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

void create(int n){
    vetor = (int*) realloc (vetor, n*(sizeof(int)));
    tam = -1;
    if (vetor == NULL) {
        printf ("\nERRO!\n");
        exit (1);
    } else
        printf("\n\nEspaço alocado com sucesso!\n");

}

void cria_vaga(int indice){        //Função local, complementa a função do TAD "add".

    for (int i=tam;i>indice;i--){
        if (i>0){
            vetor[i]=vetor[(i-1)];
        }
    }
}

int add(int valor, int espaco){
    posicao = 0;

    if (isFull(espaco) == TRUE){
        printf("\n\nVetor cheio!!!\n");
        return FALSE;
    }

    if (isempty() == TRUE){
        tam++;
        vetor[tam] = valor;
        return TRUE;
    }
    tam++;
    while (posicao < tam){
        if (vetor[posicao] < valor){
            posicao++;

        }else {
            cria_vaga(posicao);
            vetor[posicao]=valor;
            return TRUE;
        }
    }
```

```
49      vetor[posicao] = valor;
50
51      return TRUE;
52  }
53
54  void reorganiza(int indice){        //Função local que complementa a função do
    TAD "remove_item".
55
56      for (int i=indice;i<tam;i++){
57          vetor[i]=vetor[(i+1)];
58      }
59      tam--;
60  }
61
62  int remove_item(int chave){
63      posicao = 0;
64      if (isempty() == TRUE){
65          printf("\n\nVazio! Nada a remover!\n");
66          return FALSE;
67      }
68      while (posicao < tam+1){
69          if (vetor[posicao] == chave){
70              reorganiza(posicao);
71              return TRUE;
72              }
73          posicao++;
74      }
75      return FALSE;
76  }
77
78  int size_lista(){
79      return tam+1;
80  }
81
82  int linearSearch(int chave){
83      for (int i=0;i<tam+1;i++){
84          if (vetor[i] == chave)
85              return i;
86      }
87      printf("\n\nChave não encontrada!\n");
88      return -404;
89  }
90
91  int bynarySearch(int chave){
92      int esq = 0, dir = tam;
93
94      while (esq <= dir){
95          int meio = (esq + dir)/2;
96
```

```
97          if (vetor[meio] == chave)
98              return meio;
99          else if (vetor[meio] > chave)
100             dir = meio -1;
101         else
102             esq = meio + 1;
103     }
104     printf("\n\nChave não encontrada!\n");
105     return -404;
106 }
107
108
109 int isempty(){  -> O(1) constante
110     if (tam < 0)
111         return TRUE;
112     else
113         return FALSE;
114 }
115
116 int isFull(int espaco){ -> O(1) constante
117     if ((tam+1) == espaco)
118         return TRUE;
119     else
120         return FALSE;
121 }
122
123 int clear_vetor(){ -> O(1) constante
124     free(vetor);
125     return TRUE;
126 }
127
128 void print_all(){ -> O(n)
129     printf("\n"); O(1)
130     printf("["); O(1)
131     for (int i=0;i<tam+1;i++){  O(n)
132         printf(" %d ,", vetor[i]); O(1)
133     }
134     printf("]"); O(1)
135
136 }
137
```

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

lista_cde *create (){

    lista_cde *lista = (lista_cde *)malloc(sizeof (lista_cde));

    if(lista != NULL) {
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho = 0;
    }

    return lista;
}

int add_item(lista_cde *lista, int valor){

    no *pnovo = (no*) malloc(sizeof(no));

    if(pnovo != NULL) {
        pnovo->valor = valor;
        pnovo->proximo = NULL;

        if(lista->inicio == NULL) {
            lista->inicio = pnovo;
        }else {
            lista->fim->proximo = pnovo;
        }

        lista->fim = pnovo;
        lista->tamanho++;
        printf("\n\nValor add %d", pnovo->valor);
        return TRUE;
    }
    return FALSE;
}

int remove_item(lista_cde *lista, int chave){

    if (!isempty(lista)) {

        no *alvo = lista->inicio;
        no *anterior;

        while(alvo != NULL && alvo->valor != chave) {
            anterior = alvo;
            alvo = alvo->proximo;
```

```
50          }
51
52          if(alvo != NULL) {          O(1)
53              if(alvo != lista->inicio) { O(1)
54                  anterior->proximo = alvo->proximo; O(1)
55              } else {
56                  lista->inicio = alvo->proximo;   O(1)
57              }
58
59              if(alvo == lista->fim) {     O(1)
60                  lista->fim = anterior;   O(1)
61              }
62
63          lista->tamanho--;   O(1)
64          free(alvo);         O(1)
65
66          return TRUE;   O(1)
67          }
68      }
69      return FALSE;   O(1)
70  }
71
72  int size_item(lista_cde *lista){   O(1) constante
73      return lista->tamanho;
74  }
75
76  int find_item(lista_cde *lista, int valor){  => O(n)
77      int counter=0;   O(1)
78
79      if(isempty(lista))   O(1)
80          printf("Lista esta' vazia.\n"); O(1)
81      else{
82
83          no *alvo = lista->inicio;   O(1)
84
85          while(alvo != NULL){      O(n)
86                                  //printf("\n\n %d %d \n\n", alvo->valor,
                                    alvo->proximo->valor);
87              if (valor == alvo->valor){   O(1)
88                  printf("\n\nEncontrado no indice %d \n\n", counter); O(1)
89              return counter;   O(1)
90              }
91
92          alvo = alvo->proximo;   O(1)
93          counter++;          O(1)
94          }
95      }
96
97      return -1;   O(1)
```

```
98  }
99
100 int isempty(lista_cde *lista){ => O(1) constante
101     return lista->tamanho <= 0;
102 }
103
104 int clear_item(lista_cde *lista){ => O(1) constante
105
106     free(lista); O(1)
107     lista->tamanho = 0; O(1)
108
109     return TRUE; O(1)
110 }
111
112 int printAll (lista_cde *l) { => O(n)
113     no *p = l->inicio; O(1)
114                        // faz p apontar para o nó inicial
115                        // testa se lista não é vazia
116     if ((p != NULL) && (l->tamanho != 0)) {  O(1)
117                        // percorre os elementos até alcançar novamente o início
118         for (int i=0;i<(l->tamanho);i++){  O(n)
119             printf(" %d -", p->valor); O(1) // imprime informação do nó
120             p = p->proximo;  O(1)
121         }
122         printf("\n");  O(1)
123         return TRUE;  O(1)
124     }
125     printf("\n\nLista Vazia!\n\n"); O(1)
126     return FALSE;  O(1)
127
128 }
129
130
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "No.h"

No* createNo(int v, No* proximo){

   No * n = (No*) malloc(sizeof(No));

   n->valor = v;
   n->proximo = proximo;

   return n;

}

void printNo(No* n){

   if(n != NULL && n->proximo != NULL)
       printf("No [valor: %i, proximo: %i]\n", n->valor, n->proximo->valor);
   else if(n != NULL)
       printf("No [valor: %i, proximo: NULL]\n", n->valor);
   else
       printf("NULL\n");

}

void freeNo(No* n){
   free(n);
}
```

