**/\*numero 01:**

Scrivere una procedura che data in ingresso una lista di interi cancelli il primo elemento che e' preceduto e seguito da 2 elementi uguali tra loro, altrimenti rimane invariata. \*/

#include<stdio.h>

#include<stdlib.h>

/\* //Questo modo di definire la lista non e’ del tutto corretto

typedef struct Il\_node\_struct{

int info;

struct Il\_node\_struct \*next;

}Il\_node;

Il\_node \*add(Il\_node \*head, int v){ //inserisco l'elemento in coda, ovvero in ordine di scrittura

Il\_node \*corr=head;

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=v;

new\_node->next=NULL;

if(head==NULL){

new\_node->next=head;

head=new\_node;

}

else if(head!=NULL){

while(corr->next!=NULL){ //se mettessi while(corr!=NULL) e' sbagliato perche' lo considera come while(head!=NULL)

corr=corr->next;

}

corr->next=new\_node;

}

return head;

}

void canc(Il\_node \*head){

Il\_node \*prec=head, \*corr=prec->next, \*succ=corr->next; //le variabili che scandiscono la lista devono andare avanti di pari passo

int trovato=0;

if(head!=NULL){ //se la lista e' vuota non devo fare niente

if(head->next!=NULL){ //controllo quando sono sicuro che esista il campo next

while(succ!=NULL && trovato==0){

if(prec->info==succ->info){

trovato=1;

prec->next=succ;

free(corr); //cosi corr che non indica piu' niente viene tolto

//oppure

/\*

trovato=1;

prec->next=corr->next;

prec=prec->next;

free(corr); //se non viene tolto non e' errore ma crea garbage

\*/

}

else if(prec->info!=succ->info){ //devo continuare ad andare avanti con i puntatori

prec=corr;

corr=succ;

succ=corr->next;

}

}

}

}

}

void print(Il\_node \*head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Il\_node \*head=NULL;

int n;

scanf("%d",&n);

while(n!=0){

head=add(head,n);

scanf("%d",&n);

}

canc(head);

print(head);

printf("\n");

return 0;

}

\*/

//Quest’altro modo di definire la lista e’ piu’ corretto

#include<stdio.h>

#include<stdlib.h>

struct Ilnode{

int info;

struct Ilnode \*next;

}Ilnode;

typedef struct Ilnode \*ListaElementi;

void add(ListaElementi \*head, int v){ //inserisco l'elemento in coda, ovvero in ordine di scrittura

ListaElementi new\_node=(ListaElementi)malloc(sizeof(struct Ilnode));

new\_node->info=v;

new\_node->next=NULL;

if((\*head)==NULL){ //\*head va sempre tra parentesi!!!

new\_node->next=\*head;

(\*head)=new\_node;

}

else if((\*head)!=NULL){

ListaElementi corr=\*head;

while(corr->next!=NULL){ //se mettessi while(corr!=NULL) e' sbagliato perche' lo considera come while(head!=NULL)

corr=corr->next;

}

corr->next=new\_node;

}

}

void canc(ListaElementi \*head){

ListaElementi prec=\*head, corr=prec->next, succ=corr->next; //le variabili che scandiscono la lista devono andare avanti di pari passo

int trovato=0;

if((\*head)!=NULL){ //se la lista e' vuota non devo fare niente

if((\*head)->next!=NULL){ //controllo quando sono sicuro che esista il campo next

while(succ!=NULL && trovato==0){

if(prec->info==succ->info){

trovato=1;

prec->next=succ;

free(corr); //cosi corr che non indica piu' niente viene tolto

//oppure

/\*

trovato=1;

prec->next=corr->next;

prec=prec->next;

free(corr); //se non viene tolto non e' errore ma crea garbage

\*/

}

else if(prec->info!=succ->info){ //devo continuare ad andare avanti con i puntatori

prec=corr;

corr=succ;

succ=corr->next;

}

}

}

}

}

void print(ListaElementi head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

ListaElementi head=NULL;

int n;

scanf("%d",&n);

while(n!=0){

add(&head, n);

scanf("%d",&n);

}

canc(&head);

print(head);

printf("\n");

return 0;

}

**/\*numero 02:**

eliminare il primo elemento in testa ad una lista \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int x){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=x;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

//con questa versione si crea garbage, c'e' bisogno di liberare la memoria

void deleteprimo(Lista \*head){

if((\*head)!=NULL){

if((\*head)->next==NULL) (\*head)=NULL;

else if((\*head)->next!=NULL){

(\*head)=(\*head)->next;

}

}

}

//questa versione e' ancora meglio

void deleteprimocorretto(Lista \*head){

if((\*head)!=NULL){

Lista el=\*head;

(\*head)=(\*head)->next;

free(el);

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

// deleteprimo(&head);

deleteprimocorretto(&head);

print(head);

return 0;

}

**/\*numero 03:**

eliminare l'ultimo elemento di una lista se non e' vuota \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int x){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=x;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void deleteultimo(Lista \*head){

if((\*head)!=NULL){

Lista corr=\*head, prec=NULL;

while(corr->next!=NULL){

prec=corr;

corr=corr->next;

}

if(prec==NULL) (\*head)=NULL; //caso di un solo elemento

else if(prec!=NULL){

prec->next=NULL;

free(corr);

}

}

}

void deleteultimoo(Lista \*head){ //oppure

if((\*head)!=NULL){

if((\*head)->next==NULL){

free(\*head);

(\*head)=NULL;

}else if((\*head)->next!=NULL){

Lista corr=(\*head)->next, prec=(\*head); //prec punta inzialmente al primo elemento e corr al secondo

while(corr->next!=NULL){

corr=corr->next;

prec=prec->next;

}

free(corr);

prec->next=NULL;

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

// deleteultimo(&head);

deleteultimoo(&head);

print(head);

return 0;

}

**/\*numero 04:**

eliminare l’elemento che precede/segue la prima/ultima occorrenza di un valore \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int x){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=x;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

