

# Roteiro 4 - Gabriel Souza de Oliveira - 222050042

1.1 -

```
#ifndef LSE_H
#define LSE_H
#include <stdio.h>
#include <stdlib.h>

#define MAX_TAM 100

typedef struct{
    int dados[MAX_TAM];
    int qtd;
}Lista;

Lista* criaLista(){
    Lista* lista = (Lista*)malloc(sizeof(Lista));
    if (lista != NULL)
        lista->qtd = 0;
    return lista;
}

void destroiLista(Lista* lista) {
    if (lista != NULL)
        free(lista);
}

int tamanhoLista(Lista* lista) {
    if (lista == NULL)
        return -1;
    return lista->qtd;
}

int listaCheia (Lista* lista) {
    if (lista == NULL)
        return -1;
    return (lista->qtd == MAX_TAM);
}

int listaVazia(Lista* lista) {
    if (lista == NULL)
        return -1;
```

```

        return (lista->qtd == 0);
    }

int insereFim(Lista *lista, int elemento) {
    if (lista == NULL)
        return 0;
    if (!listaCheia(lista)){
        lista->dados[lista->qtd] = elemento;
        lista->qtd++;
        printf ("Elemento inserido com sucesso!\n");
        return 1;
    } else {
        printf ("Elemento nao inserido - Lista cheia!\n");
        return 0;
    }
}

int insereInicio(Lista *lista, int elemento){
    if (lista == NULL)
        return 0;
    if (!listaCheia(lista)){
        int i;
        for (i = lista->qtd - 1; i >= 0; i++)
            lista->dados[i+1] = lista->dados[i];
        lista->dados[0] = elemento;
        lista->qtd++;
        printf ("Elemento inserido com sucesso!\n");
        return 1;
    } else{
        printf ("Elemento nao inserido - Lista cheia!\n");
        return 0;
    }
}

int removeFim (Lista *lista){
    if (lista == NULL)
        return 0;
    if (!listaVazia(lista)){
        lista->qtd--;
        printf ("Elemento removido com sucesso!\n");
        return 1;
    } else {
        printf("Nenhum elemento removido - Lista vazia!\n");
    }
}

```

```

        return 0;
    }
}

int removeInicio(Lista *lista) {
    if (lista == NULL)
        return 0;
    if (!listavazia(lista)){
        int i;
        for (int i=0; i<lista->qtd; i++)
            lista->dados[i] = lista->dados[i+1];
        lista->qtd--;
        printf ("Elemento removido com sucesso!\n");
        return 1;
    } else {
        printf ("Nenhum elemento removido - Lista vazia!\n");
        return 0;
    }
}

int imprimeLista(Lista *lista) {
    if (lista == NULL)
        return 0;
    if (listavazia(lista)){
        printf ("Lista vazia!\n");
        return 0;
    }
    printf ("Elementos:\n");
    int i;
    for (i=0; i<lista->qtd; i++){
        printf ("%d ", lista->dados[i]);
    }
    printf ("\n");
    return 1;
}

#endif // LSE_H

```

## 1.2 -

```

#include <stdio.h>
#include <stdlib.h>
#include "lista_sequencial_estatica.h"

```

```

int procura(Lista *lista, int elemento){
    for (int i=0; i < lista->qtd; i++){
        if (lista->dados[i] == elemento)
            return i;
    }
    return -1;
}

int main() {
    Lista* lista = criaLista();

    insereFim(lista, 13);
    insereFim(lista, 85);
    insereFim(lista, 25);
    insereFim(lista, 48);
    insereFim(lista, 8);

    imprimeLista(lista);
    printf ("Indice de 48: %d\n", procura(lista, 48));

    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output'
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> & .\exercicio1-2.exe
Elemento inserido com sucesso!
Elementos:
13 85 25 48 8
Indice de 48: 3

```

1.3 -

```

#include <stdio.h>
#include <stdlib.h>
#include "lista_sequencial_estatica.h"

int insereOrdenado(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;
    if (!listaCheia(lista)){
        for(int i=0; i <= lista->qtd; i++){
            if (lista->dados[i] > elemento){
                for(int j=lista->qtd; j > i; j--)

