ROTEIRO 4 – LUCAS EDUARDO LEITE COSTA

1 -

main.c > main()

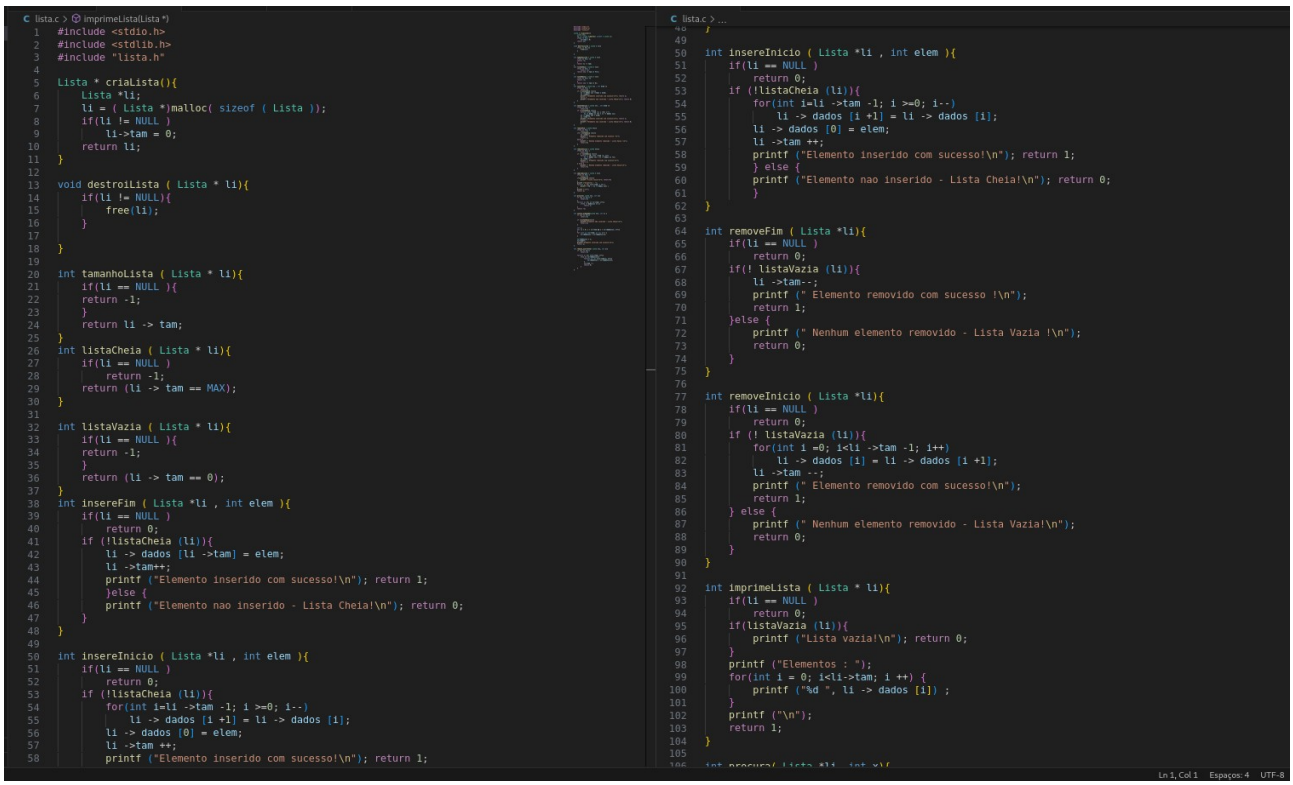
```c
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

int main(){
    Lista *l = criaLista();
    int elemento_x = 2, elemento_y = 3;
    insereInicio(l,elemento_x);
    insereFim(l,elemento_y);
    printf("Tamanho da lista : %d\n", tamanhoLista(l));
    if(-1 == procura(l,5))
        printf("Elemento não encontrado na lista\n");
    int x = 2;
    printf("A posição do elemento %d é %d\n", x , procura(l, x));
    insere_ordenada(l, 1);
    remove_ocorrente(l,3);
    imprimeLista(l);
    destroiLista(l);
    return 0;

}

