**ESTATICA SEQUENCIAL NAO ORDENADA**

\*.c

#include "estseqnord.h"

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

struct lista {

int no[MAX];

int Fim;

};

lista\* lista\_cria(){

lista \*L;

L = (lista\*)malloc(sizeof(lista));

if(L != NULL) L->Fim = 0;

return L;

}

int lista\_vazia(lista \*L){

if(L->Fim == 0){

return 1;

}

return 0;

}

int lista\_cheia(lista \*L){

if(L->Fim == MAX){

return 1;

}

return 0;

}

int insere\_elem(lista \*L,int elem) {

if (L == NULL || lista\_cheia(L))return 0;

L->no[L->Fim] = elem;

L->Fim++;

return 1;

}

int remove\_elem (lista \*L, int elem) {

if (L == NULL || lista\_vazia(L) == 1) return 0;

int i, Aux = 0;

while (Aux < L->Fim && L->no[Aux] != elem) Aux++;

for (i = Aux+1; i < L->Fim; i++) L->no[i-1] = L->no[i];

L->Fim--;

return 1;

}

int remove\_todos (lista \*L, int elem) {

if (L == NULL || lista\_vazia(L) == 1) return 0;

int i=0,j;

while(i<L->Fim){

if(L->no[i] == elem){

if(i == L->Fim-1){

L->Fim--;

return 1;

}

else{

for(j=i;j<L->Fim-1;j++){

L->no[j] = L->no[j+1];

}

L->Fim--;

}

}

else i++;

}

return 0;

}

void lista\_imprime(lista \*L){

int i;

printf("{ ");

if (! lista\_vazia(L) ){

for (i = 0; i < L->Fim ; i++) printf("%d ",L->no[i]);

}

printf("}");

printf("\n\n");

}

int insere\_inicio(lista \*L, int elem)

{

int i;

if (L == NULL || lista\_cheia(L))return 0;

for(i=L->Fim+1; i>0; i--){

L->no[i] = L->no[i-1];

}

L->no[0] = elem;

L->Fim++;

return 1;

}

int remove\_impares(lista \*L)

{

int i;

if (L == NULL || lista\_vazia(L) == 1) return 0;

for(i=0; i<L->Fim;i++){

if(L->no[i] % 2 != 0){

remove\_elem(L, L->no[i]);

}

}

return 1;

}

int menor\_elem(lista \*L)

{

if (L == NULL || lista\_vazia(L) == 1) return -10000;

int menor = L->no[0];

int i;

for(i=0;i<L->Fim;i++){

if(L->no[i]<menor){

menor = L->no[i];

}

}

return menor;

}

int tamanho(lista \*L)

{

int tam;

if(L == NULL || lista\_vazia(L) == 1) return 0;

tam = L->Fim;

return tam;

}

lista\* concatenar(lista \*A,lista \*B)

{

lista \*C = lista\_cria();

if(C==NULL) return NULL;

int i;

if(A->Fim < MAX){

for(i=0;i<A->Fim;i++){

insere\_elem(C,A->no[i]);

}

if(B->Fim < MAX-A->Fim){

for(i=0;i<B->Fim;i++){

insere\_elem(C,B->no[i]);

}

}

else return NULL;

return C;

}

return NULL;

}

\*.h

typedef struct lista lista;

lista\* lista\_cria();

int lista\_vazia(lista \*L);

int lista\_cheia(lista \*L);

int insere\_elem(lista \*L,int elem);

int remove\_elem (lista \*L, int elem);

int remove\_todos (lista \*L, int elem);

void lista\_imprime(lista \*L);

int insere\_inicio(lista \*L, int elem);

int remove\_impares(lista \*L);

int menor\_elem(lista \*L);

int tamanho(lista \*L);

lista\* concatenar(lista \*A,lista \*B);

**ESTATICA SEQUENCIAL ORDENADA**

\*.c

#include "estseqord.h"

#define max 10

struct lista {

int no[max];

int Fim;

};

lista\* lista\_cria(){

lista \*L;