```

```

        lista->dados[j] = lista->dados[j-1];
        lista->dados[i] = elemento;
        lista->qtd++;
        return 1;
    }
}
return 0;
}

int main() {
    Lista* lista = criaLista();

    insereOrdenado(lista, 5);
    imprimeLista(lista);
    insereOrdenado(lista, 2);
    imprimeLista(lista);
    insereOrdenado(lista, 27);
    imprimeLista(lista);
    insereOrdenado(lista, 30);
    imprimeLista(lista);
    insereOrdenado(lista, 50);
    imprimeLista(lista);
    insereOrdenado(lista, 15);
    imprimeLista(lista);

    destroiLista(lista);
    return 0;
}
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output'
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> & .\exercicio1-3.exe'
Elementos:
5
Elementos:
2 5
Elementos:
2 5 27
Elementos:
2 5 27 30
Elementos:
2 5 27 30 50
Elementos:
2 5 15 27 30 50
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> []

```

1.4 -

```

#include <stdio.h>
#include <stdlib.h>
#include "lista_sequencial_estatica.h"

```

```

int removePrimeiro(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;
    if (listaVazia(lista))
        return 0;

    int i;
    for (i=0; i < lista->qtd; i++) {
        if (lista->dados[i] == elemento) {
            break;
        }
    }

    if (i < lista->qtd) {
        for (int j=i; j < lista->qtd-1; j++) {
            lista->dados[j] = lista->dados[j+1];
        }
        lista->qtd--;
    }

    return 1;
}

int main() {
    Lista* lista = criaLista();

    insereFim(lista, 10);
    insereFim(lista, 4);
    insereFim(lista, 27);
    insereFim(lista, 19);
    insereFim(lista, 41);
    insereFim(lista, 27);
    imprimeLista(lista);
    removePrimeiro(lista, 27);
    imprimeLista(lista);

    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output' & .\exercicio1-4.exe
Elemento inserido com sucesso!
Elementos:
10 4 27 19 41 27
Elementos:
10 4 19 41 27
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> []

```

## 2.1 -

```

#ifndef LSEC_H
#define LSEC_H
#include <stdlib.h>
#include <stdio.h>

typedef struct No{
    int info;
    struct No *prox;
} No;
typedef No *Lista;

Lista* criarLista(){
    Lista *lista;
    lista = (Lista*)malloc(sizeof(Lista));
    if (lista != NULL)
        *lista = NULL;
    return lista;
}

int listaVazia(Lista* lista){
    if (lista == NULL) // Erro ao alocar inicialmente
        return 1;
    if (*lista == NULL)
        return 1; // Lista Vazia = True
    return 0; // Lista Vazia = False
}

No* alocarNo(){
    return (No*)malloc(sizeof(No));
}

void liberarNo(No* q){
    free(q);
}

```

```

int insereInicio(Lista* lista, int elemento){
    if (lista == NULL)
        return 0;
    No* novo = alokarNo();
    if (novo == NULL)
        return 0;
    novo->info = elemento;
    novo->prox = *lista;
    *lista = novo;
    return 1;
}

int insereFim(Lista* lista, int elemento){
    if (lista == NULL)
        return 0;
    No* novo = alokarNo();
    if (novo == NULL)
        return 0;
    novo->info = elemento;
    novo->prox = NULL;
    if (listavazia(lista)){
        *lista = novo;
    } else{
        No* auxiliar = *lista;
        while (auxiliar->prox != NULL)
            auxiliar = auxiliar->prox;
        auxiliar->prox = novo;
    }
    return 1;
}

int removeInicio(Lista* lista){
    if (lista == NULL)
        return 0;
    if (listavazia(lista))
        return 0;
    No* auxiliar = *lista;
    *lista = auxiliar->prox;
    liberarNo(auxiliar);
    return 1;
}