//eliminare l'elemento di una lista che precede la prima occorrenza di x se non e' vuota

void deleteprec(Lista \*head, int m){

if((\*head)!=NULL){

Lista corr=\*head, prec=NULL;

int trovato=0;

while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->next->info==m) trovato=1;

else if(corr->next->info!=m){

prec=corr;

corr=corr->next;

}

}

if(trovato==1){

if(prec==NULL){

Lista el=\*head;

(\*head)=(\*head)->next;

free(el);

}else if(prec!=NULL){

prec->next=corr->next;

free(corr);

}

}

}

}

//eliminare l'elemento di una lista che segue la prima occorrenza di x se non e' vuota

void deletesuc(Lista \*head, int m){

if((\*head)!=NULL){

Lista corr=\*head;

int trovato=0;

while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->info==m) trovato=1;

else if(corr->info!=m) corr=corr->next;

}

if(trovato=1 && corr->next!=NULL){

corr->next=corr->next->next;

}

}

}

//conta quante volte l’elemento m e’ presente nella lista

int contaocc(Lista head, int m){

int count=0;

while(head!=NULL){

if(head->info==m) count++;

head=head->next;

}

return count;

}

//elimino l'elemento che precede l'ultima occorrenza di x

void deleteprecorr(Lista \*head, int m){

int v=0;

Lista corr=\*head, prec=NULL;

while(v<contaocc((\*head),m) && corr->next!=NULL){

if(corr->next->next->info==m) v++;

prec=corr;

corr=corr->next;

}

if(prec==NULL){

Lista el=\*head;

(\*head)=(\*head)->next;

free(el);

}else if(prec!=NULL){

prec->next=prec->next->next;

free(corr);

}

}

//elimino l'elemento che segue l'ultima occorrenza di x

void deletesucorr(Lista \*head, int m){

int v=0;

Lista corr=\*head, prec=NULL;

while(v<contaocc((\*head),m) && corr->next!=NULL){

if(corr->info==m) v++;

prec=corr;

corr=corr->next;

}

if(prec!=NULL){

prec->next=prec->next->next;

free(corr);

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

// deleteprec(&head, m);

// deletesuc(&head, m);

// deleteprecorr(&head, m);

deletesucorr(&head, m);

print(head);

return 0;

}

**/\*numero 05:**

Scrivere una procedura che cambia l'ordine degli ultimi 2 elementi di una lista data \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void cambia(Lista \*head){

if((\*head)!=NULL){ //se la lista non e' vuota

if((\*head)->next!=NULL){ //se esiste il campo next della lista

if((\*head)->next->next==NULL){ //se la lista ha solo 2 elementi

Lista el=\*head;

(\*head)->next->next=el;

(\*head)=(\*head)->next;

(\*head)->next->next=NULL;

}else if((\*head)->next->next!=NULL){ //se la lista non ha solo 2 elementi

Lista prec=\*head, corr=prec->next;

while(corr->next->next!=NULL){

prec=corr;

corr=corr->next;

}

corr->next->next=corr;

prec->next=corr->next;

corr->next=NULL;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

cambia(&head);

print(head);

return 0;

}

**/\*numero 06:**

Scrivere una procedura che sposti in ultima posizione il primo elemento di una lista di interi maggiore di un valore dato, supponiamo che esista sempre uno maggiore. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void sposta(Lista \*head, int x){

if((\*head)!=NULL){ //se la lista e' vuota non si fa nulla

if((\*head)->next!=NULL){ //se c'e' l'elemento successivo

Lista prec=NULL, corr=\*head, succ=corr->next;

if((\*head)->info>x){

while(corr->next!=NULL)

corr=corr->next; //scorro la lista per arrivare fino in fondo

corr->next=\*head; //posiziono il primo nodo maggiore in fondo

(\*head)=(\*head)->next; //e il puntatore head diventa il secondo elemento

corr->next->next=NULL;

}else if((\*head)->info<=x){

if((\*head)->next!=NULL)

sposta(&(\*head)->next, x);

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

sposta(&head, x);

print(head);

return 0;

}

/\*ES 6 ESAME 03/06/14: Scrivere una procedura che, dati in ingresso attraverso opportuni parametri una lista di interi e un intero x, elimina dalla lista l’ultimo elemento maggiore di x. Se nessun elemento della lista `e maggiore di x, la procedura elimina, se esiste, l’ultimo elemento della lista. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int contamaggiori(Lista head, int x){

int count=0, lengh=0;

while(head!=NULL){

if(head->info>x) count++;

lengh++;

head=head->next;

}

if(lengh==count) return 0;

return count;

}

void delete(Lista \*head, int x){

if(\*head!=NULL){ //questa condizione e' fondamentale

Lista corr=\*head, prec=NULL, preprec=NULL;

// printf("conta%d\n",contamaggiori(\*head, x));

if(contamaggiori(\*head, x)!=0){

int v=0;

while(v<contamaggiori(\*head, x)){

if(corr->info>x) v++;

preprec=prec;

prec=corr;

corr=corr->next;

}

// printf("corr%d\n",corr->info);

// printf("v%d\n",v);

if(preprec==NULL){

(\*head)=(\*head)->next;

free(prec);

}else if(preprec!=NULL){

if(corr!=NULL){

preprec->next=preprec->next->next;

free(prec);

}else if(corr==NULL) preprec->next=NULL;

}

}else if(contamaggiori(\*head, x)==0){

while(corr->next!=NULL){

prec=corr;

corr=corr->next;

}

if(prec!=NULL){

prec->next=NULL;

free(corr);

}else if(prec==NULL)

(\*head)=NULL;

}

}

}

void delete1(Lista \*l, int x){ //soluzione del prof

if(\*l!=NULL){

Lista corr=\*l, prec=NULL, prec\_prec=NULL, prec\_ultimo=NULL;

int trovato=0;

while(corr!=NULL){

if(corr->info>x){

trovato=1;

prec\_ultimo=prec;

}

prec\_prec=prec;

prec=corr;

corr=corr->next;