L = (lista\*)malloc(sizeof(lista));

if(L != NULL) L->Fim = 0;

return L;

}

int lista\_vazia(lista \*L){

if(L->Fim == 0){

return 1;

}

return 0;

}

int lista\_cheia(lista \*L){

if(L->Fim == max){

return 1;

}

return 0;

}

void lista\_imprime(lista \*L){

int i;

printf("{ ");

if (! lista\_vazia(L) ){

for (i = 0; i < L->Fim ; i++) printf("%d ",L->no[i]);

}

printf("}\n\n");

}

int insere\_ord(lista \*L, int elem){

int i;

if(L == NULL || lista\_cheia(L)) return 0;

for(i=0;i<L->Fim;i++){

if(elem < L->no[i]) break;

}

if(i > L->Fim-1){

L->no[i] = elem;

L->Fim++;

return 1;

}

else{

int aux = 0;

while (elem >= L->no[aux]) aux++;

for (i = L->Fim; i > aux; i--) L->no[i] = L->no[i - 1];

L->no[aux] = elem;

}

L->Fim++;

return 1;

}

int remove\_ord(lista \*L, int elem){

if (L == NULL || lista\_vazia(L) == 1 ) return 0;

int i,j;

for (i = 0; i < L->Fim; i++)

{

if (L->no[i] == elem)

{

if (i == L->Fim)

{

L->Fim--;

return 1;

}

else

{

for (j = i; j < L->Fim; j++)

{

L->no[j] = L->no[j + 1];

}

L->Fim --;

return 1;

}

}

}

return 0;

return 1;

}

lista\* intercala\_ord(lista \*L1, lista \*L2)

{

lista \*C;

C = lista\_cria();

int i;

for (i = 0; i < L1->Fim; i++)

{

insere\_ord(C, L1->no[i]);

}

for (i = 0; i < L2->Fim; i++)

{

insere\_ord(C, L2->no[i]);

}

return C;

}

int remove\_pares(lista \*L)

{

if(L==NULL||lista\_vazia(L)) return 0;

int i=0,j,controle=L->Fim;

while(i<L->Fim)

{

if(L->no[i]%2==0){

if(i==L->Fim-1){

L->Fim--;

return 1;

}

else{

for(j=i;j<L->Fim-1;j++){

L->no[j] = L->no[j+1];

}

L->Fim--;

}

}

else i++;

}

if(L->Fim<controle) return 1;

return 0;

}

int maior(lista \*L)

{

if(L == NULL || lista\_vazia(L) == 1) return -10000;

return L->no[L->Fim-1];

}

int tamanho(lista \*L)

{

return L->Fim;

}

int iguais(lista \*L1, lista \*L2)

{

if(L1->Fim != L2->Fim) return 0;

int flag=0,i;

for(i=0;i<L1->Fim;i++){

if(L1->no[i] != L2->no[i])flag++;

}

if(flag != 0)return 0;

return 1;

}

\*.h

#include <stdio.h>

#include <stdlib.h>

typedef struct lista lista;

lista\* lista\_cria();

int lista\_vazia(lista \*L);

int lista\_cheia(lista \*L);

int lista\_insere(lista \*L,int elem);

int remove\_elem (lista \*L, int elem);

int remove\_todos (lista \*L, int elem);

void lista\_imprime(lista \*L);

int insere\_ord(lista \*L, int elem);

int remove\_ord(lista \*L, int elem);

lista \*intercala\_ord(lista \*L1, lista \*L2);

int remove\_pares(lista \*L);

int maior(lista \*L);

int tamanho(lista \*L);

int iguais(lista \*L1, lista \*L2);

**DINAMICA ENCADEADA NAO ORDENADA (SEM CABEÇALHO)**

\*.c

#include "dinencnord.h"

struct encadeada

{

int elemento;

struct encadeada \*prox;

};

Lista \*cria\_lista()

{

return 0;//NULL

}

int lista\_vazia(Lista \*l)

{

if(l->prox==NULL) return 1;

return 0;

}

int insere\_elem(Lista \*\*l, int n)