```

```

int removeFim (Lista* lista){
    if (lista == NULL)
        return 0;
    if (listaVazia(lista))
        return 0;
    No* anterior, *auxiliar = *lista;
    while (auxiliar->prox != NULL) {
        anterior = auxiliar;
        auxiliar = auxiliar->prox;
    }
    if (auxiliar == *lista)
        *lista = auxiliar->prox;
    else
        anterior->prox = auxiliar->prox;
    liberarNo(auxiliar);
    return 1;
}

void imprimeLista(Lista* lista) {
    if (lista == NULL)
        return;
    if (listaVazia(lista)) {
        printf ("Lista vazia!\n");
        return;
    }
    printf ("Elementos:\n");
    No* auxiliar = *lista;
    while (auxiliar != NULL) {
        printf ("%d ", auxiliar->info);
        auxiliar = auxiliar->prox;
    }
    printf ("\n");
}

void destroiLista (Lista* lista){
    if (lista != NULL){
        No* auxiliar;
        while ((*lista) != NULL) {
            auxiliar = *lista;
            *lista = (*lista)->prox;
            liberarNo(auxiliar);
        }
        free(lista);
    }
}

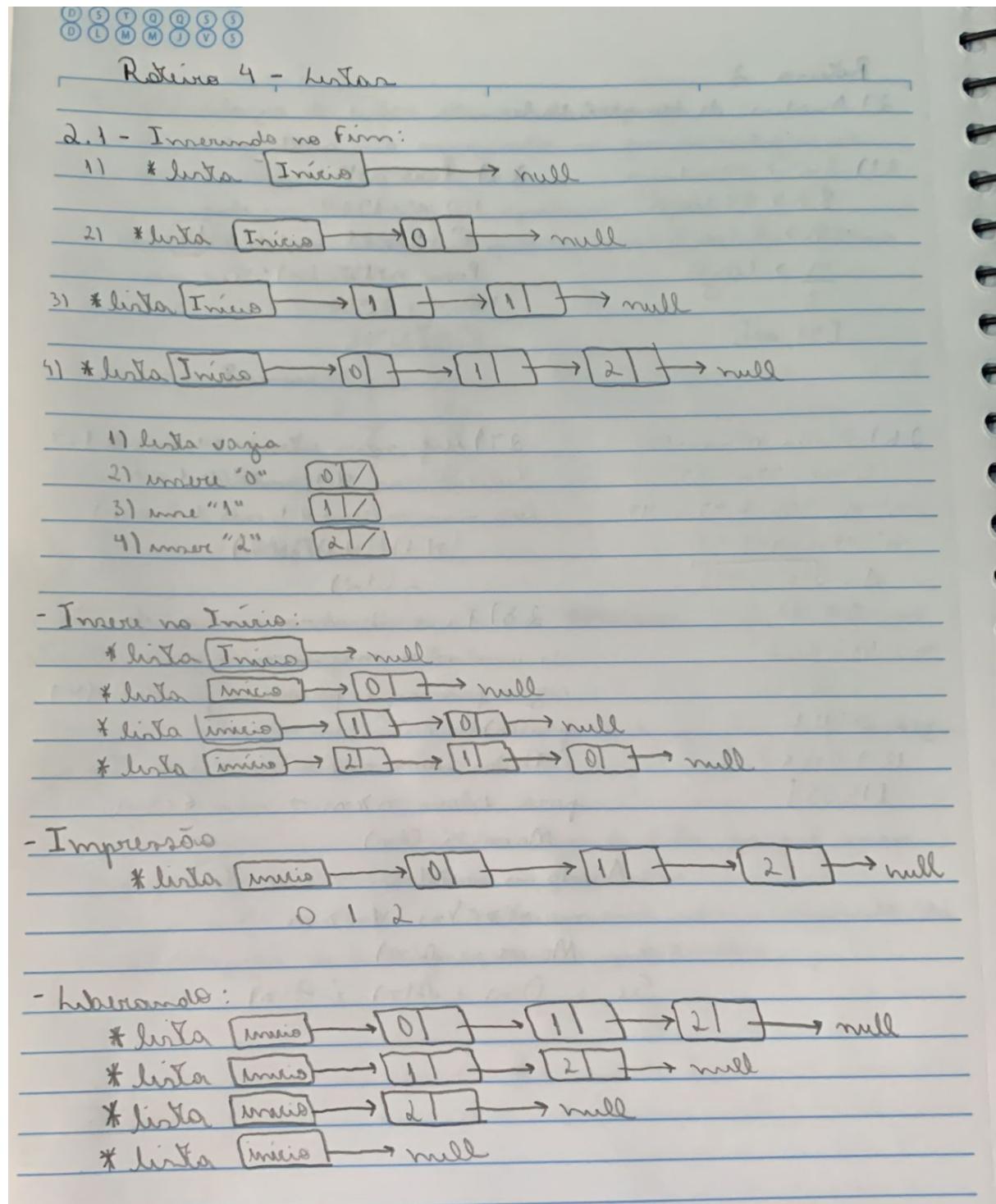
```

```

    }
}

#endif // LSEC_H

