Laboratório de Listas Encadeadas

1) Implemente o TAD ListaEncadeada utilizando um nó descritor como o mostrado abaixo

Padronização de erros

Sugestão para padronizar códigos de erro

```
#define SUCCESS 0
#define INVALID_NULL_POINTER -1
#define OUT_OF_MEMORY -2
#define OUT_OF_RANGE -3
#define ELEM_NOT_FOUND -4
….
```

// LinkedList.c

```
typedef struct list_node List_node;

struct list{
   List_node *head;
};

struct list_node{
    struct aluno data;
    List_node *next;
};
```

// LinkedList.h

```
typedef struct list List;


List* list_create();
int list_free(List *li);

// insere o aluno no início da lista
int list_push_front(List *li, struct aluno al);
// insere o aluno no final da lista
int list_push_back(List *li, struct aluno al);
// insere o aluno na posição 'pos' – essa posição inicia em 1
int list_insert(List *li, int pos, struct aluno al);
// ordem por matricula  na função que insere ordenado
int list_insert_sorted(List *li, struct aluno al);
```

```c
// retorna o tamanho da lista (valores negativos em caso de erro)
int list_size(List *li);



// retira da lista o primeiro aluno
int list_pop_front(List *li);
// retira da lista o último aluno
int list_pop_back(List *li);
// retira da lista o aluno de matricula 'mat'
int list_erase_data(List *li, int mat);
// retira da lista o aluno da posição 'pos' (posição inicia em 1)
int list_erase_pos(List *li, int pos);

// encontrar o aluno pela posição na lista (posição inicia em 1)
int list_find_pos(List *li, int pos, struct aluno *al);
// encontrar o aluno pelo número de matrícula
int list_find_mat(List *li, int nmat, struct aluno *al);
// retornar o aluno que está no início da lista
int list_front(List *li, struct aluno *al);
// retornar o aluno que está no final da lista
int list_back(List *li, struct aluno *al);
// dado um número de matrícula, retornar a posição a lista
int list_get_pos(List *li, int nmat, int *pos);
// imprime a lista (única função que permite printf!)
int list_print(List *li);
```

(em português, somente para consulta – não usar)

```c
Lista* list_create();
void libera_lista(Lista* li);
int consulta_lista_pos(Lista* li, int pos, struct aluno *al);
int consulta_lista_mat(Lista* li, int mat, struct aluno *al);
int insere_lista_final(Lista* li, struct aluno al);
int insere_lista_inicio(Lista* li, struct aluno al);
int insere_lista_pos(Lista* li, int pos, struct aluno al);
int insere_lista_ordenada(Lista* li, struct aluno al);
int remove_lista(Lista* li, int mat);
int remove_lista_inicio(Lista* li);
int remove_lista_final(Lista* li);
int tamanho_lista(Lista* li);
int lista_cheia(Lista* li);
int lista_vazia(Lista* li);
int imprime_lista(Lista* li);
```

2) Crie um novo TAD a partir do TAD do exercício anterior que inclua as seguintes modificações no nó descritor

```
struct list {
   List_node *head;
   int size; // quantidade de elementos na lista
   int sorted; // indica se a lista é ordenada por número de
matrícula: 0 – não ordenada ; 1 - ordenada
};
```

Caso uma lista seja considerada ordenada, o usuário só poderá utilizar a função `list_insert_sorted` para inserção. Caso contrário, só estarão disponíveis as funções de inserção no início, fim e em uma posição específica. Caso o usuário tente utilizar uma função não disponível (ou seja, tentar chamar a função `list_insert_sorted` quando a lista é desordenada), a função deve retornar um código de erro.

A indicação se a lista é ordenada ou não é feita durante a chamada da função que cria a lista (passar um booleano como parâmetro). Não possível alterar o status da lista após a criação.

O elemento *size* da *struct* da lista guarda o número de elementos na lista e deve estar sempre atualizada.