```c
/*****************************************************************************
//     listaSequencial.c
// Este programa gerencia listas lineares sequenciais.
// As listas gerenciadas podem ter MAX elementos (posicoes de 0 a MAX-1 no
//   arranjo A). Alocamos MAX+1 posicoes no arranjo para possibilitar a
//   criacao de um elemento sentinela.
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX 50
#define ERRO -1
#define true 1
#define false 0

typedef int bool;

typedef int TIPOCHAVE;

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

typedef struct {
  REGISTRO A[MAX+1];
  int nroElem;
} LISTA;

/* Inicialização da lista sequencial (a lista já está criada e é apontada
pelo endereço em l) */
void inicializarLista(LISTA* l){
  l->nroElem = 0;
} /* inicializarLista */

/* Exibição da lista sequencial */
void exibirLista(LISTA* l){
  int i;
  printf("Lista: \" ");
  for (i=0; i < l->nroElem; i++)
    printf("%i ", l->A[i].chave); // só lembrando TIPOCHAVE = int
  printf("\"\n");
} /* exibirLista */

/* Retornar o tamanho da lista (numero de elementos "validos") */
int tamanho(LISTA* l) {
  return l->nroElem;
} /* tamanho */

/* Retornar o tamanho em bytes da lista. Neste caso, isto nao depende do numero
   de elementos que estao sendo usados, pois a alocacao de memoria eh estatica.
   A priori, nao precisariamos do ponteiro para a lista, vamos utiliza-lo apenas
   porque teremos as mesmas funcoes para listas ligadas.
*/
int tamanhoEmBytes(LISTA* l) {
  return sizeof(LISTA);
} /* tamanhoEmBytes */

/* Retornar a chave do primeiro elemento da lista sequencial (caso haja) e ERRO
```

```
61:    caso a lista esteja vazia */
62: TIPOCHAVE primeiroElem(LISTA* l){
63:   if(l->nroElem > 0) return l->A[0].chave;
64:   else return ERRO; // lista vazia
65: } /* primeiroElem */
66:
67: /* Retornar a chave do ultimo elemento da lista sequencial (caso haja) e ERRO
68:    caso a lista esteja vazia */
69: TIPOCHAVE ultimoElem(LISTA* l) {
70:   if(l->nroElem > 0) return l->A[l->nroElem-1].chave;
71:   else return ERRO; // lista vazia
72: } /* ultimoElem */
73:
74: /* Retornar a chave do elemento que está na posição n da LISTA. Lembre-se que as p
75:    arranjo A vao de 0 a MAX-1  */
76: TIPOCHAVE enesimoElem(LISTA* l, int n) {
77:   if( (n >= 0) && (n < l->nroElem)) return l->A[n].chave ;
78:   else return ERRO;
79: } /* enesimoElem */
80:
81: /* Reinicializar a estrutura */
82: void reinicializarLista(LISTA* l) {
83:   l->nroElem = 0;
84: } /* reinicializarLista */
85:
86: /* Inserção "direta" na iésima posição (posicao i do arranjo A).
87:    Funciona da mesma maneira de um insertionSort: deve-se deslocar todos os
88:    elementos a partir da iesima posicao e entao se insere o novo elemento. */
89: bool inserirElemLista(LISTA* l, REGISTRO reg, int i){
90:   int j;
91:   if ((l->nroElem >= MAX) || (i < 0) || (i > l->nroElem))
92:      return(false); // lista cheia ou índice inválido
93:   for (j = l->nroElem; j > i; j--) l->A[j] = l->A[j-1];
94:   l->A[i] = reg;
95:   l->nroElem++;
96:   return true;
97: } /* inserirElemLista */
98:
99: /* Busca sequencial em lista ordenada ou não SEM SENTINELA */
100: int buscaSequencial(LISTA* l, TIPOCHAVE ch) {
101:   int i = 0;
102:   while (i < l->nroElem){
103:     if(ch == l->A[i].chave) return i; // achou
104:     else i++;
105:   }
106:   return ERRO; // não achou
107: } /* buscaSequencial */
108:
109:
110: /* Exclusão do elemento cuja chave seja igual a ch */
111: bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
112:   int pos, j;
113:   pos = buscaSequencial(l,ch);
114:   if(pos == ERRO) return false; // não existe
115:   for(j = pos; j < l->nroElem-1; j++) l->A[j] = l->A[j+1];
116:   l->nroElem--;
117:   return true;
118: } /* excluirElemLista */
119:
120:
```

```c
/* Busca sequencial em lista COM SENTINELA (vetor criado com MAX+1 posições) */
int buscaSentinela(LISTA* l, TIPOCHAVE ch) {
  int i = 0;
  l->A[l->nroElem].chave = ch; // sentinela
  while(l->A[i].chave != ch) i++;
  if (i > l->nroElem -1) return ERRO; // não achou
  else return i;
} /* buscaSentinela */


int main(){
  LISTA lista;
  inicializarLista(&lista);
  exibirLista(&lista);
  printf("Numero de elementos na lista: %i.\n",tamanho(&lista));
  printf("Tamanho da lista (em bytes): %i.\n",tamanhoEmBytes(&lista));
  REGISTRO reg;
  reg.chave = 9;
  inserirElemLista(&lista,reg,0);
  exibirLista(&lista);
  reg.chave=3;
  inserirElemLista(&lista,reg,1);
  reg.chave=4;
  inserirElemLista(&lista,reg,2);
  reg.chave=1;
  inserirElemLista(&lista,reg,3);
  reg.chave=12;
  inserirElemLista(&lista,reg,2);
  exibirLista(&lista);
  printf("Numero de elementos na lista: %i.\n",tamanho(&lista));
  printf("Tamanho da lista (em bytes): %i.\n",tamanhoEmBytes(&lista));
  printf("Chave 4 encontrada na posicao: %i do arranjo A.\n",buscaSequencial(&lista,4));
  printf("Chave 4 encontrada na posicao: %i do arranjo A.\n",buscaSentinela(&lista,4));
  if (excluirElemLista(&lista,4)) printf("Exclusao bem sucedida: 4.\n");
  if (excluirElemLista(&lista,8)) printf("Exclusao bem sucedida: 8.\n");
  if (excluirElemLista(&lista,9)) printf("Exclusao bem sucedida: 9.\n");
  exibirLista(&lista);
  printf("Numero de elementos na lista: %i.\n",tamanho(&lista));
  printf("Tamanho da lista (em bytes): %i.\n",tamanhoEmBytes(&lista));
  reinicializarLista(&lista);
  exibirLista(&lista);
  printf("Numero de elementos na lista: %i.\n",tamanho(&lista));
  printf("Tamanho da lista (em bytes): %i.\n",tamanhoEmBytes(&lista));
  return 0;
}

/* SAIDA DO PROGRAMA:
Lista: " "
Numero de elementos na lista: 0.
Tamanho da lista (em bytes): 208.
Lista: " 9 "
Lista: " 9 3 12 4 1 "
Numero de elementos na lista: 5.
Tamanho da lista (em bytes): 208.
Chave 4 encontrada na posicao: 3 do arranjo A.
Chave 4 encontrada na posicao: 3 do arranjo A.
Exclusao bem sucedida: 4.
Exclusao bem sucedida: 9.
Lista: " 3 12 1 "
```

```
181: Numero de elementos na lista: 3.
182: Tamanho da lista (em bytes): 208.
183: Lista: " "
184: Numero de elementos na lista: 0.
185: Tamanho da lista (em bytes): 208.
186: */
```