```



## 2.2 -

```

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

```

```

#include "lista_simplesmente_encadeada.h"

int tamanhoLista(Lista* lista) {
    int tamanho = 0;
    No* atual = *lista;
    while (atual != NULL) {
        tamanho++;
        atual = atual->prox;
    }
    return tamanho;
}

int procura(Lista* lista, int elemento) {
    No* atual = *lista;
    while (atual != NULL) {
        if (atual->info == elemento) {
            printf ("Elemento %d presente na lista.\n", elemento);
            return 0;
        }
        atual = atual->prox;
    }
    return -1;
}

int main() {
    Lista* lista = criarLista();
    insereFim(lista, 13);
    insereFim(lista, 29);
    insereFim(lista, 4);
    insereFim(lista, 68);
    insereFim(lista, 51);
    imprimeLista(lista);

    printf ("Tamanho da Lista: %d\n", tamanhoLista(lista));
    procura(lista, 29);
    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output'
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> & .\exercicio2-2.exe
Elementos:
13 29 4 68 51
Tamanho da Lista: 5
Elemento 29 presente na lista.

```

## 2.3 -

```
#include <stdio.h>
#include <stdlib.h>
#include "lista_simplesmente_encadeada.h"

int insereOrdenado(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;

    No* novo = alocarNo();
    novo->info = elemento;
    novo->prox = NULL;

    if (listaVazia(lista)) {
        *lista = novo;
    }
    else {
        No atual;
        atual.prox = *lista;

        No* auxiliar = &atual;
        while (auxiliar->prox != NULL && auxiliar->prox->info <
elemento)
            auxiliar = auxiliar->prox;

        if (auxiliar->prox == *lista) {
            novo->prox = *lista;
            *lista = novo;
        }

        novo->prox = auxiliar->prox;
        auxiliar->prox = novo;
    }

    return 1;
}

int main() {
    Lista* lista = criarLista();

    insereOrdenado(lista, 13);
    imprimeLista(lista);
```

```

    insereOrdenado(lista, 28);
    imprimeLista(lista);
    insereOrdenado(lista, 50);
    imprimeLista(lista);
    insereOrdenado(lista, 5);
    imprimeLista(lista);
    insereOrdenado(lista, 10);
    imprimeLista(lista);
    insereOrdenado(lista, 45);
    imprimeLista(lista);

    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output' & .\exercicio2-3.exe'
Elementos:
13
Elementos:
13 28
Elementos:
13 28 50
Elementos:
5 13 28 50
Elementos:
5 10 13 28 50
Elementos:
5 10 13 28 45 50

```

2.4 -

```

#include <stdio.h>
#include <stdlib.h>
#include "lista_simplesmente_encadeada.h"

int removePrimeiro(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;
    if (listavazia(lista))
        return 0;

    No* auxiliar = *lista;
    while (auxiliar->prox != NULL) {
        if (auxiliar->prox->info == elemento) {
            No* auxiliar2 = auxiliar->prox;
            auxiliar->prox = auxiliar2->prox;
            liberarNo(auxiliar2);
            break;
        }
    }
}

```

```

        auxiliar = auxiliar->prox;
    }

    return 1;
}

int main() {
    Lista *lista = criarLista();

    insereFim(lista, 13);
    insereFim(lista, 28);
    insereFim(lista, 65);
    insereFim(lista, 5);
    insereFim(lista, 14);
    insereFim(lista, 65);

    imprimeLista(lista);
    removePrimeiro(lista, 65);
    imprimeLista(lista);

    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output'
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> & .\exercicio2-4.exe'
Elementos:
13 28 65 5 14 65
Elementos:
13 28 5 14 65
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output>

```

3.1 -

```

#ifndef LDE_H
#define LDE_H

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

typedef struct NO{
    int info;
    struct NO* anterior;
    struct NO* proximo;
}NO;
typedef struct NO* Lista;

Lista* criarLista() {