{

Lista \*temp;

temp = (Lista \*) malloc(sizeof(Lista));

if(temp==NULL) return 0;

temp->elemento = n;

temp->prox = \*l;

\*l=temp;

return 1;

}

int remove\_elem(Lista \*\*l,int n)

{

if(lista\_vazia(\*l)) return 0;

Lista \*temp = \*l;

if(n==(\*l)->elemento){

\*l = temp->prox;

free(temp);

return 1;

}

while(temp->prox!=NULL && temp->prox->elemento!=n) temp = temp->prox;

if(temp->prox==NULL) return 0;

Lista \*temp\_aux = temp->prox;

temp->prox = temp\_aux->prox;

free(temp\_aux);

return 1;

}

void print\_lista(Lista \*l)

{

Lista \* temp;

temp = l;

printf("Lista = {");

while(temp!=NULL){

if(temp->prox!=NULL)printf("%d ",temp->elemento);

else printf("%d",temp->elemento);

temp = temp->prox;

}

printf("}\n");

}

int remove\_todos(Lista \*\*l, int elem)

{

int cont=0;

Lista \*temp = \*l;

if(lista\_vazia(temp)) return 0;

while(temp!=NULL){

if(temp->elemento==elem){

remove\_elem(l,temp->elemento);

cont++;

}

temp = temp->prox;

}

return cont;

}

int insere\_fim(Lista \*l, int elem)

{

Lista \*temp;

temp = (Lista \*) malloc(sizeof(Lista));

if(temp==NULL) return 0;

temp->elemento = elem;

if(l==NULL){

temp->prox = l;

l = temp;

return 1;

}

Lista \*aux;

aux = l;

while(aux->prox!=NULL){

aux = aux->prox;

}

temp->prox = aux->prox;

aux->prox = temp;

return 1;

}

int remove\_pares(Lista \*\*l)

{

int cont=0;

Lista \*temp = \*l;

if(lista\_vazia(temp)) return 0;

while(temp!=NULL){

if(temp->elemento%2==0){

remove\_elem(l,temp->elemento);

cont++;

}

temp = temp->prox;

}

return cont;

}

int tamanho(Lista \*l)

{

int cont = 0;

Lista \*no = l;

while (no != NULL)

{

no = (no)->prox;

cont++;

}

return cont;

}

int maior(Lista \*l)

{

Lista \*aux = l;

if(lista\_vazia(aux))return 0;

int maior = aux->elemento;

while(aux != NULL){

if(aux->elemento>maior) maior = aux->elemento;

aux = aux->prox;

}

return maior;

}

Lista\* concatenar(Lista \*\*A, Lista \*\*B)

{

Lista \*C = cria\_lista();

Lista \*temp = \*A;

Lista \*temp\_aux = \*B;

while(temp!=NULL){

insere\_elem(&C,temp->elemento);

temp = temp->prox;

}

while(temp\_aux!=NULL){

insere\_elem(&C,temp\_aux->elemento);

temp\_aux = temp\_aux->prox;

}

return C;

}

\*.h

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct encadeada Lista;

Lista \*cria\_lista();

int insere\_elem(Lista \*\*l, int n);

int remove\_elem(Lista \*\*l, int n);

int lista\_vazia(Lista \*l);

void print\_lista(Lista \*l);

int remove\_todos(Lista \*\*l, int elem);

int insere\_fim(Lista \*l, int elem);

int remove\_pares(Lista \*\*l);

int tamanho(Lista \*l);

int maior(Lista \*l);

Lista\* concatenar(Lista \*\*A, Lista \*\*B);

**DINAMICA ENCADEADA ORDENADA (SEM CABEÇALHO)**

\*.c

#include "dinencord.h"

struct encadeada

{

int elemento;

struct encadeada\* prox;

};

Lista \*cria\_lista()

{

return 0;//NULL

}

int lista\_vazia(Lista \*l)

{

if(l->prox==NULL) return 1;

return 0;

}

int insere\_elem\_ord(Lista \*\*l,int elem)