}

if(trovato)

if(prec\_ultimo==NULL){

corr=\*l;

\*l=(\*l)->next;

free(corr);

}else{

corr=prec\_ultimo->next;

prec\_ultimo->next=corr->next;

free(corr);

}

else

if(prec\_prec==NULL){

corr=\*l;

\*l=(\*l)->next;

free(corr);

}else{

corr=prec\_prec->next;

prec\_prec->next=NULL;

free(corr);

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

print(head);

printf("\n");

// delete(&head, x);

delete1(&head, x);

print(head);

printf("\n");

return 0;

}

**/\*ES 6 ESAME 03/06/14:**

Scrivere una procedura che, dati in ingresso attraverso opportuni parametri una lista di interi e un intero x, elimina dalla lista l’ultimo elemento maggiore di x. Se nessun elemento della lista `e maggiore di x, la procedura elimina, se esiste, l’ultimo elemento della lista. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int contamaggiori(Lista head, int x){

int count=0, lengh=0;

while(head!=NULL){

if(head->info>x) count++;

lengh++;

head=head->next;

}

if(lengh==count) return 0;

return count;

}

void delete(Lista \*head, int x){

if(\*head!=NULL){ //questa condizione e' fondamentale

Lista corr=\*head, prec=NULL, preprec=NULL;

// printf("conta%d\n",contamaggiori(\*head, x));

if(contamaggiori(\*head, x)!=0){

int v=0;

while(v<contamaggiori(\*head, x)){

if(corr->info>x) v++;

preprec=prec;

prec=corr;

corr=corr->next;

}

// printf("corr%d\n",corr->info);

// printf("v%d\n",v);

if(preprec==NULL){

(\*head)=(\*head)->next;

free(prec);

}else if(preprec!=NULL){

if(corr!=NULL){

preprec->next=preprec->next->next;

free(prec);

}else if(corr==NULL) preprec->next=NULL;

}

}else if(contamaggiori(\*head, x)==0){

while(corr->next!=NULL){

prec=corr;

corr=corr->next;

}

if(prec!=NULL){

prec->next=NULL;

free(corr);

}else if(prec==NULL)

(\*head)=NULL;

}

}

}

void delete1(Lista \*l, int x){ //soluzione del prof

if(\*l!=NULL){

Lista corr=\*l, prec=NULL, prec\_prec=NULL, prec\_ultimo=NULL;

int trovato=0;

while(corr!=NULL){

if(corr->info>x){

trovato=1;

prec\_ultimo=prec;

}

prec\_prec=prec;

prec=corr;

corr=corr->next;

}

if(trovato)

if(prec\_ultimo==NULL){

corr=\*l;

\*l=(\*l)->next;

free(corr);

}else{

corr=prec\_ultimo->next;

prec\_ultimo->next=corr->next;

free(corr);

}

else

if(prec\_prec==NULL){

corr=\*l;

\*l=(\*l)->next;

free(corr);

}else{

corr=prec\_prec->next;

prec\_prec->next=NULL;

free(corr);

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

print(head);

printf("\n");

// delete(&head, x);

delete1(&head, x);

print(head);

printf("\n");

return 0;

}

**/\*ES 6 ESAME 25/06/14:**

Scrivere una procedura che, dati in ingresso, una lista di interi e un intero x, sposta in prima posizione l'ultima occorrenza dell'elemento x. N.B.: non e' consentito utilizzare assegnamenti sui campi info degli elementi della lista. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int contaocc(Lista head, int x){

int count=0;

while(head!=NULL){

if(head->info==x) count++;

head=head->next;

}

return count;

}

void sposta(Lista \*head, int x){

if((\*head)!=NULL){

//per assicurarci che abbia almeno due elementi perche' se ne ha uno non facciamo niente

if((\*head)->next!=NULL && contaocc((\*head), x)>0){

int v=0;

Lista corr=\*head, prec=NULL, prev=\*head;

while(v<contaocc(\*head, x) && corr->next!=NULL){

if(corr->next->info==x) v++;

prec=corr;

corr=corr->next;

}

(\*head)=prec->next;

prec->next=corr->next;

corr->next=prev;

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

sposta(&head, x);

print(head);

return 0;

}

**/\*ES 6 ESAME 14/07/14:**

Scrivere una procedura che, data una lista di interi e un valore intero positivo n, sposta in fondo alla lista i primi n elementi. Se n e' maggiore o uguale alla lunghezza della lista, essa rimane inalterata. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){ //inserisco l'elemento in coda, ovvero in ordine di scrittura

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL) //se mettessi while(corr!=NULL) e' sbagliato perche' lo considera come while(head!=NULL)

corr=corr->next;

corr->next=new;

}

}

int lunghezza(Lista head){

int conta=0;

while(head!=NULL){

conta++;

head=head->next;

}

return conta;

}

void sposta(Lista \*head, int n){

if(n<lunghezza(\*head)){

int conta=lunghezza(\*head), contael=0;

Lista corr=\*head, prec=NULL, succ=\*head, prev=\*head;

while(contael<n){

prec=corr;

corr=corr->next;

contael++;

}

if(prec!=NULL){

while(succ->next!=NULL)

succ=succ->next;

succ->next=prev;

// printf("%+d\n", prec->info);

// printf("%+d\n", corr->info);

(\*head)=corr;

prec->next=NULL;

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, val;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &val);

sposta(&head, val);

print(head);

return 0;

}

**/\*ES 5 ESAME 05/09/14:**

Scrivere una procedura che, dati in ingresso attraverso opportuni parametri una lista di interi e un intero n, aggiunge n in fondo alla lista se n non `e gi`a presente in essa, lascia la lista inalterata altrimenti. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int n){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=n;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int member(Lista head, int m){

int trovato=0;

while(head!=NULL && trovato==0){

if(head->info==m) trovato=1;

head=head->next;

}

return trovato;

}

void copia(Lista \*head, int m){

if(member(\*head, m)==0) add(&(\*head),m);