```

```

Lista* lista;
lista = (Lista*)malloc(sizeof(Lista));
if (lista != NULL)
    *lista = NULL;
return lista;
}

int listaVazia(Lista* lista){
    if (lista == NULL)
        return 0;
    if (*lista == NULL)
        return 1;
    return 0;
}

NO* alocarNo() {
    return (NO*)malloc(sizeof(NO));
}

void liberarNo(NO* q) {
    free(q);
}

int insereInicio(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;
    NO* novo = alocarNo();
    if (novo == NULL)
        return 0;
    novo->info = elemento;
    novo->proxima = *lista;
    novo->anterior = NULL;
    if (!listaVazia(lista))
        (*lista)->anterior = novo;
    *lista = novo;
    return 1;
}

int insereFim(Lista *lista, int elemento) {
    if (lista == NULL)
        return 0;
    NO* novo = alocarNo();
    if (novo == NULL)

```

```

        return 0;
novo->info = elemento;
novo->proximo = NULL;
if (listaVazia(lista)) {
    novo->anterior = NULL;
    *lista = novo;
} else{
    NO* auxiliar = *lista;
    while (auxiliar->proximo != NULL)
        auxiliar = auxiliar->proximo;
    auxiliar->proximo = novo;
    novo->anterior = auxiliar;
}
return 1;
}

int removeInicio(Lista* lista) {
    if (lista == NULL)
        return 0;
    if (listaVazia(lista))
        return 0;
    NO* auxiliar = *lista;
    *lista = auxiliar->proximo;
    if (auxiliar->proximo != NULL)
        auxiliar->proximo->anterior = NULL;
    liberarNo(auxiliar);
    return 1;
}

int removeFim(Lista* lista) {
    if (lista == NULL)
        return 0;
    if (listaVazia(lista))
        return 0;
    NO* auxiliar = *lista;
    while (auxiliar->proximo != NULL)
        auxiliar = auxiliar->proximo;
    if (auxiliar->anterior == NULL)
        *lista = auxiliar->proximo;
    else
        auxiliar->anterior->proximo = NULL;
    liberarNo(auxiliar);
    return 1;
}

```

```
}
```

```
void imprimeLista(Lista* lista) {
    if (lista == NULL)
        return;
    if (listavazia(lista)) {
        printf ("Lista vazia!\n");
        return;
    }

    printf ("Elementos:\n");
    NO* auxiliar = *lista;
    while (auxiliar != NULL) {
        printf ("%d ", auxiliar->info);
        auxiliar = auxiliar->proximo;
    }
    printf("\n");
}

void destroiLista(Lista* lista) {
    if (lista != NULL){
        NO *auxiliar;
        while ((*lista) !=NULL) {
            auxiliar = *lista;
            *lista = (*lista)->proximo;
            liberarNo(auxiliar);
        }
        free(lista);
    }
}

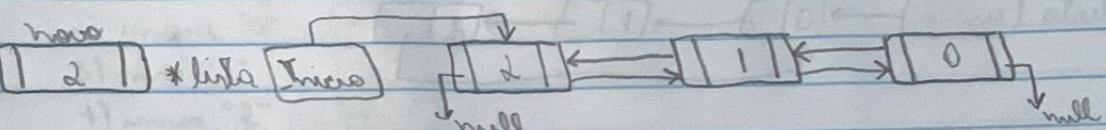
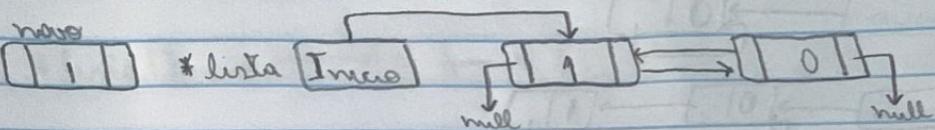
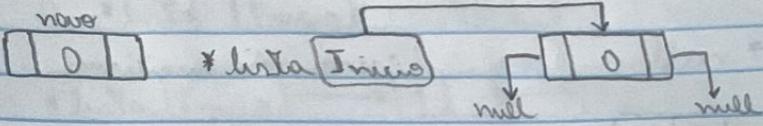
#endif //LDE_H
```