{

Lista \*temp = (Lista \*) malloc(sizeof(Lista));

if(temp==NULL) return 0;

temp->elemento = elem;

if((\*l)==NULL||elem<=(\*l)->elemento){

temp->prox = \*l;

\*l = temp;

return 1;

}

Lista \*aux = \*l;

while(aux->prox!=NULL&&aux->prox->elemento<elem){

aux = aux->prox;

}

temp->prox = aux->prox;

aux->prox = temp;

return 1;

}

int remove\_elem\_ord(Lista \*\*l, int n)

{

if((\*l)==NULL||n < (\*l)->elemento) return 0;

Lista \*temp;

Lista \*temp\_aux;

temp = \*l;

if(n==(\*l)->elemento){

\*l = temp->prox;

free(temp);

return 1;

}

while(temp->prox!=NULL && temp->prox->elemento < n){

temp = temp->prox;

}

if(temp->prox==NULL||temp->prox->elemento > n) return 0;

temp\_aux = temp->prox;

temp->prox = temp\_aux->prox;

free(temp\_aux);

return 1;

}

void print\_lista(Lista \*l)

{

Lista \* temp;

temp = l;

printf("Lista = {");

while(temp!=NULL){

if(temp->prox!=NULL)printf("%d ",temp->elemento);

else printf("%d",temp->elemento);

temp = temp->prox;

}

printf("}\n");

}

Lista \* intercala\_ord(Lista \*\*A, Lista \*\*B)

{

Lista \*C = cria\_lista();

Lista \*temp;

Lista \*temp\_aux;

temp = \*A;

while(temp!=NULL){

insere\_elem\_ord(&C,temp->elemento);

temp = temp->prox;

}

temp\_aux = \*B;

while(temp\_aux!=NULL){

insere\_elem\_ord(&C,temp\_aux->elemento);

temp\_aux = temp\_aux->prox;

}

return C;

}

int remove\_impares(Lista \*\*l)

{

int cont=0;

Lista \*temp = \*l;

if(lista\_vazia(temp)) return 0;

while(temp!=NULL){

if((temp->elemento) % 2 != 0){

remove\_elem\_ord(l,temp->elemento);

cont++;

}

temp = temp->prox;

}

return cont;

}

int menor(Lista \*\*l)

{

Lista \*temp = \*l;

if(lista\_vazia(temp))return 0;

int menor = temp->elemento;

while(temp != NULL){

if(temp->elemento<menor)menor = temp->elemento;

temp = temp->prox;

}

return menor;

}

int tamanho(Lista \*\*l)

{

int tam=0;

Lista \*temp = \*l;

if(lista\_vazia(temp))return 0;

while(temp != NULL){

temp = temp->prox;

tam++;

}

return tam;

}

int iguais(Lista \*A, Lista \*B)

{

if(lista\_vazia(A) && lista\_vazia(B))return 1;

int cont = 0;

if(tamanho(&A)!= tamanho(&B))return 0;

Lista \*temp = A, \*temp\_aux = B;

while(temp != NULL){

if(temp->elemento != temp\_aux->elemento) return 0;

temp = temp->prox;

temp\_aux = temp\_aux->prox;

cont++;

}

if(tamanho(&A) == cont) return 1;

return 0;

}

\*.h

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct encadeada Lista;

Lista \*cria\_lista();

int lista\_vazia(Lista \*l);

void print\_lista(Lista \*l);

int insere\_elem\_ord(Lista \*\*l, int n);

int remove\_elem\_ord(Lista \*\*l, int n);

Lista \* intercala\_ord(Lista \*\*A, Lista \*\*B);

int remove\_impares(Lista \*\*l);

int menor(Lista \*\*l);

int iguais(Lista \*A, Lista \*B);

**DINAMICA ENCADEADA ORDENADA (COM CABEÇALHO)**

\*.c

#include "dinencord.h"

struct no

{

int elemento;

struct no \* prox;

};

Lista \*cria\_lista()

{

Lista \*L = (Lista \*) malloc(sizeof(Lista));

if(L){

L->elemento=0;