}

void copia1(Lista \*l, int n){

Lista prec=NULL, corr=\*l;

int trovato=0;

while(corr!=NULL && !trovato){

if(corr->info==n) trovato=1;

else{

prec=corr;

corr=corr->next;

}

}

if(!trovato){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=n;

new->next=NULL;

if(prec==NULL) \*l=new;

else prec->next=new;

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

// copia(&head, m);

copia1(&head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 5 ESAME 18/12/14:**

Scrivere una procedura che, dato una lista di interi e un valore x, elimina, se esiste, il primo elemento della lista seguito da un elemento che contiene x. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void canc(Lista \*head, int x){

if((\*head)!=NULL){

if((\*head)->next!=NULL){

int trovato=0;

Lista corr=\*head;

while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->info==x) trovato=1;

else if(corr->info!=x) corr=corr->next;

}

if(trovato==1){

if(corr->next!=NULL){

Lista prev=corr->next;

corr->next=corr->next->next;

free(prev);

}

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

canc(&head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 3 ESAME 12/01/15:**

Scrivere una procedura che, data una lista di interi in cui tutti gli elementi con valore pari precedono tutti gli elementi con valore dispari, e dato un intero n, inserisce n nella lista in modo che anche nella lista risultante tutti gli elementi con valore pari precedano tutti gli elementi con valore dispari. HINT: Inserire l'elemento in testa alla lista se e' pari oppure la lista e' vuota, altrimenti inserirlo in fondo. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void inserisci(Lista \*head, int n){ //soluzione del prof.

Lista new=(Lista)malloc(sizeof(struct el)), prec=NULL, corr=\*head;

new->info=n;

new->next=NULL;

if(n%2==0 || (\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if(n%2!=0 || (\*head)!=NULL){

while(corr!=NULL){

prec=corr;

corr=corr->next;

}//all'uscita dal ciclo corr=NULL e prec e' l'ultimo elemento

prec->next=new;

/\* //oppure

while(corr->next!=NULL) //si esce dal ciclo con corr che e' l'ultimo elemento

corr=corr->next;

corr->next=new;

\*/

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

inserisci(&head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 3 ESAME 09/02/15: (perche’ funziona la prima procedura?)**

Scrivere una procedura che, data una lista di interi ed un intero n, cancella dalla lista, se c'e', l'ultimo elemento che contiene n.\*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void canc(Lista \*head, int n){

Lista prec=NULL, corr=\*head, save=NULL;

int trovato=0;

if((\*head)!=NULL){

while(corr!=NULL){

if(corr->info==n){

save=prec;

trovato=1;

}

prec=corr;

corr=corr->next;

}

if(trovato==1){

if(save==NULL){

corr=(\*head);

(\*head)=(\*head)->next;

}else if(save!=NULL){

corr=save->next;

save->next=save->next->next;

}

free(corr);

}

}

}

/\* //oppure

int contaocc(Lista head, int n){

int count=0;

while(head!=NULL){

if(head->info==n) count++;

head=head->next;

}

return count;

}

void canc(Lista \*head, int n){

if((\*head)!=NULL){

int v=0;

//se non mettessi l'if comprendente l'if-else dopo il while eliminereri in ogni caso il primo elemento della lista

if(contaocc(\*head, n)>0){

Lista prec=NULL, preprec=NULL, corr=\*head;

while(v<contaocc(\*head, n)){

if(corr->info==n) v++;

preprec=prec;

prec=corr;

corr=corr->next;

}

if(preprec==NULL){

(\*head)=(\*head)->next;

free(prec);

}else if(preprec!=NULL){

preprec->next=preprec->next->next;

free(preprec);

}

}

}

} \*/

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

canc(&head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 4 ESAME 23/06/15:**

Scrivere una procedura che, date due liste di interi, aggiunge alla seconda lista tutti gli elementi della prima che non appartengono anche alla seconda. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void agg(Lista \*head, int x){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=x;

new->next=(\*head);

(\*head)=new;

}

int member(Lista head, int v){

int trovato=0;

while(head!=NULL && trovato==0){

if(head->info==v) trovato=1;

else if(head->info!=v) head=head->next;

}

return trovato;

}

void aggiungi(Lista \*head, Lista \*head1){

if((\*head)!=NULL){

Lista corr=\*head;

while(corr!=NULL){

if(member(\*head1, corr->info)==1)

corr=corr->next;

else if(member(\*head1, corr->info)==0){

//add(&(\*head1), corr->info); //aggiunge in coda i nuovi elementi

agg(&(\*head1), corr->info); //aggiunge in testa

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL, head1=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

while(m!=0){

add(&head1, m);

scanf("%d", &m);

}

print(head);

printf("\n");

print(head1);

printf("\n");

aggiungi(&head, &head1);

print(head1);

printf("\n");

return 0;

}

**/\*ES 4 ESAME 15/07/15:**

Scrivere una procedura che, date due liste di interi, elimina dalla seconda lista tutti gli elementi che non occorrono anche nella prima. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int member(Lista head, int val){

int trovato=0;

while(head!=NULL && trovato==0){

if(head->info==val) trovato=1;

else head=head->next;

}

return trovato;

}

void elimina(Lista \*head, Lista \*head1){

if(head1!=NULL){

Lista prec=NULL, corr=\*head1;

while(corr!=NULL){

if(member(\*head, corr->info)==1){

prec=corr;

corr=corr->next;

}else{

if(prec==NULL){

(\*head1)=(\*head1)->next;

free(corr);

corr=(\*head1);

}else{

prec->next=corr->next;

free(corr);

corr=prec->next;

}

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL, head1=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

while(m!=0){

add(&head1, m);

scanf("%d", &m);

}

print(head);

printf("\n");

print(head1);

printf("\n");

elimina(&head, &head1);

//print(head);

print(head1);

printf("\n");

return 0;

}

**/\*ES 4 ESAME 1/09/15.**

-Scrivere una procedura che data una lista di interi, antepone ad ogni elemento negativo 0, se non ci sono elementi negativi, la lista deve rimanere inalterata.