```

lista.h > remove_ocorrente(Lista *, int)

```c
#ifndef LISTA_H
#define LISTA_H
#define MAX 100
typedef struct {
int dados [MAX];
int tam;
}Lista;
Lista* criaLista();
void destroiLista ( Lista * li);
int tamanhoLista ( Lista * li);
int listaCheia ( Lista * li);
int listaVazia ( Lista * li);
int insereFim ( Lista *li , int elem );
int insereInicio ( Lista *li , int elem );
int removeFim ( Lista *li);
int removeInicio ( Lista *li);
int imprimeLista ( Lista * li);
int procura( Lista *li, int x);
int insere_ordenada(Lista *li, int x);
int remove_ocorrente( Lista *li, int x);

#endif
```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPURAÇÃO

```
[lucascosta@fedora roteiro_4]$ make
gcc -c lista.c
gcc main.o lista.o -o main
[lucascosta@fedora roteiro_4]$ ./main
Elemento inserido com sucesso!
Elemento inserido com sucesso!
Tamanho da lista : 2
Elemento não encontrado na lista
A posição do elemento 2 é 0
Elemento inserido com sucesso!
Elementos : 1 2
[lucascosta@fedora roteiro_4]$
```



lista.c > imprimeLista(Lista *)

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

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

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

}

int tamanhoLista ( Lista * li){
    if(li == NULL ){
    return -1;
    }
    return li -> tam;
}
int listaCheia ( Lista * li){
    if(li == NULL )
        return -1;
    return (li -> tam == MAX);
}

int listaVazia ( Lista * li){
    if(li == NULL ){
    return -1;
    }
    return (li -> tam == 0);
}
int insereFim ( Lista *li , int elem ){
    if(li == NULL )
        return 0;
    if (!listaCheia (li)){
        li -> dados [li ->tam] = elem;
        li ->tam++;
        printf ("Elemento inserido com sucesso!\n"); return 1;
        }else {
        printf ("Elemento nao inserido - Lista Cheia!\n"); return 0;
    }
}

int insereInicio ( Lista *li , int elem ){
    if(li == NULL )
        return 0;
    if (!listaCheia (li)){
        for(int i=li ->tam -1; i >=0; i--)
            li -> dados [i +1] = li -> dados [i];
        li -> dados [0] = elem;
        li ->tam ++;
        printf ("Elemento inserido com sucesso!\n"); return 1;
```

lista.c > ...

```c
}

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

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

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

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

int procura( Lista *li, int x){
```

Ln 1, Col 1 Espaços: 4 UTF-8

```c
}

int procura( Lista *li, int x){
    if (li == NULL) {
        return 0;
    }
    for(int i = 0; i< li->tam; i++){
        if(li -> dados[i] == x)
            return i;
    }
    return -1;
}

int insere_ordenada(Lista *li, int x) {
    if (li == NULL)
        return 0;

    if (listaCheia(li)){
        printf("Elemento não inserido - Lista Cheia!\n");
        return 0;
    }

    int i;
    for (i = 0; i < li->tam && x > li->dados[i]; i++){
    }
    for (int j = li->tam; j > i; j--) {
        li->dados[j] = li->dados[j-1];
    }

    li->dados[i] = x;
    li->tam++;
    printf("Elemento inserido com sucesso!\n");
    return 1;
}

int remove_ocorrente( Lista *li, int x){
    if (li == NULL)
        return 0;

    for(int i = 0; i<li->tam; i++){
        if(x == li->dados[i]){
            for(int j = i; j<li->tam-1; j++){
                li->dados[j] = li->dados[j+1];
            }
            li->tam --;
            return 0;
        }
    }
}
```

2

2 > C main.c > main()

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

int main(){
    Lista li = criaLista();
    insere_ordenado(li,5);
    insere_ordenado(li,7);
    insere_ordenado(li,2);
    insere_ordenado(li,8);
    remove_ocorrente(li,8);
    if(!procura(li,15))
        printf("Elemento não pertence a lista\n");
    if(procura(li,5))
        printf("Elemento encontrado!\n");
    imprimeLista(li);
    destroiLista(li);
    return 0;

}
```

2 > C lista.h > insere_ordenado(Lista, int)

```c
#ifndef LISTA_H
#define LISTA_H

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

typedef struct NO* Lista;
NO* alocarNO();
int procura(Lista li, int x);
int tamanho(Lista li);
Lista criaLista();
int listaVazia(Lista li);
void imprimeLista(Lista li);
void remove_ocorrente(Lista lista, int x);
void insere_ordenado(Lista lista, int x);
void destroiLista(Lista * li);
void liberarNO (NO* atual);
#endif
```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPURAÇÃO

```
[lucascosta@fedora 2]$ ./main
Elemento não pertence a lista
Elemento encontrado!
Elementos: 2 5 7
[lucascosta@fedora 2]$
```

.

2 > C lista.c > ...

