UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA - CCT
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – DCC
DCC302 – ESTRUTURA DE DADOS I
PROF. MSC. ACAUAN C. RIBEIRO

Nome(s): Lucas Prado Nota:

EXERCÍCIO - Aula 08 - Listas Duplamente Encadeadas

→ Tendo como base o código visto na Aula 08 – Listas Duplamente Encadeadas implemente:

1) Uma função que adicione um elemento ao final da lista.

```
List_add_last(List *L, int val)
```

2) Implemente uma função que remova um elemento da lista. Neste sentido a função deve procurar o elemento na lista (pode ser inicio, meio ou fim), remove-lo e atualizar os ponteiros para manter as propriedade de uma lista duplamente encadeada.

```
List_remove(List *L, int val)
```

Nome: Lucas Prado Nº Matricula: 2020018187

Exercício Aula 8 – Listas duplamente encadeada

Douby_list_linked.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// ------------ DEFINIÇÃO DAS ESTRUTURAS ----------------
typedef struct _doubly_node {
  int val;
  struct _doubly_node *next;
  struct _doubly_node *prev;
} Doubly_node, Node;

typedef struct _doubly_linked_list {
  Doubly_node *begin;
  Doubly_node *end;
  size_t size;
} Doubly_linked_list, List;

// ------------ CONSTRUTORES E DESTRUTORES DAS ESTRUTURAS -
------------
Node *create_node(int val) {
    Node *node = (Node *) malloc(sizeof(Node));
    node->val = val;
    node->next = NULL;
    node->prev = NULL;
    return node;
```

```
}

List *create_list() {
    List *L = (List *) malloc(sizeof(List));
    L->begin = NULL;
    L->end = NULL;
    L->size = 0;
    return L;
}

void destroy_list(List **L_ref) {
    List *L = *L_ref;
    Node *p = L->begin;
    Node *aux = NULL;
    while (p != NULL){
        aux = p;
        p = p->next;
        free(aux);
    }
    free(L);
    *L_ref = NULL;
}

// ------------ FUNÇÔES --------------

bool list_is_empty(List *L) {
    return L->size == 0;
}

void List_add_first(List *L, int val) {
    Node *p = create_node(val);
```

```
    // Caso 1: Lista vazia
    if(list_is_empty(L)) {
        L->end = p;
    } else { // Caso 2: Lista não está vazia
        p->next = L->begin;
        L->begin->prev = p;
    }
    L->begin = p;
    L->size++;
}

void List_add_last(List *L, int val) {
    Node *p = create_node(val);
    if(list_is_empty(L)) {
        L->begin = p;
        L->end = p;
    } else {
        L->end->next = p;
        p->prev = L->end;
        L->end = p;
    }
    L->size++;
}

void List_print(List* L) {
    Node* p = L->begin;
    printf("L-> ");
    while(p != NULL) {
        printf("%d-> ", p->val);
        p = p->next;
    }
```

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

    L->end == NULL ? printf("L->end = NULL\n") : printf("L->end = %d\n", L->end->val);

    printf("Size = %lu\n", L->size);
}

void List_remove(List* L, int val) {
    if (list_is_empty(L)) {
        printf("list is empty\n");
    }
    Node* p = L->begin;
    //caso 1: se tiver na cabeça da lista
    if (L->begin->val == val) {
        L->begin = p->next;

        //1.1: se tiver só um unico elemento
        if (L->end == p ){
            L->end = NULL;
        } else { //1.2: se tiver mais de um elemento
            L->begin->prev = NULL;
        }
        free(p);
        L->size--;
    }
    // caso 2: o elemento está no meio da lista
    // caso 3: o elemento está na calda da lista
    else {
        p = p->next;
```

```
        while (p != NULL) {
            if (p->val == val){
                p->prev->next = p->next;

                if (L->end == p) {
                    L->end = p->prev;

                }

                else {
                     p->next->prev = p->prev;

                }

                free(p);
                p = NULL;
                L->size--;
            }
            else {
                p = p->next;
            }
        }
    }
}

// ------------ Teste do codigo ------------

int main() {
    List *L = create_list();
    List_add_first(L, 4);
    List_add_first(L, 2);
```

```
    List_add_first(L, 10);
    List_add_last(L, 7);

    List_print(L);

    List_remove(L, 7);

    List_print(L);

    destroy_list(&L);

    return 0;
}
```