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

ListaEncadeada *create() {
    ListaEncadeada *lista = (ListaEncadeada *)malloc(sizeof (ListaEncadeada));

    if(lista != NULL) {
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho = 0;
    }

    return lista;
}


int add(ListaEncadeada *lista, int valor) {

    No* pnovo = (No*) malloc(sizeof(No));

    if(pnovo != NULL) {
            pnovo->valor = valor;
            pnovo->proximo = NULL;

            if(lista->inicio == NULL) {
               lista->inicio = pnovo;
            } else {
               lista->fim->proximo = pnovo;
            }

            lista->fim = pnovo;
            lista->tamanho++;
            return 1;
    }

    return 0;
}

int remover(ListaEncadeada *lista, int chave){

    if (!isEmpty(lista)) {

        No *alvo = lista->inicio;
        No *anterior;

        while(alvo != NULL && alvo->valor != chave) {
            anterior = alvo;
            alvo = alvo->proximo;
```

```
50          }
51
52          if(alvo != NULL) {
53              if(alvo != lista->inicio) {
54                  anterior->proximo = alvo->proximo;
55              } else {
56                  lista->inicio = alvo->proximo;
57              }
58
59              if(alvo == lista->fim) {
60                  lista->fim = anterior;
61              }
62
63              lista->tamanho--;
64              freeNo(alvo);
65              return 1;
66
67          }
68      }
69      return 0;
70  }
71
72  int size_p(ListaEncadeada *lista){
73      return lista->tamanho;
74  }
75
76  int finder(ListaEncadeada* lista, int valor){
77      int counter=0;
78
79      if(isEmpty(lista))
80              printf("Lista esta' vazia.\n");
81          else{
82              No *alvo = lista->inicio;
83
84              while(alvo != NULL){
85                  //printf("\n\n %d %d \n\n", alvo->valor, alvo->proximo->valor);
86                  if (valor == alvo->valor){
87                      printf("\n\nEncontrado na posição %d \n\n", counter);
88                      return counter;
89                  }
90
91                  alvo = alvo->proximo;
92                  counter++;
93              }
94          }
95
96      return -1;
97  }
98
```

```
99  int isEmpty(ListaEncadeada* lista){ =>Constante
100     return lista->tamanho <= 0; O(1)
101 }
102
103 void printListaEncadeada(ListaEncadeada* lista){ =>
104
105         printf("\n----------IMPRIMINDO LISTA ------------\n\n"); O(n)
106
107         printf("Lista [tamanho: %i, limite: quantidade de memo'ria aloca'vel]
            \n\n", lista->tamanho); O(1)
108
109         if(isEmpty(lista))          O(1)
110             printf("Lista esta' vazia.\n"); O(1)
111         else{
112             No *alvo = lista->inicio;   O(1)
113
114             while(alvo != NULL){        O(n)
115                 printNo(alvo);      O(1)
116                 alvo = alvo->proximo; O(1)
117             }
118         }
119
120 }
121
122 void clear_p(ListaEncadeada* lista){ => Constante
123
124     free(lista);        O(1)
125     lista->tamanho = 0; O(1)
126 }
127
```