D S T Q O S S  
D L M M J V S

### Roteiro 4 - Lista Duplamente Encadeada

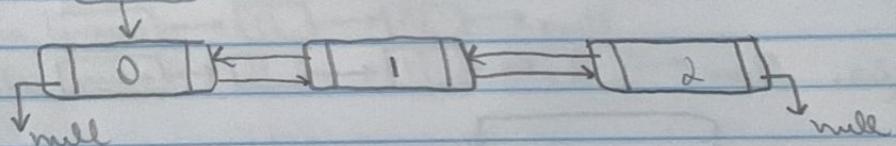
#### 3.1 Inserção no Início

\* lista [Início] → null

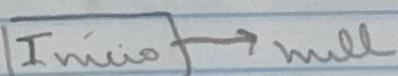
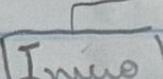
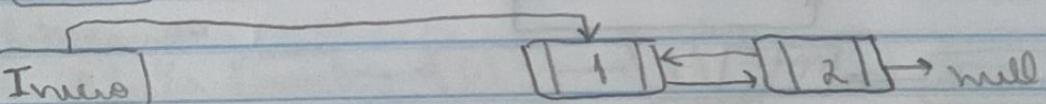
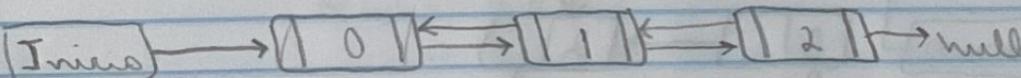


Impressão da lista

\* lista [Início]



Desfazendo a lista:



3.2 -

```

#include <stdio.h>
#include <stdlib.h>
#include "lista_duplamente_encadeada.h"

int procura(Lista* lista, int elemento){
    NO* atual = *lista;
    while (atual != NULL) {
        if (atual->info == elemento) {
            printf ("Elemento %d presente na lista.\n", elemento);
            return 0;
        }
        atual = atual->prox;
    }
    return -1;
}

int tamanhoLista(Lista* lista) {
    int tamanho = 0;
    NO* atual = *lista;
    while (atual != NULL) {
        tamanho++;
        atual = atual->prox;
    }
    return tamanho;
}

int main() {
    Lista* lista = criarLista();

    insereFim(lista, 13);
    insereFim(lista, 29);
    insereFim(lista, 4);
    insereFim(lista, 68);
    insereFim(lista, 51);
    imprimeLista(lista);

    printf ("Tamanho da Lista: %d\n", tamanhoLista(lista));
    procura(lista, 29);
    destroiLista(lista);
    return 0;
}

```

Saída:

```
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output'
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> & .\exercicio3-2.exe
Elementos:
13 29 4 68 51
Tamanho da Lista: 5
Elemento 29 presente na lista.
```

### 3.3 -

```
#include <stdio.h>
#include <stdlib.h>
#include "lista_duplamente_encadeada.h"

int insereOrdenado(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;

    NO* novo = alokarNo();
    novo->info = elemento;
    novo->anterior = NULL;
    novo->proximo = NULL;

    if (listaVazia(lista)) {
        *lista = novo;
    }
    else {
        NO atual;
        atual.proximo = *lista;

        NO* auxiliar = &atual;
        while (auxiliar->proximo != NULL && auxiliar->proximo->info < elemento)
            auxiliar = auxiliar->proximo;

        if (auxiliar->proximo == *lista) {
            novo->proximo = *lista;
            *lista = novo;
        } else{
            novo->anterior = auxiliar;
        }

        novo->proximo = auxiliar->proximo;
        auxiliar->proximo = novo;
    }

    return 1;
}
```

```

}

int main() {
    Lista* lista = criarLista();

    insereOrdenado(lista, 13);
    imprimeLista(lista);
    insereOrdenado(lista, 28);
    imprimeLista(lista);
    insereOrdenado(lista, 50);
    imprimeLista(lista);
    insereOrdenado(lista, 5);
    imprimeLista(lista);
    insereOrdenado(lista, 10);
    imprimeLista(lista);
    insereOrdenado(lista, 45);
    imprimeLista(lista);
    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output' & .\exercicio3-3.exe'
Elementos:
13
Elementos:
13 28
Elementos:
13 28 50
Elementos:
5 13 28 50
Elementos:
5 10 13 28 50
Elementos:
5 10 13 28 45 50
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> []