L->prox =NULL;

}

return L;

}

int lista\_vazia(Lista \*l)

{

if((l->prox=NULL)) return 1;

return 0;

}

int insere\_ord(Lista \*\*l, int elem)

{

Lista \*no = (Lista \*) malloc(sizeof(Lista));

if(no){

no->elemento = elem;

Lista \*temp = \*l;

while(temp->prox!=NULL && temp->prox->elemento < elem){

temp = temp->prox;

}

no->prox = temp->prox;

temp->prox = no;

(\*l)->elemento++;

return 1;

}

return 0;

}

int remove\_ord(Lista \*\*l, int elem)

{

if((\*l)==NULL) return 0;

Lista \*temp = (\*l)->prox;

if(elem==(\*l)->prox->elemento){

(\*l)->prox = temp->prox;

free(temp);

(\*l)->elemento--;

return 1;

}

while(temp!=NULL && temp->prox->elemento!=elem) temp = temp->prox;

if(temp->prox==NULL) {

return 0;

}

Lista \*temp\_aux = temp->prox;

temp->prox = temp\_aux->prox;

free(temp\_aux);

(\*l)->elemento--;

return 1;

}

void print\_lista(Lista \*l)

{

printf("{ ");

Lista \*temp= l->prox;

while(temp!=NULL){

printf(" %d ",temp->elemento);

temp = temp->prox;

}

printf(" }\n");

}

int tamanho(Lista \*L)

{

if(lista\_vazia(L)) return 0;

int tam=0;

while(L!=NULL){

L = L->prox;

tam++;

}

return tam;

}

int iguais(Lista \*A, Lista \*B)

{

if(lista\_vazia(A)&&lista\_vazia(B)) return 1;

int iguais = 0;

if(tamanho(A)!=tamanho(B)) return 0;

Lista \*temp = A->prox, \*temp\_aux = B->prox;

while(temp!=NULL){

if(temp->elemento!=temp\_aux->elemento) return 0;

temp = temp->prox;

temp\_aux = temp\_aux->prox;

iguais++;

}

if(iguais==tamanho(A)) return 1;

return 0;

}

int remove\_impares(Lista \*\*A)

{

if(lista\_vazia(\*A)) return 0;

Lista \*temp = (\*A)->prox;

int removido=0;

while(temp!=NULL){

if(temp->elemento%2!=0){

remove\_ord(A,temp->elemento);

removido++;

}

temp = temp->prox;

}

return removido;

}

int menor(Lista \*A)

{

if(lista\_vazia(A)) return 0;

Lista \*temp = A;

int menor = temp->prox->elemento;

while(temp->prox!=NULL){

if(temp->prox != NULL && temp->prox->elemento<menor)menor = temp->prox->elemento;

temp = temp->prox;

}

return menor;

}

Lista\* intercalar(Lista \*\*A, Lista \*\*B)

{

Lista \*C = (Lista \*) malloc(sizeof(Lista));

if(C==NULL) return NULL;

Lista \*temp = (\*A)->prox;

Lista \*temp\_aux = (\*B)->prox;

while(temp!=NULL){

insere\_ord(&C,temp->elemento);

temp = temp->prox;

}

while(temp\_aux!=NULL){

insere\_ord(&C,temp\_aux->elemento);

temp\_aux = temp\_aux->prox;

}

C->elemento = (\*A)->elemento+(\*B)->elemento;

return C;

}

\*.h

#include <stdlib.h>

#include <stdio.h>

typedef struct no Lista;

Lista \*cria\_lista();

int lista\_vazia(Lista \*l);

int insere\_ord(Lista \*\*l, int elem);

int remove\_ord(Lista \*\*l, int elem);

void print\_lista(Lista \*l);

int tamanho(Lista \*L);

int remove\_impares(Lista \*\*A);

int menor(Lista \*A);

int iguais(Lista \*A, Lista \*B);

Lista\* intercalar(Lista \*\*A,Lista \*\*B);