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

Lista criaLista() {
    Lista li = (Lista)malloc(sizeof(struct NO));
    if (li != NULL)
        li->prox = NULL;
    return li;
}

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

void insere_ordenado(Lista lista, int n) {
    if (lista == NULL)
        return;

    NO* novo = alocarNO();
    if (novo == NULL)
        return;

    novo->info = n;

    NO* atual = lista->prox;
    NO* anterior = NULL;

    while (atual != NULL && n > atual->info) {
        anterior = atual;
        atual = atual->prox;
    }

    if (anterior == NULL) {
        novo->prox = lista->prox;
        lista->prox = novo;
    } else {
        novo->prox = anterior->prox;
        anterior->prox = novo;
    }
}

void liberarNO (NO* atual){
    free(atual);
}

void remove_ocorrente(Lista lista, int n) {
    if (lista == NULL || lista->prox == NULL)
```

2 > C lista.c > ...

```c

void remove_ocorrente(Lista lista, int n) {
    if (lista == NULL || lista->prox == NULL)
        return;

    NO* atual = lista->prox;
    NO* anterior = NULL;

    while (atual != NULL) {
        if (atual->info == n){

            if (anterior == NULL) {

                lista->prox = atual->prox;
            } else {

                anterior->prox = atual->prox;
            }
            free(atual);
            return;
        }
        anterior = atual;
        atual = atual->prox;
    }
}

int listaVazia(Lista li) {
    if (li == NULL || li->prox == NULL)
        return 1;
    return 0;
}

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

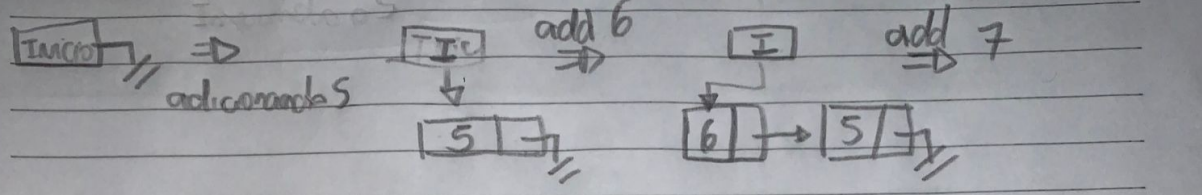
•

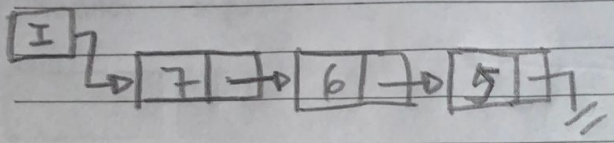
2 > C lista.c > ...

```c
    printf("\n");
}


int tamanho(Lista li) {
    int tam = 0;
    if (li == NULL) {
        return tam;
    }
    NO* aux = li->prox;
    while (aux != NULL) {
        tam++;
        aux = aux->prox;
    }
    return tam;
}

int procura(Lista li, int x) {
    if (li == NULL)
        return 0;
    NO* aux = li->prox;
    while (aux != NULL) {
        if (aux->info == x) {
            return 1;
        }
        aux = aux->prox;
    }
    return 0;
}

void destroiLista( Lista * li){
    if(li != NULL ){
        NO* aux;
        while ((* li) != NULL ){
            aux = *li;
            *li = (* li) ->prox ;
            liberarNO (aux);
        }
        free(li);
    }
}
```

Inserção de 3 elementos no início
Inserindo 5, 6 e 7 numa lista vazia

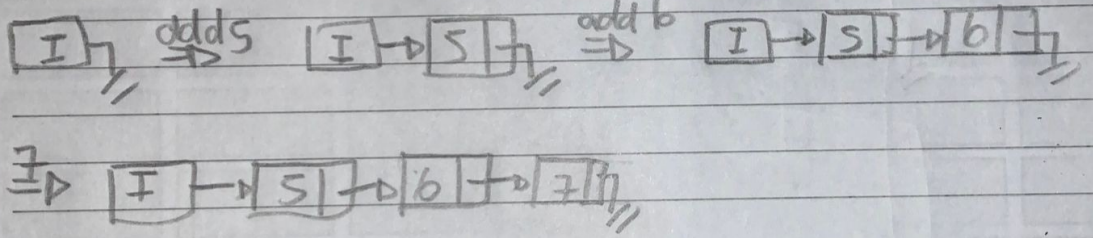
Notação = [I] = Início
Elemento = [Info | Prox]