//se sfruttiamo la lista nella modalita’ sbagliata, questa procedura non funziona per il caso in cui il primo elemento della lista sia negativo.

#include<stdio.h>

#include<stdlib.h>

typedef struct Il\_node\_struct{

int info;

struct Il\_node\_struct \*next;

}Il\_node;

Il\_node \*add(Il\_node \*head, int v){

Il\_node \*corr=head;

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=v;

new\_node->next=NULL;

if(head==NULL){

new\_node->next=head;

head=new\_node;

}

else if(head!=NULL){

while(corr->next!=NULL){

corr=corr->next;

}

corr->next=new\_node;

}

return head;

}

//questa procedura non funziona per il primo elemento se negativo

void zero(Il\_node \*head){

Il\_node \*prec, \*corr;

if(head!=NULL){ //dobbiamo assicurarci che la lista non sia vuota

if(head->info<0){ //per il primo elemento della lista negativo

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=0;

new\_node->next=head; //questo non puo' essere fatto con l'ordine inverso con quello che segue

head=new\_node;

prec=head->next; //avanzo dal primo elemento che ho aggiunto essere zero

corr=prec->next;

}

else if(head->info>0){ //per il primo elemento della lista positivo

prec=head; //avanzo dal primo elemento della lista

corr=prec->next;

}

while(corr!=NULL){

if(corr->info<0){

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=0;

new\_node->next=corr;

prec->next=new\_node;

prec=corr; //dopo aver aggiunto il nodo zero avanzo con i puntatori di scorrimento

corr=corr->next;

}

else if(corr->info>0){

prec=corr;

corr=corr->next;

}

}

}

}

int main(){

Il\_node \*head=NULL;

int n, x;

scanf("%d",&n);

while(n!=0){

head=add(head,n);

scanf("%d",&n);

}

zero(head);

print(head);

return 0;

}

//se invece usiamo la modalita’ di definizione corretta della lista, tale procedura funziona.

#include<stdio.h>

#include<stdlib.h>

struct el{

int info;

struct el \*next;

}el;

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL){

corr=corr->next;

}

corr->next=new;

}

}

void zero(Lista \*head){

Lista prec, corr;

if((\*head)!=NULL){ //dobbiamo assicurarci che la lista non sia vuota

if((\*head)->info<0){ //per il primo elemento della lista negativo

Lista new=(Lista)malloc(sizeof(struct el));

new->info=0;

new->next=\*head; //questo non puo' essere fatto con l'ordine inverso con quello che segue

(\*head)=new;

prec=(\*head)->next; //avanzo dal primo elemento che ho aggiunto essere zero

corr=prec->next;

}else if((\*head)->info>0){ //per il primo elemento della lista positivo

prec=\*head; //avanzo dal primo elemento della lista

corr=prec->next;

}

while(corr!=NULL){ //se facessi while(corr->next!=NULL){ mi salterebbe l'ultimo elemento

if(corr->info<0){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=0;

new->next=corr;

prec->next=new;

prec=corr; //dopo aver aggiunto il nodo zero avanzo con i puntatori di scorrimento

corr=corr->next;

}

else if(corr->info>0){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x;

scanf("%d", &n);

while(n!=0){

add(&head,n);

scanf("%d", &n);

}

zero(&head);

print(head);

return 0;

}

//..VARIANTE: Scrivere una procedura che data una lista di interi e un intero x, aggiunge un elemento che contiene x prima di ogni elemento negativo presente nella lista. Se non ci sono elementi negativi, la lista deve rimanere inalterata.

Anche qui, se usiamo il modo sbagliato di definire la lista, la procedura non funziona per il primo elemento se negativo.

#include<stdio.h>

#include<stdlib.h>

typedef struct Il\_node\_struct{

int info;

struct Il\_node\_struct \*next;

}Il\_node;

Il\_node \*add(Il\_node \*head, int v){

Il\_node \*corr=head;

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=v;

new\_node->next=NULL;

if(head==NULL){

new\_node->next=head;

head=new\_node;

}

else if(head!=NULL){

while(corr->next!=NULL){

corr=corr->next;

}

corr->next=new\_node;

}

return head;

}

void variante(Il\_node \*head, int x){

Il\_node \*prec, \*corr;

if(head!=NULL){ //dobbiamo assicurarci che la lista non sia vuota

if(head->info<0){ //per il primo elemento della lista negativo

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=x;

new\_node->next=head; //questo non puo' essere fatto con l'ordine inverso con quello che segue

head=new\_node;

prec=head->next; //avanzo dal primo elemento che ho aggiunto essere zero

corr=prec->next;

}

else if(head->info>0){ //per il primo elemento della lista positivo

prec=head; //avanzo dal primo elemento della lista

corr=prec->next;

}

while(corr!=NULL){

if(corr->info<0){

Il\_node \*new\_node=malloc(sizeof(Il\_node));

new\_node->info=x;

new\_node->next=corr;

prec->next=new\_node;

prec=corr; //dopo aver aggiunto il nodo zero avanzo con i puntatori di scorrimento

corr=corr->next;

}

else if(corr->info>0){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Il\_node \*head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Il\_node \*head=NULL;

int n, x;

scanf("%d",&n);

while(n!=0){

head=add(head,n);

scanf("%d",&n);

}

scanf("%d",&x);

variante(head,x);

print(head);

return 0;

}

//questa e’ la versione corretta della variante

#include<stdio.h>

#include<stdlib.h>

struct el{

int info;

struct el \*next;

}el;

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void variante(Lista \*head, int x){

Lista prec, corr;

if((\*head)!=NULL){ //dobbiamo assicurarci che la lista non sia vuota

if((\*head)->info<0){ //per il primo elemento della lista negativo

Lista new=(Lista)malloc(sizeof(struct el));

new->info=x;

new->next=\*head; //questo non puo' essere fatto con l'ordine inverso con quello che segue

\*head=new;

prec=(\*head)->next; //avanzo dal primo elemento che ho aggiunto essere zero

corr=prec->next;

}else if((\*head)->info>0){ //per il primo elemento della lista positivo

prec=\*head; //avanzo dal primo elemento della lista

corr=prec->next;

}

while(corr!=NULL){

if(corr->info<0){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=x;

new->next=corr;

prec->next=new;

prec=corr; //dopo aver aggiunto il nodo zero avanzo con i puntatori di scorrimento

corr=corr->next;

}else if(corr->info>0){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

variante(&head, x);

print(head);

return 0;

}

//..VARIANTE2: Scrivere una procedura che data una lista di interi, un intero x e un intero y, aggiunge un elemento che contiene y prima di ogni elemento uguale ad x presente nella lista. Se non ci sono elementi uguali ad x, la lista deve rimanere inalterata.

