1 Esercizi

1.1 Esempio Prima libreria

1.1.1 main

```c
//ESEMPIO DI SVILUPPO DI UN PROGRAMMA IN PIU' FILE: File di dicharazione funzioni,ed
//eventualmente #define (header file), e file di definizione di funzioni
// Qui e' riportato un esempio semplice costituito da un unico header file ed un unico
//file di definizione di funzioni:
// Il file header e' "matriciDinamiche.h",
// Il file di definizione delle funzioni e': libMatriciDinamiche.c, di cui deve essere creato
// l'object file libMatriciDinamiche.o tramite il comando gcc -c libMatriciDinamiche.c
//COMPILARE CON: gcc esempioPropriaLibreria.c libMatriciDinamiche.o -o esempioPropriaLibreria
//PER ULTERIORI APPROFONDIMENTI VEDERE: https://www.cs.dartmouth.edu/~campbell/cs50/buildlib.html

#include <stdio.h>
#include "matriciDinamiche.h"
int main(){
       int r,c;
       int **M1,**M2,*M3;
       printf("Dammi r e c:");
       scanf("%d %d",&r,&c);
       M1=creaMatriceDinamica1(r,c);
       M2=creaMatriceDinamica2(r,c);
       M3=creaMatriceDinamica3(r,c);
       //ALTRO CODICE
       liberaMatriceDinamica1(M1,r);
       liberaMatriceDinamica2(M2);
       liberaMatriceDinamica3(M3);
   return 0;
}
```

1.1.2 lib

```c
#include <stdlib.h>
#include "matriciDinamiche.h"

int **creaMatriceDinamica1(int r,int c){
       int i;
       int **M;
       M= (int **) malloc(r*sizeof(int *));
       for (i=0;i<r;++i) M[i]= (int *)malloc(c*sizeof(int));
       // In maniera equivalente, con l'unica differenza che inizializzo a zero
       // tutti gli elementi:
       //for (i=0;i<r;++i) A[i]= (int *)calloc(c,sizeof(int));
```

```
        return M;
}

int **creaMatriceDinamica2(int r,int c){
        int i;
        int *V= (int *)malloc(r*c*sizeof(int));
        // In maniera equivalente, con l'unica differenza che inizializzo a zero
        // tutti gli elementi:
        // int *V= (int *)calloc(r*c,sizeof(int));
        int **M=(int **) malloc(r*sizeof(int *));
        for (i=0;i<r;++i) M[i]= &V[i*c];
        return M;
}
int *creaMatriceDinamica3(int r,int c){
        int i;
        int *M= (int *)malloc(r*c*sizeof(int));
        // In maniera equivalente, con l'unica differenza che inizializzo a zero
        // tutti gli elementi:
        // int *M= (int *)calloc(r*c,sizeof(int));
        return M;
}

void liberaMatriceDinamica1(int **M, int r){
        int i;
        for (i=0;i<r;++i) free(M[i]);
        free(M);
}

void liberaMatriceDinamica2(int **M){
        int i;
        free(M[0]);
        free(M);
}
void liberaMatriceDinamica3(int *M){
        free(M);
}
```

1.1.3 header

```
//FILE HEADER
int **creaMatriceDinamica1(int r,int c);
int **creaMatriceDinamica2(int r,int c);
int *creaMatriceDinamica3(int r,int c);
//Liberare memoria per una matrice dinamica (tre modi diversi, uno per ogni tipo di matrice)
void liberaMatriceDinamica1(int **M, int r);
```

```
void liberaMatriceDinamica2(int **M);
void liberaMatriceDinamica3(int *M);
```

1.2 Esercizio 1

```
#include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
                     di questo file*/
/*Versione Procedurale di tipo top-down*/
struct risAndRes {
	int ris;
	int res;
	float pippo;
}; //Ricordo che sto definendo la struttura ma NON sto creando alcuna variabile!!!!
int leggiNumero();
struct risAndRes estraiPrimaCifra(int);
void stampa(int,int);
int main() {
	int x=0,c=0;
	struct risAndRes ss; //Creo la variabile ss di tipo "struct risAndRes"
	x = leggiNumero();
	do {
		ss= estraiPrimaCifra(x);
		stampa(c,ss.res);
		x=ss.ris;
		++c;
	} while(x!=0);
return 0;
}

int leggiNumero(){
	int x;

	printf("**********************\n");
	printf("Dammi un numero intero:");
	scanf("%d",&x);

	return x;
}
struct risAndRes estraiPrimaCifra(int x){
	struct risAndRes tmp;

	tmp.ris=x/10;
	tmp.res=x%10;

	return tmp;
```

```
}

void stampa(int conta, int cifra){

	printf("cifra corrispondente a 10 elevoto a%d =%d\n",conta,cifra);
}
```

1.3 Esercizio 2

```
#include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
                    di questo file*/
/*Versione Procedurale di tipo top-down*/
struct risAndRes {
	int ris;
	int res;
	float pippo;
}; //Ricordo che sto definendo la struttura ma NON sto creando alcuna variabile!!!!
int leggiNumero();
int leggiCifra();
int verificaCifra(int,int);
void stampa(int);
int main() {
	int x=0,cifra, presente=0;
	struct risAndRes ss; //Creo la variabile ss di tipo "struct risAndRes"
	x = leggiNumero();
	cifra=leggiCifra();
	presente=verificaCifra(x,cifra);
	stampa(presente);
return 0;
}
int leggiCifra(){
	int x;
	printf("Dammi la cifra da trovare:");
	scanf("%d",&x);
	return x;
}

int leggiNumero(){
	int x;

	printf("**********************\n");
	printf("Dammi un numero intero:");
	scanf("%d",&x);

	return x;
```

```c
}
struct risAndRes estraiPrimaCifra(int x){
	struct risAndRes tmp;

	tmp.ris=x/10;
	tmp.res=x%10;

	return tmp;
}

int verificaCifra(int x,int cifra){
	int presente=0;
	struct risAndRes ss;
	do {
		ss= estraiPrimaCifra(x);
		if (cifra== ss.res) presente=1;
		x=ss.ris;
	} while(x!=0 && presente==0);

	return presente;
}
void stampa(int presente){
	if (presente==1)
		printf("La cifra e' presente");
		else
		printf("La cifra non e' presente");
}
```