```

3.4 -

```

#include <stdio.h>
#include <stdlib.h>
#include "lista_duplamente_encadeada.h"

int removePrimeiro(Lista* lista, int elemento) {
    if (lista == NULL)
        return 0;
    if (listaVazia(lista))
        return 0;
    NO* auxiliar = *lista;
    while (auxiliar->proximo != NULL) {
        if (auxiliar->proximo->info == elemento) {

```

```

        NO* auxiliar2 = auxiliar->proximo;
        auxiliar->proximo = auxiliar2->proximo;
        auxiliar->proximo->anterior = auxiliar2->anterior;
        liberarNo(auxiliar2);
        break;
    }
    auxiliar = auxiliar->proximo;
}
return 1;
}

int main(){
    Lista *lista = criarLista();

    insereFim(lista, 13);
    insereFim(lista, 28);
    insereFim(lista, 65);
    insereFim(lista, 5);
    insereFim(lista, 14);
    insereFim(lista, 65);
    imprimeLista(lista);
    removePrimeiro(lista, 65);
    imprimeLista(lista);
    destroiLista(lista);
    return 0;
}

```

Saída:

```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output'
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> & .\exercicio3-4.exe'
Elementos:
13 28 65 5 14 65
Elementos:
13 28 5 14 65
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output>

```

4.1 -

```

#ifndef LCSE_H
#define LCSE_H

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

typedef struct NO{
    int info;
    struct NO* prox;
} NO;
typedef struct NO* Lista;

```

```

Lista* criarLista() {
    Lista *lista;
    lista = (Lista*) malloc (sizeof(Lista));
    if (lista != NULL)
        *lista = NULL;
    return lista;
}

int listaVazia(Lista* lista) {
    if (lista == NULL)
        return 1;
    if (*lista == NULL)
        return 1;
    return 0;
}

NO* alocarNo() {
    return (NO*)malloc(sizeof(NO));
}

void liberarNo(NO* q) {
    free(q);
}

int insereInicio(Lista* lista, int elem) {
    if(lista == NULL) return 0;
    NO* novo = alocarNo();
    if(novo == NULL) return 0;
    novo->info = elem;
    if(listaVazia(lista)){
        novo->prox = novo;
        *lista = novo;
    }else{
        NO* aux = *lista;
        while(aux->prox != (*lista))
            aux = aux->prox;
        aux->prox = novo;
        novo->prox = *lista;
        *lista = novo;
    }
    return 1;
}

```

```

int insereFim(Lista* lista, int elem) {
    if(lista == NULL) return 0;
    NO* novo = alocarNo();
    if(novo == NULL) return 0;
    novo->info = elem;
    if(listaVazia(lista)) {
        novo->prox = novo;
        *lista = novo;
    } else{
        NO* aux = *lista;
        while(aux->prox != (*lista))
            aux = aux->prox;
        aux->prox = novo;
        novo->prox = *lista;
    }
    return 1;
}

int removeInicio(Lista *lista) {
    if(lista == NULL) return 0;
    if(listaVazia(lista)) return 0;
    NO* prim = *lista;
    if(prim == prim->prox) {
        //apenas 1 elemento
        *lista = NULL;
    } else{
        NO* aux = *lista;
        while(aux->prox != (*lista))
            aux = aux->prox;
        aux->prox = (*lista)->prox;
        *lista = (*lista)->prox;
    }
    liberarNo(prim);
    return 1;
}

int removeFim(Lista *lista){
    if(lista == NULL) return 0;
    if(listaVazia(lista)) return 0;
    NO* aux = *lista;
    if(aux == aux->prox){
        //apenas 1 elemento