#include<stdio.h>

#include<stdlib.h>

struct el{

int info;

struct el \*next;

}el;

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void variante(Lista \*head, int x, int y){

Lista prec, corr;

if((\*head)!=NULL){ //dobbiamo assicurarci che la lista non sia vuota

if((\*head)->info==x){ //per il primo elemento della lista negativo

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

new->next=\*head; //questo non puo' essere fatto con l'ordine inverso con quello che segue

\*head=new;

prec=(\*head)->next; //avanzo dal primo elemento che ho aggiunto essere zero

corr=prec->next;

}else if((\*head)->info!=x){ //per il primo elemento della lista positivo

prec=\*head; //avanzo dal primo elemento della lista

corr=prec->next;

}

while(corr!=NULL){

if(corr->info==x){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

new->next=corr;

prec->next=new;

prec=corr; //dopo aver aggiunto il nodo zero avanzo con i puntatori di scorrimento

corr=corr->next;

}else if(corr->info!=x){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x, y;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

scanf("%d", &y);

variante(&head, x,y);

print(head);

return 0;

}

//se vogliamo invece inserire un solo elemento prima/dopo un dato elemento

#include<stdio.h>

#include<stdlib.h>

struct el{

int info;

struct el \*next;

}el;

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void variante(Lista \*head, int x, int y){

//cosi inseriamo y prima di x, che non sia il primo, non funziona cioe' su una lista contenente solo x

Lista corr=\*head;

int trovato=0;

//i puntatori di attacco sono gli stessi ma cambia dove ci fermiamo e chi abbiamo controllato, per inserire prima controlliamo se l'elemento seguente e' x, e viceversa, per inserire dopo controlliamo se l'elemento corrente e' x

//per inserire y subito prima di x

/\* while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->next->info==x) trovato=1;

else corr=corr->next;

}

if(trovato==1){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

new->next=corr->next; //e' importante quest'ordine

corr->next=new;

} \*/

/\*//se invece vogliamo che funzioni anche su una lista contenente solo x oppure come primo elemento x abbiamo bisogno di 2 puntatori

Lista corr=\*head, prec=NULL;

int trovato=0;

while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->info==x) trovato=1;

else{

prec=corr;

corr=corr->next;

}

}

if(trovato==1){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

// if(prec==NULL){

// new->next=(\*head);

// (\*head)=new;

// }else if(prec!=NULL){

// new->next=prec->next;

// prec->next=new;

// }

// //oppure analogamente

new->next=corr;

if(prec!=NULL) prec->next=new;

else (\*head)=new;

} \*/

//per inserire y subito dopo di x, non funziona se x e' l'ultimo elemento

while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->info==x) trovato=1;

else corr=corr->next;

}

if(trovato==1){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

new->next=corr->next; //e' importante quest'ordine

corr->next=new;

}

//se x non e' presente, oppure se x e' l'ultimo della lista inseriamo la y in fondo alla lista, altrimenti cosi come e' rimane inalterata

else if(trovato==0){

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

corr->next=new;

new->next=NULL; //e' importante quest'ordine

}

}

//cosi inseriamo y subito dopo l'ultima occorrenza di x

int var(Lista head, int x){

int conta=0;

while(head!=NULL){

if(head->info==x)

conta++;

head=head->next;

}

return conta;

}

void varr(Lista \*head, int x, int y){

int v=0;

Lista corr=\*head;

while(v<var((\*head), x) && corr->next!=NULL){

if(corr->next->info==x) v++;

corr=corr->next;

}

Lista new=(Lista)malloc(sizeof(struct el));

new->info=y;

new->next=corr->next;

corr->next=new;

}

void print(Lista head){

while(head!=NULL){

printf("%d\n",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, x, y;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &x);

scanf("%d", &y);

variante(&head, x, y);

// varr(&head, x, y);

print(head);

return 0;

}

**/\*ES 5 ESAME 16/12/15:**

Scrivere una procedura che, data in ingresso una lista di interi, porta in ultima posizione il primo elemento negativo che compare nella lista. La procedura lascia la lista inalterata se in essa non occorre alcun elemento negativo. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void spostaneg(Lista \*head){

if((\*head)!=NULL){

Lista prec=NULL, corr=\*head, prev=\*head;

int trovato=0;

while(corr!=NULL && corr->next!=NULL && trovato==0){

if(corr->info<0) trovato=1;

else if(corr->info>=0){

prec=corr;

corr=corr->next;

}

}

if(trovato==1){

while(prev->next!=NULL){

prev=prev->next;

}

if(prec==NULL){

(\*head)=(\*head)->next;

prev->next=corr;

corr->next=NULL;

}else if(prec!=NULL){

prec->next=corr->next;

prev->next=corr;

corr->next=NULL;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

spostaneg(&head);

print(head);

return 0;

}

**/\*ES 3 ESAME 14/01/16:**

Scrivere una procedura che, prese due liste di interi, cancella dalla prima lista tutti gli elementi che compaiono anche nella seconda.