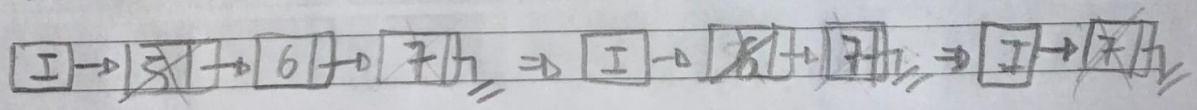


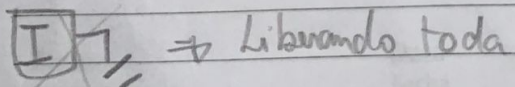
As mesmas inserções no fim

Percorre a lista e insere no final

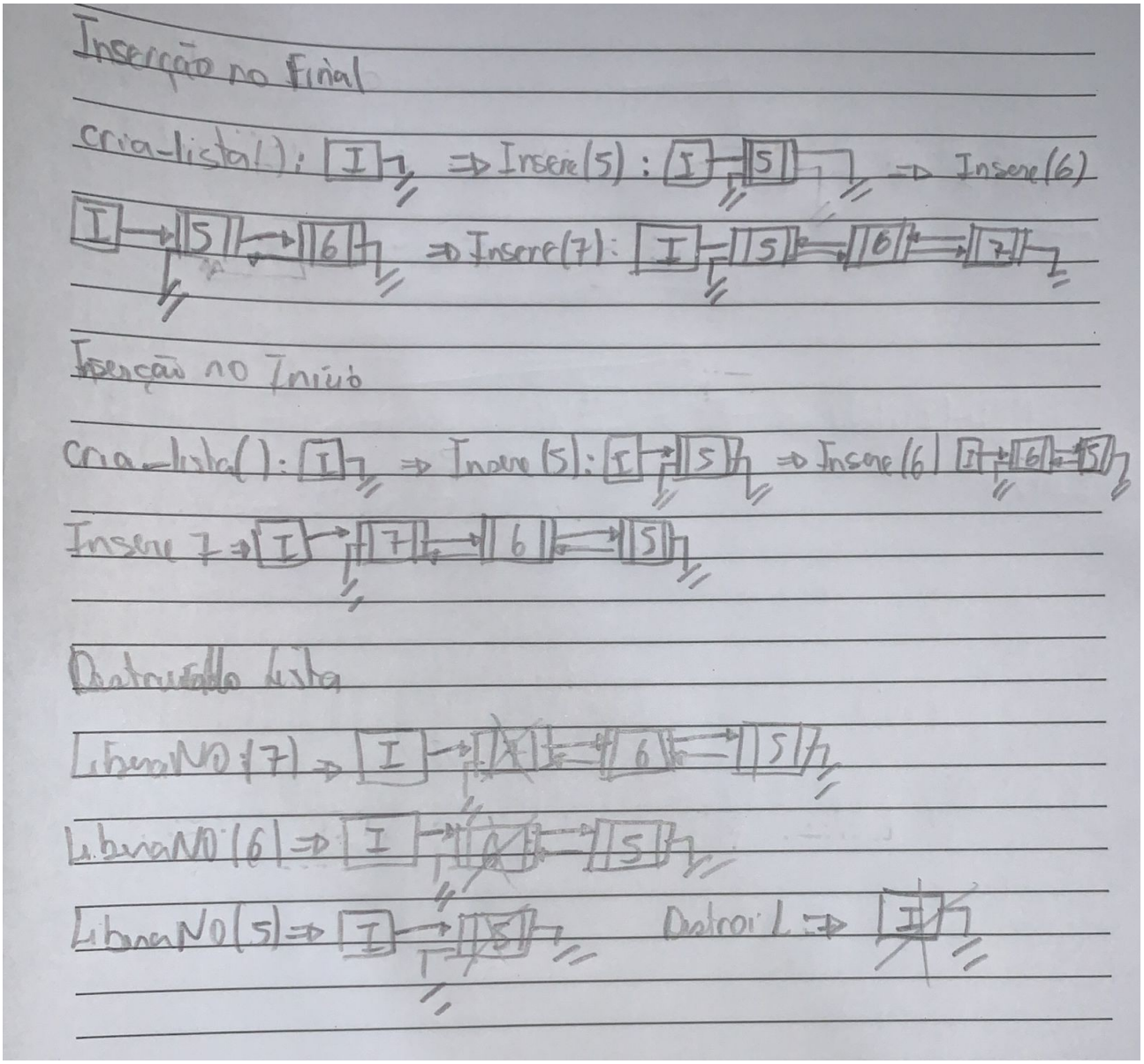


Destruindo - uma a uma, liberando cada nó





3


Inserção no Final
cria-lista(): I
Insere(5): I 5
Insere(6)
I 5 6
Insere(7): I 5 6 7
Inserção no Início
cria-lista(): I
Insere(5): I 5
Insere(6) I 6 5
Insere 7 I 7 6 5
Destruindo Lista
LiberaNO(7) I X 6 5
LiberaNO(6) I X 5
LiberaNO(5) I X
Destroi L I

3 > C main.c > main()

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

int main(){
    Lista *li = criaLista();
    insere_ordenado(li,6);
    insere_ordenado(li,1);
    insere_ordenado(li,5);
    insere_ordenado(li,9);
    insere_ordenado(li,7);
    imprimeLista(li);
    remove_ocorrente(li,5);
    imprimeLista(li);
    printf("Tamanho da lista: %d\n", tamanhoLista(li));
    if(procura(li,9))
        printf("O elemento está na lista!\n");
    destroiLista(li);
    return 0;
}
```

3 > C lista.h > ...