**DINAMICA ENCADEADA NAO ORDENADA CIRCULAR**

\*.c

#include "dinenccircularnord.h"

struct lista{

int elemento;

struct lista \*prox;

};

Lista\* cria\_lista()

{

Lista \*C = (Lista \*) malloc(sizeof(Lista));

if(C){

C->prox = C;

C->elemento = 0;

}

return C;

}

int lista\_vazia(Lista \*\*l)

{

Lista \*temp = \*l;

if(temp->prox == temp) return 1;

return 0;

}

int insere\_elem(Lista \*l, int elem)

{

Lista \*no = (Lista \*) malloc(sizeof(Lista));

if(no==NULL) return 0;

no->elemento = elem;

if(lista\_vazia(&l)){

no->prox = l;

l->prox = no;

return 1;

}

Lista \*aux = l;

Lista \*temp = l;

while(aux->prox!=temp){

aux = aux->prox;

}

no->prox = aux->prox;

aux->prox = no;

return 1;

}

int remove\_elem(Lista \*\*l, int elem)

{

if(lista\_vazia(l)) return 0;

Lista \*temp = \*l;

Lista \*temp\_aux = \*l;

while(temp\_aux->prox!=temp && temp\_aux->prox->elemento!=elem){

temp\_aux = temp\_aux->prox;

}

if(temp\_aux->prox==temp) return 0;

Lista \*aux = temp\_aux->prox;

temp\_aux->prox = aux->prox;

free(aux);

return 1;

}

int tamanho(Lista \*l)

{

if(lista\_vazia(&l)) return 0;

int i=0;

while(l!=NULL){

i++;

l = l->prox;

}

return i;

}

void imprime\_lista(Lista \*l)

{

if(lista\_vazia(&l)) {

printf("{ }\n");

return;

}

Lista \*temp\_aux = l;

Lista \*temp = l;

printf("{ ");

while(temp\_aux->prox!=temp){

printf("%d ",temp\_aux->prox->elemento);

temp\_aux = temp\_aux->prox;

}

printf(" }\n");

}

int insere\_inicio(Lista \*\*l, int elem)

{

Lista \*temp = \*l;

Lista \*no = (Lista \*) malloc(sizeof(Lista));

if(no==NULL) return 0;

no->elemento = elem;

no->prox = temp->prox;

temp->prox = no;

return 1;

}

int insere\_posicao(Lista \*l, int elem ,int posicao)

{

Lista \*no = (Lista \*) malloc(sizeof(Lista));

if(no==NULL) return 0;

no->elemento = elem;

int contador=0;

if(lista\_vazia(&l)){

no->prox = l;

l->prox = no;

return 1;

}

Lista \*aux = l;

Lista \*temp = l;

while(aux->prox!=temp&&contador < posicao){

aux = aux->prox;

contador++;

}

no->prox = aux->prox;

aux->prox = no;

return 1;

}

int remove\_posicao(Lista \*\*l, int posicao)

{

if(lista\_vazia(l)) return 0;

Lista \*temp = \*l;

Lista \*temp\_aux = \*l;

int contador=0;

while(temp\_aux->prox!=temp && contador<posicao){

temp\_aux = temp\_aux->prox;

contador++;

}

if(temp\_aux->prox==temp) return 0;

Lista \*aux = temp\_aux->prox;

temp\_aux->prox = aux->prox;

free(aux);

return 1;

}

int maior(Lista \*l)

{

if(lista\_vazia(&l)) return 0;

int maior = l->prox->elemento;

Lista \*temp = l;

Lista \*temp\_aux = l;

while(temp\_aux->prox!=temp){

if(temp\_aux->prox->elemento > maior) maior =

temp\_aux->prox->elemento;

temp\_aux = temp\_aux->prox;

}

return maior;

}

\*.h

#include <stdlib.h>

#include <stdio.h>

typedef struct lista Lista;