(Supponiamo che non debbano essere cancellati tutti quanti gli elementi della prima lista, altrimenti se vuota da' segmentation fault. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int contamembro(Lista head, int val){

int count=0;

while(head!=NULL){

if(head->info==val) count++;

head=head->next;

}

return count;

}

void cancella(Lista \*head, Lista \*head1){

if((\*head)!=NULL || (\*head1)!=NULL){

Lista corr=\*head, corr1=\*head1, prec=NULL;

int v;

while(corr1!=NULL){ //se mettessi while(corr1->next!=NULL){.. non controllerebbe mai l'ultimo elemento di \*head1

// printf("corrr %d ", corr1->info);

// printf("count %d ", contamembro(\*head, corr1->info));

if(contamembro(\*head, corr1->info)>0){

v=0, corr=\*head, prec=NULL; //questi devono essere sempre inizializzati cosi per ogni nuovo elemento

while(v<=contamembro(\*head, corr1->info)){

if(corr->info==corr1->info){

if(prec==NULL){

Lista prev=\*head;

(\*head)=(\*head)->next;

free(prev);

}else if(prec!=NULL){

Lista prev=corr;

prec->next=prec->next->next;

corr=prec->next;

free(prev);

}

v++;

}

prec=corr;

corr=corr->next;

}

// printf("%+d ",prec->info);

// printf("%+d ",corr->info);

//per l'ultimo corr che non controlla il while perche' esce dal ciclo visto che v ha raggiunto il limite

if(corr->info==corr1->info){

if(prec!=NULL){

prec->next=NULL;

free(corr);

}else if(prec==NULL) (\*head)=NULL;

}

}

corr1=corr1->next;

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL, head1=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

while(m!=0){

add(&head1, m);

scanf("%d", &m);

}

print(head);

printf("\n");

print(head1);

printf("\n");

cancella(&head, &head1);

print(head);

printf("\n");

return 0;

}

**/\*ES 3 ESAME 4/02/16:**

Scrivere una procedura che, dati attraverso opportuni parametri una lista di interi ed un intero n, sposta in testa alla lista tutti gli elementi maggiori di n. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void sposta(Lista \*head, int n){

if((\*head)!=NULL){

if((\*head)->next!=NULL){

Lista prec=\*head, corr=(\*head)->next;

while(corr!=NULL){

if(corr->info>n){

prec->next=corr->next;

corr->next=\*head;

(\*head)=corr;

corr=prec->next;

}else if(corr->info<=n){

prec=corr;

corr=corr->next;

}

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d\n", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

sposta(&head, m);

print(head);

return 0;

}

**/\*ES 4 ESAME 8/06/16:**

Scrivere una procedura che, prese due liste di interi cancella dalla prima tutti gli elementi che non compaiono nella seconda. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int member(Lista head1, int val){

int trovato=0;

while(head1!=NULL && trovato==0){

if(head1->info==val) trovato=1;

else if(head1->info!=val) head1=head1->next;

}

return trovato;

}

void canc(Lista \*head, Lista \*head1){

if((\*head)!=NULL || (\*head1)!=NULL){

Lista prec=NULL, corr=\*head;

while(corr!=NULL){

if(member(\*head1, corr->info)==0){

if(prec==NULL){

(\*head)=(\*head)->next;

free(corr);

corr=(\*head);

}else if(prec!=NULL){

prec->next=corr->next;

free(corr);

corr=prec->next;

}

}else if(member(\*head1, corr->info)==1){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL, head1=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

while(m!=0){

add(&head1, m);

scanf("%d", &m);

}

print(head);

printf("\n");

print(head1);

printf("\n");

canc(&head, &head1);

print(head);

printf("\n");

return 0;

}

**/\*ES 4 ESAME 6/07/16 opp ES 4 ESAME 3/06/15:**

Scrivere una procedura che, data una lista di interi a ed un intero n, cancella dalla lista i primi n elementi maggiori di 0. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int val){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=val;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void canc(Lista \*head, int n){

if((\*head)!=NULL){

Lista prec=NULL, corr=\*head;

while(corr!=NULL && n>0){

if(corr->info>0){

if(prec==NULL){

(\*head)=(\*head)->next;

free(corr);

corr=(\*head);

}else if(prec!=NULL){

prec->next=corr->next;

free(corr);

corr=prec->next;

}

n--;

}else if(corr->info<0){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=-100){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

canc(&head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 4 ESAME 06/07/16:**

Si scriva una procedura che, presi attraverso opportuni parametri una lista di interi ed un intero n, cancella dalla lista i primi n elementi maggiori di 0. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int v){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=v;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

void delete(Lista \*head, int m){

if((\*head)!=NULL && m>0){

int v=0;

Lista corr=\*head, prec=NULL, prev=corr;

while(corr!=NULL && v<m){

if(corr->info>0){

if(prec==NULL){

if(corr->next==NULL) (\*head)=NULL;

else if(corr->next!=NULL){

(\*head)=(\*head)->next;

prev=corr; //per non creare garbage e errori dobbiamo liberare lo spazio

free(prev);

corr=prev->next;

//free(corr); //oppure

//corr=(\*head);

}

}else if(prec!=NULL){

if(corr->next==NULL){

prec->next=NULL;

prev=corr;

free(prev);

corr=prev->next;

}else if(corr->next!=NULL){

prec->next=prec->next->next;

prev=corr;

free(prev);

corr=prev->next;

}

}

v++;

}else if(corr->info<0){

prec=corr;

corr=corr->next;

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

delete(&head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 4 ESAME 06/09/16:**

Scrivere una procedura che, presa attraverso un opportuno parametro una lista di almeno 3 elementi, scambia le posizioni del primo e del penultimo elemento, senza utilizzare assegnamenti sui campi info delle strutture che compongono la lista. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int e){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=e;

new->next=NULL;

if((\*head)==NULL){

new->next=\*head;

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

/\*

void scambia(Lista \*head){

if((\*head)!=NULL && (\*head)->next!=NULL && (\*head)->next->next!=NULL){

Lista primo=\*head, corr=\*head, prec=NULL;

while(corr->next->next!=NULL){

prec=corr;

corr=corr->next;

}

if(prec==(\*head)){

primo=corr;