```c
# ifndef LISTA_H
# define LISTA_H

typedef struct NO{
    int info ;
    struct NO* ant;
    struct NO* prox ;
}NO;
typedef struct NO* Lista ;
#endif


Lista *criaLista();
int listaVazia(Lista * li);
NO* alocarNO();
void liberarNO(NO* q);
int insereIni(Lista * li , int elem );
int insereFim(Lista * li , int elem );
int removeIni(Lista * li);
int removeFim(Lista * li);
void imprimeLista(Lista * li);
void destroiLista(Lista * li);
int tamanhoLista (Lista * li);
int procura(Lista *li, int elem);
int insere_ordenado(Lista * li , int elem );
int remove_ocorrente(Lista * li, int elem);

```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPURAÇÃO

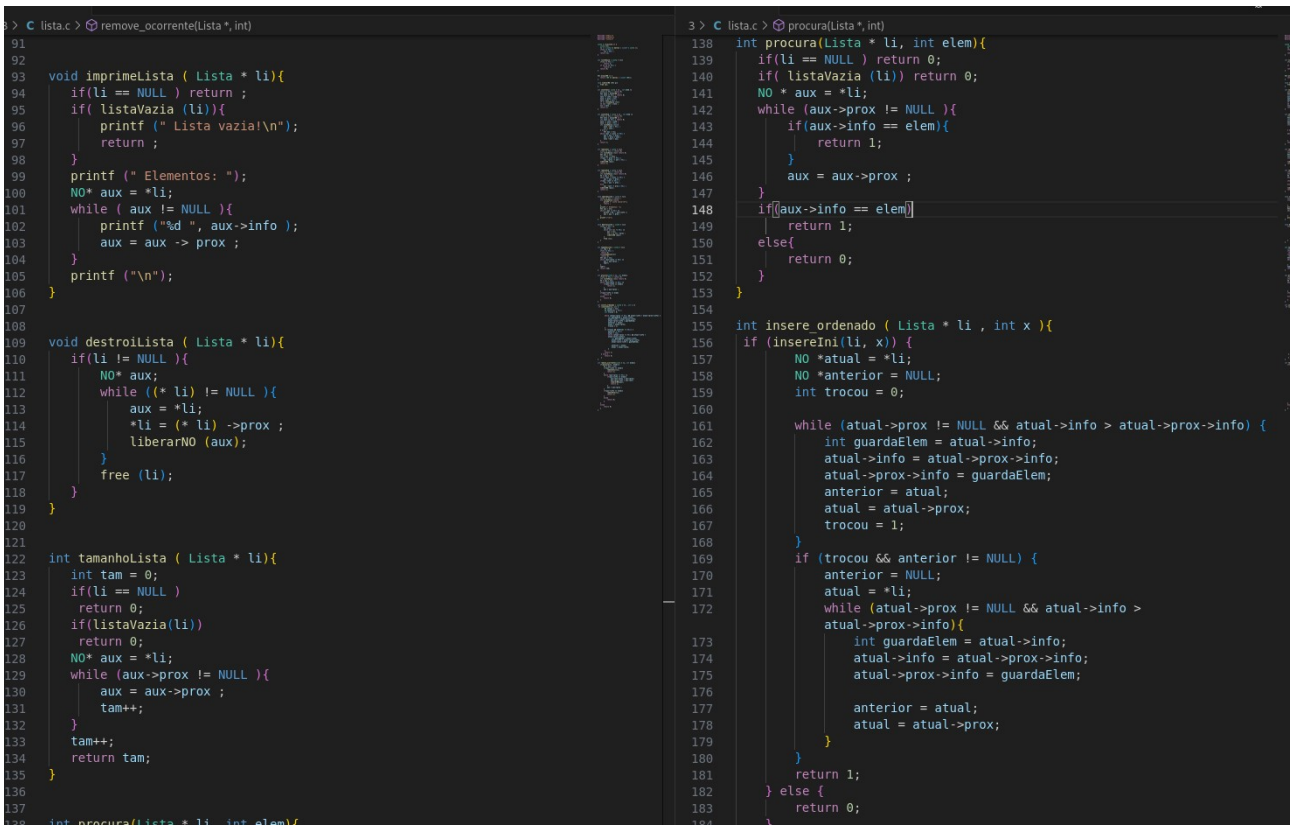
```
[lucascosta@fedora 3]$ ./main
 Elementos: 1 5 6 7 9
 Elementos: 1 6 7 9
Tamanho da lista: 4
O elemento está na lista!
[lucascosta@fedora 3]$
```

3 > C lista.c > remove_ocorrente(Lista *, int)

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


Lista * criaLista () {
    Lista *li;
    li = ( Lista *) malloc ( sizeof ( Lista ));
    if(li != NULL )
        *li = NULL ;
    return li;
}


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


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

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

int insereIni( Lista * li , int elem ){
    if(li == NULL ) return 0;
    NO* novo = alocarNO () ;
    if( novo == NULL ) return 0;
    novo -> info = elem ;
    novo -> prox = *li;
    novo ->ant = NULL ;
    if (! listaVazia (li))
    (* li) ->ant = novo ;
    *li = novo ;
    return 1;
}


int insereFim ( Lista * li , int elem ){
    if(li == NULL ) return 0;
    NO* novo = alocarNO () ;
```

