVE ch){
  PONT pos = l->inicio;
  while (pos != NULL && pos->reg.chave < ch) pos = pos->prox;
  if (pos != NULL && pos->reg.chave == ch) return pos;
  return NULL;
} /* buscaSequencial */

/* Busca sequencial - funcao de exclusao (retorna no endereço *ant o indice do
   elemento anterior ao elemento que está sendo buscado [ant recebe o elemento
   anterior independente do elemento buscado ser ou não encontrado]) */
PONT buscaSeqExc(LISTA* l, TIPOCHAVE ch, PONT* ant){
  *ant = NULL;
  PONT atual = l->inicio;
  while (atual != NULL && atual->reg.chave<ch){
    *ant = atual;
    atual = atual->prox;
  }
  if ((atual != NULL) && (atual->reg.chave == ch)) return atual;
  return NULL;
}
/* buscaSequencialExc */

/* Exclusão do elemento de chave indicada */
bool excluirElemLista(LISTA* l, TIPOCHAVE ch){
  PONT ant, i;
  i = buscaSeqExc(l,ch,&ant);
  if (i == NULL) return false;
  if (ant == NULL) l->inicio = i->prox;
  else ant->prox = i->prox;
  if (i->prox) i->prox->ant = ant;
  free(i);
  return true;
} /* excluirElemLista */

/* Destruindo da lista sequencial
   libera a memoria de todos os elementos da lista*/
void reinicializarLista(LISTA* l) {
  PONT end = l->inicio;
  while (end != NULL){
    PONT apagar = end;
    end = end->prox;
    free(apagar);
  }
  l->inicio = NULL;
} /* destruirLista */
```

```
//************************** Atividade 4 **********************

/* Inserção em lista ordenada sem duplicação */
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {

  TIPOCHAVE ch = reg.chave;
  PONT ant, prox, i;
  i = buscaSeqExc(l,ch,&ant);
  if (i != NULL)  return false;
  i = (PONT) malloc(sizeof(ELEMENTO));
  i->reg = reg;
  if (ant == NULL) { // o novo elemento serah o 1o da lista
    i->prox = l->inicio;
    l->inicio = i;
    i->ant = NULL; //"o campo anterior dele deverá valer NULL"
  } else {  // inserção após um elemento já existente
    i->prox = ant->prox;
    ant->prox = i;
    i->ant = ant;
  }

  if (i->prox != NULL){
        i->prox->ant = l->inicio;
  }

  return true;
} /* inserirElemListaOrd */

//************************** Atividade 4 **********************

/* retornarPrimeiro - retorna o endereco do primeiro elemento da lista e (caso
   a lista nao esteja vazia) retorna a chave desse elemento na memoria
   apontada pelo ponteiro ch */
PONT retornarPrimeiro(LISTA* l, TIPOCHAVE *ch){
  if (l->inicio != NULL) *ch = l->inicio->reg.chave;
  return l->inicio;
} /* retornarPrimeiro */

/* retornarUltimo - retorna o endereco do ultimo elemento da lista e (caso
   a lista nao esteja vazia) retorna a chave desse elemento na memoria
   apontada pelo ponteiro ch */
PONT retornarUltimo(LISTA* l, TIPOCHAVE *ch){
  PONT ultimo = l->inicio;
  if (l->inicio == NULL) return NULL;
  while (ultimo->prox != NULL) ultimo = ultimo->prox;
  *ch = ultimo->reg.chave;
  return ultimo;
} /* retornarUltimo */
```