(\*head)->next=corr->next;

corr->next=(\*head);

(\*head)=primo;

}

else if(prec!=(\*head)){

//mossa poco corretta

int a=(\*head)->info, b=corr->info;

(\*head)->info=b;

corr->info=a;

}

}

} \*/

void scambia(Lista \*head){

if((\*head)!=NULL && (\*head)->next!=NULL && (\*head)->next->next!=NULL){

Lista terzultimo=\*head, penultimo=(\*head)->next, ultimo=(\*head)->next->next;

while(ultimo->next!=NULL){

terzultimo=penultimo;

penultimo=ultimo;

ultimo=ultimo->next;

}

// printf("terzultimo%+d",terzultimo->info);

// printf("penultimo%+d",penultimo->info);

// printf("ultimo%+d",ultimo->info);

terzultimo->next=(\*head);

penultimo->next=(\*head)->next;

(\*head)->next=ultimo;

(\*head)=penultimo;

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int i;

scanf("%d", &i);

while(i!=0){

add(&head, i);

scanf("%d", &i);

}

print(head);

printf("\n");

scambia(&head);

print(head);

printf("\n");

return 0;

}

**/\*ES 3 ESAME 19/01/17:**

Si scriva una procedura che, presi una lista e un numero naturale n, crea una copia dei primi n elementi della lista e aggiunge tali nuovi elementi in coda alla lista stessa. Se la lista contiene meno di n elementi, tutti gli elementi vengono copiati e aggiunti in coda. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int m){

Lista corr=\*head, new=(Lista)malloc(sizeof(struct el));

new->info=m;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int lunghezza(Lista head){

int count=0;

while(head!=NULL){

count++;

head=head->next;

}

return count;

}

void copia(Lista head, int m){ //il parametro head non va passato per indirizzo perche' non si modifica mai il puntatore head

if((head)!=NULL){

Lista corr=head;

int i=0;

if(m<lunghezza(head)){

while(i<m){

add(&(head), corr->info);

i++;

corr=corr->next;

}

}

else if(m>=lunghezza(head)){

m=lunghezza(head);

while(i<=m-1){ //non posso mettere qui i<lunghezza(head) perche' quest'ultima cambia ogni volta

add(&(head), corr->info);

i++;

corr=corr->next;

}

}

}

}

//Una possibile soluzione consiste nel copiare un elemento per volta, andando via via ad aggiungere la copia in fondo alla lista.

void copia1(Lista l, int n){

Lista tail=l, oldtail, tmp;

if(l!=NULL && n>0){

while(tail->next!=NULL)

tail=tail->next;

oldtail=tail;

while(n>0 && l!=oldtail->next){

tmp=(Lista)malloc(sizeof(struct el));

tmp->info=l->info;

tmp->next=NULL;

tail->next=tmp;

tail=tail->next;

n--;

l=l->next;

}

}

}

//Una soluzione alternativa consiste nel creare una nuova lista costituita dalle copie dei primi n elementi, da “appendere” in un colpo solo in coda alla lista originaria.

void copia2(Lista l, int n){

Lista copiati=NULL, last\_copiati=NULL, tmp, curr=l;

if(l!=NULL){

while(n>0 && curr!=NULL){

tmp=(Lista)malloc(sizeof(struct el));

tmp->info=curr->info;

tmp->next=NULL;

if(copiati==NULL){

copiati=tmp;

last\_copiati=tmp;

}else{

last\_copiati->next=tmp;

last\_copiati=tmp;

}

n--;

curr=curr->next;

}

while(l->next!=NULL)

l=l->next;

l->next=copiati;

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ",head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

// copia(head, m);

// copia1(head, m);

copia2(head, m);

print(head);

printf("\n");

return 0;

}

**/\*ES 3 ESAME 09/02/17:**

Scrivere una procedura che, presi una lista e un numero naturale n, elimina gli ultimi n elementi della lista e inserisce in coda (al loro posto) un nuovo elemento il cui valore e' pari alla somma dei valori contenuti negli elementi rimossi. Se la lista contiene meno di n elementi, l'operazione viene eseguita usando tutti gli elementi contenuti nella lista. Se nessun elemento viene rimosso (perche' n=0 o perche' la lista e' vuota) si dovra' comunque inserire in coda un nuovo elemento contenente il valore 0. \*/

#include<stdio.h>

#include<stdlib.h>

struct el{int info; struct el \*next;};

typedef struct el \*Lista;

void add(Lista \*head, int x){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head;

new->info=x;

new->next=NULL;

if((\*head)==NULL){

new->next=(\*head);

(\*head)=new;

}else if((\*head)!=NULL){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}

}

int length(Lista head){

int conta=0;

while(head!=NULL){

conta++;

head=head->next;

}

return conta;

}

void delete(Lista \*head, int n){

Lista new=(Lista)malloc(sizeof(struct el)), corr=\*head, prev=NULL, corrcopia=NULL;

new->info=0;

new->next=NULL;

if((\*head)==NULL) (\*head)=new;

else if((\*head)!=NULL){

if(n==0){

while(corr->next!=NULL)

corr=corr->next;

corr->next=new;

}else if(n!=0){

if(n>=length(\*head)){

while(corr!=NULL){

new->info=new->info+corr->info;

corr=corr->next;

}

free(\*head);

(\*head)=new;

}else if(n<length(\*head)){

int diff=length(\*head)-n, i=0;

while(i<diff){

prev=corr;

corr=corr->next;

i++;

}corrcopia=corr;

while(corr!=NULL){

new->info=new->info+corr->info;

corr=corr->next;

}prev->next=new;

free(corrcopia);

}

}

}

}

void print(Lista head){

while(head!=NULL){

printf("%d ", head->info);

head=head->next;

}

}

int main(){

Lista head=NULL;

int n, m;

scanf("%d", &n);

while(n!=0){

add(&head, n);

scanf("%d", &n);

}

scanf("%d", &m);

print(head);

printf("\n");

delete(&head, m);

print(head);

printf("\n");

return 0;

}