3 > C lista.c > remove_ocorrente(Lista *, int)

```c

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


int removeIni ( Lista * li){
    if(li == NULL ) return 0;
    if( listaVazia (li)) return 0;
    NO* aux = *li;
    *li = aux -> prox ;
    if(aux -> prox != NULL )
        aux ->prox -> ant = NULL ;
    liberarNO (aux);
    return 1;
}


int removeFim ( Lista * li){
    if(li == NULL ) return 0;
    if( listaVazia (li)) return 0;
    NO * aux = *li;
    while (aux -> prox != NULL )
        aux = aux -> prox ;
    if(aux ->ant == NULL )
        *li = aux -> prox ;
    else
        aux ->ant -> prox = NULL ;
    liberarNO (aux);
    return 1;
}

```



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


void destroiLista ( Lista * li){
   if(li != NULL ){
       NO* aux;
       while ((* li) != NULL ){
           aux = *li;
           *li = (* li) ->prox ;
           liberarNO (aux);
       }
       free (li);
   }
}


int tamanhoLista ( Lista * li){
   int tam = 0;
   if(li == NULL )
    return 0;
   if(listaVazia(li))
    return 0;
   NO* aux = *li;
   while (aux->prox != NULL ){
       aux = aux->prox ;
       tam++;
   }
   tam++;
   return tam;
}
```

```c
int procura(Lista * li, int elem){
   if(li == NULL ) return 0;
   if( listaVazia (li)) return 0;
   NO * aux = *li;
   while (aux->prox != NULL ){
       if(aux->info == elem){
           return 1;
       }
       aux = aux->prox ;
   }
   if(aux->info == elem)
       return 1;
   else{
       return 0;
   }
}

int insere_ordenado ( Lista * li , int x ){
 if (insereIni(li, x)) {
       NO *atual = *li;
       NO *anterior = NULL;
       int trocou = 0;

       while (atual->prox != NULL && atual->info > atual->prox->info) {
           int guardaElem = atual->info;
           atual->info = atual->prox->info;
           atual->prox->info = guardaElem;
           anterior = atual;
           atual = atual->prox;
           trocou = 1;
       }
       if (trocou && anterior != NULL) {
           anterior = NULL;
           atual = *li;
           while (atual->prox != NULL && atual->info >
           atual->prox->info){
               int guardaElem = atual->info;
               atual->info = atual->prox->info;
               atual->prox->info = guardaElem;

               anterior = atual;
               atual = atual->prox;
           }
       }
       return 1;
   } else {
       return 0;
   }
```

```c
int remove_ocorrente(Lista * li, int elem){
   if(procura(li, elem)){
       NO * aux = *li;
       if(aux->info == elem){
           removeIni(li);
           return 1;
       }
       while (aux->prox != NULL ){
           if(aux->info == elem){
               aux->ant->prox = aux->prox;
               aux->prox->ant = aux->ant;
               liberarNO(aux);
               return 1;
           }
           aux = aux->prox ;
       }
       if(aux->info == elem){
           removeFim(li);
           return 1;
       }
       else{
           return 0;
       }
   }
   else{
       return 0;
   }
}
```

.4

4 > C main.c > main()

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

int main(){
    Lista* l = criaLista();
    insereFim(l,7);
    insereFim(l,9);
    insereFim(l,1);
    insereFim(l,8);
    if(!procura(l,25))
        printf("Elemento não encontrado!\n");
    if(procura(l,8))
        printf("Elemento pertence a lista\n");
    printf("Tamanho da lista: %d\n", tamanho(l));
    imprimeLista(l);
    destroiLista(l);
}
```