```

```

        *lista = NULL;
    }else{
        NO* ant;
        while(aux->prox != (*lista)) {
            ant = aux;//anterior
            aux = aux->prox;
        }
        ant->prox = *lista;
    }
    liberarNo(aux);
    return 1;
}

void imprimeLista(Lista* lista) {
    if(lista == NULL) return;
    if(listaVazia(lista)){
        printf("Lista Vazia!\n");
        return;
    }
    printf("Elementos:\n");
    NO* aux = *lista;
    while(aux->prox != *lista) {
        printf("%d ", aux->info);
        aux = aux->prox;
    }
    printf("%d ", aux->info);
    printf("\n");
}

void destroiLista(Lista *lista) {
    if(lista != NULL && (*lista) != NULL) {
        NO* prim, *aux;
        prim = *lista;
        *lista = (*lista)->prox;
        while((*lista) != prim) {
            aux = *lista;
            *lista = (*lista)->prox;
            printf("Destruindo.. %d\n", aux->info);
            liberarNo(aux);
        }
        printf("Destruindo.. %d\n", prim->info);
        liberarNo(prim);
        free(lista);
    }
}

```

```

    }
}

#endif //LCSE_H

```

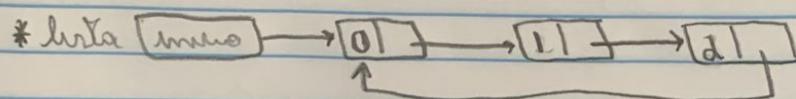
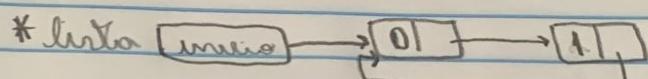
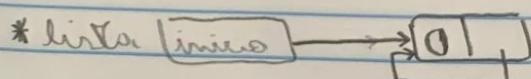


### Roteiro 4 - Listas

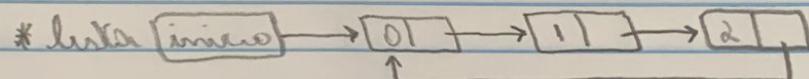
#### 4.1 - Lista Circular Simplemente Encadeada

##### - Inserção no Fim:

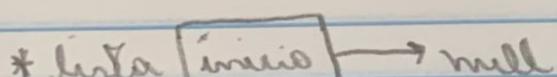
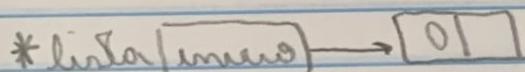
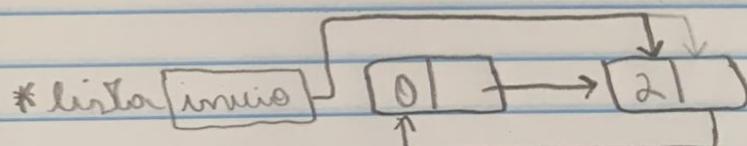
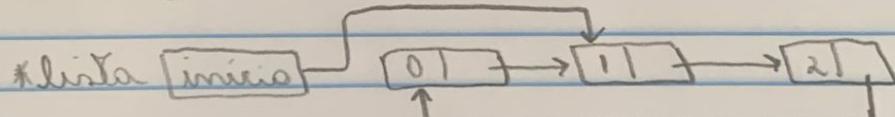
\* lista [início] → null      usar nó auxiliar.



##### - Impressão:



##### - Liberação:



```

#include <stdio.h>
#include <stdlib.h>
#include "lista_circular_simplesmente_encadeada.h"

int tamanho(Lista* lista) {
    int aux = 0;
    NO* aux2 = *lista;

    while (aux2 != *lista || aux == 0) {
        aux2 = aux2->prox;
        aux++;
    }
    return aux;
}

int procura(Lista* lista, int elemento) {
    int i = 0;
    NO* aux = *lista;

    while (aux != *lista || i == 0) {
        if (aux->info == elemento) return i;
        aux = aux->prox;
        i++;
    }
    return -1;
}

int main() {
    Lista* lista = criarLista();

    insereFim(lista, 13);
    insereFim(lista, 4);
    insereFim(lista, 28);
    insereFim(lista, 43);
    insereFim(lista, 66);
    imprimeLista(lista);
    printf ("Tamanho da Lista = %d\n", tamanho(lista));
    printf ("Indice de 43: %d\n", procura(lista, 43));
    destroiLista(lista);
    return 0;
}

```

Saída:

```
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output> cd 'c:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 4\output' & .\exercicio4-1.exe'
Elementos:
13 4 28 43 66
Tamanho da Lista = 5
Indice de 43: 3
Destruindo.. 4
Destruindo.. 28
Destruindo.. 43
Destruindo.. 66
Destruindo.. 13
```