Lista \*cria\_lista();

int lista\_vazia(Lista \*\*l);

int insere\_elem(Lista \*l, int elem);

int remove\_elem(Lista \*\*l, int elem);

int tamanho(Lista \*l);

void imprime\_lista(Lista \*l);

int insere\_inicio(Lista \*\*l, int elem);

int insere\_posicao(Lista \*l, int elem,int posicao);

int remove\_posicao(Lista \*\*l, int posicao);

int maior(Lista \*l);

**DINAMICA DUPLAMENTE ENCADEADA**

\*c

#include "encduplo.h"

struct no

{

int info;

struct no \* prox;

struct no \* ant;

};

Lista \*cria\_lista(){

Lista \*L = (Lista \*) malloc(sizeof(Lista));

if(L){

L->prox=NULL;

L->ant=NULL;

L->info=0;

}

return L;

}

int lista\_vazia(Lista \*L)

{

if(L->prox==NULL && L->ant==NULL) return 1;

return 0;

}

int insere\_elemento(Lista \*L, int elem)

{

if(L==NULL) return 0;

Lista \*temp = L;

Lista \*no = cria\_lista();

if(no){

no->info = elem;

no->prox = temp->prox;

no->ant = temp;

temp->prox = no;

return 1;

}

return 0;

}

int remove\_elemento(Lista \*L, int elem)

{

if(L == NULL) return 0;

if(lista\_vazia(L)) return 0;

Lista \*temp = L;

Lista \*temp\_aux;

while(temp->prox != NULL && temp->prox->info != elem){

temp = temp->prox;

}

if(temp->prox==NULL) return 0;

temp\_aux = temp->prox;

temp->prox = temp\_aux->prox;

if(temp\_aux->prox!=NULL){

Lista \*aux = temp\_aux->prox;

aux->ant = temp\_aux->ant;

}

free(temp\_aux);

return 1;

}

void print\_lista(Lista \*L)

{

if(L==NULL) return;

if(lista\_vazia(L)) {

printf("{ }\n");

return;

}

Lista \*temp = L;

printf("{ ");

while((temp)->prox!=NULL){

printf("%d ",(temp)->prox->info);

(temp) = (temp)->prox;

}

printf(" }\n\n");

}

int tamanho(Lista \*l)

{

if(l==NULL||lista\_vazia(l)) return 0;

Lista \*temp = l;

int tam = 0;

while((temp)->prox!=NULL){

tam++;

(temp) = (temp)->prox;

}

return tam;

}

int remove\_todos(Lista \*l,int elem) //nao funciona corretamente

{

if(l==NULL) return 0;

if(lista\_vazia(l)) return 0;

int removido = 0;

Lista \*temp = l->prox;

Lista \*temp\_aux;

while(temp != NULL){

if(temp->info == elem){

temp\_aux = temp;

temp = temp\_aux->prox;

if(temp\_aux->prox != NULL){

Lista \*aux = temp\_aux->prox;

aux->ant = temp\_aux->ant;

}

free(temp\_aux);

removido++;

}

temp = temp->prox;

}

if(removido)return 1;

return 0;

}

int remove\_maior(Lista \*l)

{

if(l==NULL || lista\_vazia(l)) return 0;

int maior = l->prox->info;

Lista \*temp = l;

while(temp->prox!=NULL){

if(temp->prox->info > maior) maior=temp->prox->info;

temp = temp->prox;

}

remove\_elemento(l,maior);

return 1;

}

Lista \*multiplo\_3(Lista \*l)

{

Lista \*C = cria\_lista();

if(C){

Lista \*temp = l;

while(temp->prox!=NULL){

if(temp->prox->info%3==0){

insere\_elemento(C,temp->prox->info);

}

temp = temp->prox;

}

}

return C;

}

\*.h

#include <stdio.h>

#include <stdlib.h>

typedef struct no Lista;

Lista \*cria\_lista();

int lista\_vazia(Lista \*L);

int insere\_elemento(Lista \*L, int elem);

int remove\_elemento(Lista \*L, int elem);

void print\_lista(Lista \*L);

int tamanho(Lista \*l);

int remove\_todos(Lista \*l,int elem);

int remove\_maior(Lista \*l);

Lista \* multiplo\_3(Lista \*l);