4 > C lista.h > LISTA

```c
#ifndef LISTA_H
#define LISTA_H
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

typedef struct NO {
    int info;
    struct NO* prox;
} NO;
typedef struct NO* Lista;
Lista* criaLista();
int tamanho(Lista* lista);
int procura(Lista* lista, int n);
void imprimeLista(Lista* li);
NO* alocarNO(int n);
int insereFim(Lista* li, int n);
void destroiLista(Lista *li);
void liberarNO(NO* no);
#endif
```

PROBLEMAS SAÍDA TERMINAL PORTAS CONSOLE DE DEPURAÇÃO

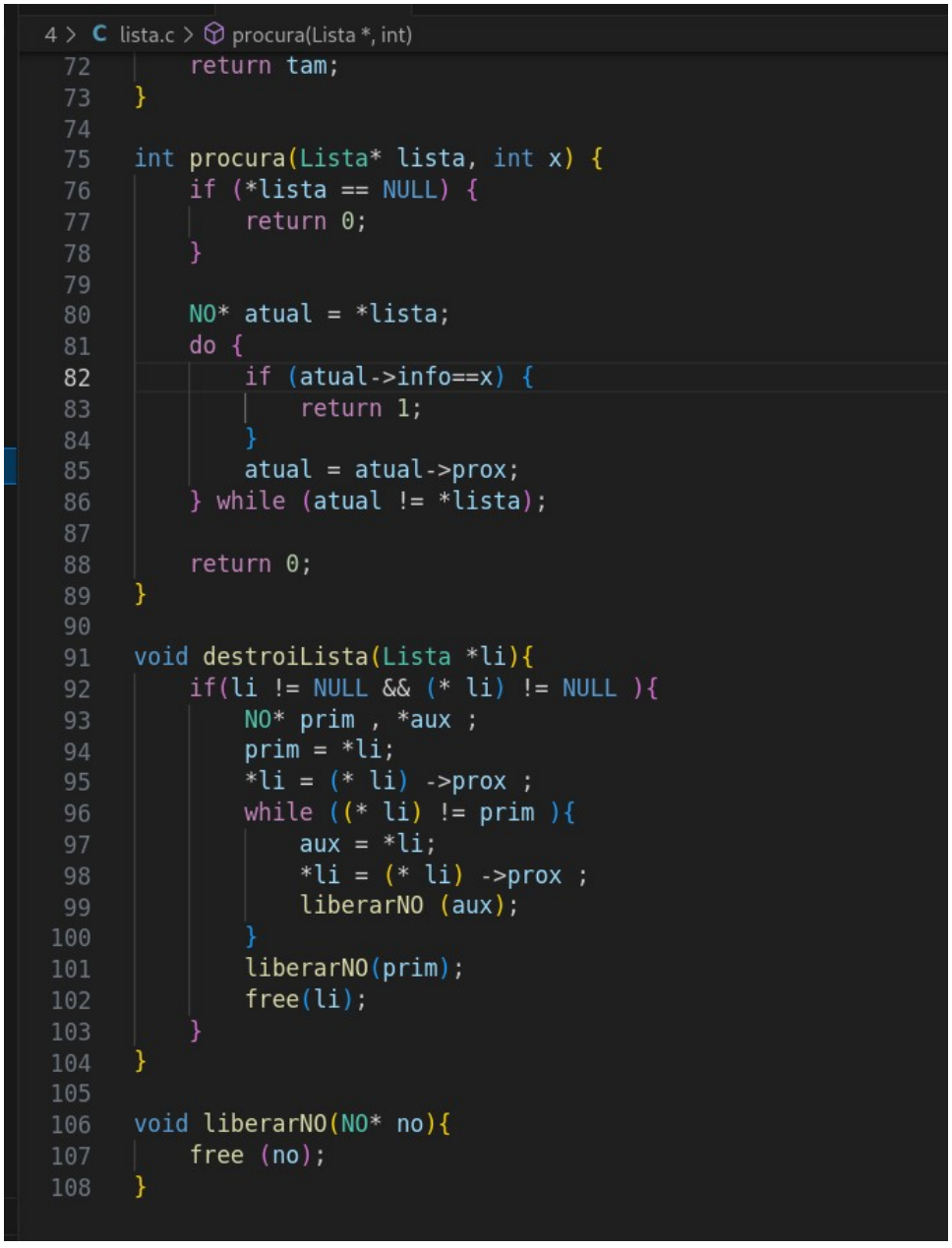
```
[lucascosta@fedora 4]$ ./main
Elemento não encontrado!
Elemento pertence a lista
Tamanho da lista: 4
Elementos na lista: 7 9 1 8
[lucascosta@fedora 4]$
```

4 > C lista.c > procura(Lista *, int)

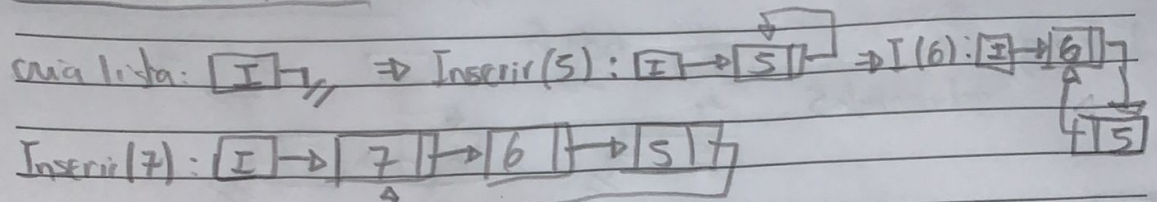
```c
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

Lista* criaLista () {
    Lista *li;
    li = (Lista *) malloc(sizeof(Lista ));
    if(li != NULL )
        *li = NULL ;
    return li;
}

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

NO* alocarNO(int n) {
    NO* novo = (NO*)malloc(sizeof(NO));
    if (novo == NULL) {
        return NULL;
    }

    novo->info = n;
    novo->prox = NULL;
    return novo;
}
```

4 > C lista.c > ...

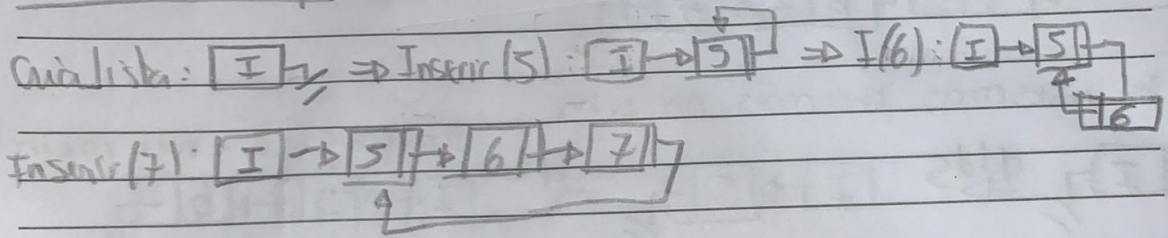
```c

int insereFim(Lista* li, int n) {
    NO* novo = alocarNO(n);
    if (novo == NULL) {
        return 0;
    }

    if (*li == NULL) {
        *li = novo;
        novo->prox = novo;
    } else {
        NO* aux = *li;
        while (aux->prox != *li) {
            aux = aux->prox;
        }
        aux->prox = novo;
        novo->prox = *li;
    }

    return 1;
}

int tamanho(Lista* lista) {
    if (*lista == NULL) {
        return 0;
    }
    int tam = 1;
    NO* atual = *lista;
    atual = atual->prox;
    while (atual != *lista){
        tam++;
        atual = atual->prox;
    }

    return tam;
}
```


4 > C lista.c > procura(Lista *, int)
72 return tam;
73 }
74
75 int procura(Lista* lista, int x) {
76 if (*lista == NULL) {
77 return 0;
78 }
79
80 NO* atual = *lista;
81 do {
82 if (atual->info==x) {
83 return 1;
84 }
85 atual = atual->prox;
86 } while (atual != *lista);
87
88 return 0;
89 }
90
91 void destroiLista(Lista *li){
92 if(li != NULL && (* li) != NULL){
93 NO* prim , *aux ;
94 prim = *li;
95 *li = (* li) ->prox ;
96 while ((* li) != prim){
97 aux = *li;
98 *li = (* li) ->prox ;
99 liberarNO (aux);
100 }
101 liberarNO(prim);
102 free(li);
103 }
104 }
105
106 void liberarNO(NO* no){
107 free (no);
108 }

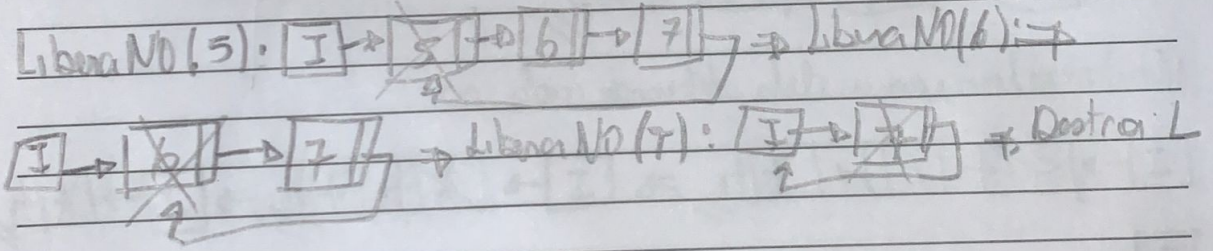
Inserindo no início



Inserindo no final



Destruindo lista



Lista destruída.