1.4 Esercizio 3

```c
#include <stdio.h>

int verificaCodizione(int,int,int);
void stampa(int);
int main(){
    int verifica=0;
    int a,b,c;
    printf("Dammi tre interi positivi:");
    scanf("%d %d %d",&a,&b,&c);
    verifica=verificaCodizione(a,b,c);
    stampa(verifica);
return 0;
}

int verificaCodizione(int a,int b,int c){
```

```
    int ver=0;
    if (a*a == (b*b + c*c)) ver=1;
    else if (b*b== (a*a+c*c)) ver=1;
    else if (c*c == (b*b+a*a)) ver=1;
    return ver;
}
void stampa(int verifica){
    if (verifica==1) printf("E' un triangolo rettangolo!\n");
    else printf("Non e' un triangolo rettangolo!\n");
}
```

1.5 Esercizio 4

```
#include <stdio.h>
//Il fattoriale di un numero intero n non negativo, si scrive n! e si legge "n "fattoriale ed è
    definito come: n*(n-1)*...*1 quando n è maggiore o uguale ad 1, mentre se n è uguale a 0 è pari
    a 1. Scrivere un programma che legga un numero intero non negativo e stampi il suo fattoriale.

int riceviInput();
int fattoriale(int);
int main() {
       int n=0,res=0;
       //FASE IN CUI PRENDO LE INFORMAZIONI
       //printf("Dammi il valore di n:");
       //scanf("%d",&n);
       n=riceviInput();
       //FASE IN CUI RISOLVO IL PROBLEMA
       res=fattoriale(n);

       //FASE IN CUI DO LA SOLUZIONE ALL'ESTERNO
       printf("Il fattoriale di n=%d e' = %d\n",n,res);
       return 0;
}
int riceviInput(){
       int n;
       do {
         printf("Dammi il valore di n maggiore o uguale a 0:");
         scanf("%d",&n);
       } while (n<0);
       return n;
}
//PRE-CONDIZIONE: n>=0
int fattoriale(int n){
       int fatt=1;

```

```
        for (fatt=1; n>1;n=n-1) fatt=n*fatt;
        //if (n>0) fatt=n*fattoriale(n-1);
        return fatt;
}
```

1.6 Esercizio 5

```
//Scrivere un programma che riceva in input una stringa e stampi a video
//la lunghezza della stringa.
#include <stdio.h>
#define MAXLEN 100
int lenStr(char str[]);

int main() {
        char s[MAXLEN];

        int len;
        printf("Dammi una stringa:");
        scanf("%s",s);
        len=lenStr(s);
        printf("Lunghezza stringa:%d\n",len);
return 0;
}

//PRECONDIZIONE: str deve essere una stringa, cioe' array di caratteri
//con carattere di fine stringhe
int lenStr(char str[]){
        int len=0;
        while (str[len]!='\0') len=len+1;
        return len;
}
```

1.7 Esercizio 6

```
/*
Scrivere un programma che riceva in input due stringhe s1 e s2 e stampi a video se la stringa s1 è
    una sottostringa di s2 sono oppure no. Ad esempio se s1"="ala e s2"="calamaro allora s1 è una
    sottostringa di s2. Se s1"="ala e s2= ""calma, allora s1 non è una sottoscritta di s2.
*/
#include <stdio.h>
#define MAXSTR 100 //Lunghezza massima di una stringa

//DICHIARO LE FUNZIONI CHE MI SERVONO NEL MAIN (PROTOTIPI)
void leggiStringhe(char s1[],char s2[]); //Leggo i dati
int verificaSottostringa(char s1[],char s2[]); //Operazioni da svolgere
```

```c
void stampaRisultato(int ris,char s1[],char s2[]); //Stampo il risultato
int main(){
	char s1[MAXSTR], s2[MAXSTR]; //STUTTURE DATI DOVE MEMORIZZARE I DATI
	int ris=0;
	printf("PARAMETRO ATTUALE: Indirizzo di memoria di s1=%p e suo valore=%p\n", &s1,s1);
	leggiStringhe(s1,s2); //Leggo i dati
	ris=verificaSottostringa(s1,s2); //Operazioni da svolgere
	printf("Indirizzo memoria di ris del main (parametro attuale):%p\n",&ris);
	stampaRisultato(ris,s1,s2); //Stampo il risultato
return 0;
}

void leggiStringhe(char s1[],char s2[]){
	printf("Dammi la prima stringa:");
	scanf("%s",s1); //scanf("%s",&s1[0]);
	printf("Dammi la seconda stringa:");
	scanf("%s",s2); //scanf("%s",&s1[0]);
}

//VERIFICO SE E' UNA SOTTOSTRINGA A PARTIRE DALLA CONDIZIONE CHE IL CARATTERE
//PRECEDENTE E' UGUALE
int subStr(char s1[],char s2[],int c1,int c2){
	//s1="ala" s2="camalaro"
	int ris=0;

	//CONFRONO SE IN SEQUENZA I CARATTERI SONO TUTTI UGUALI
	while ((s1[c1]!='\0' && s2[c2]!='\0') && (s2[c2]==s1[c1])) {
			++c1;
			++c2;
	}
	//S1 e' una sottostringa se sono arrivato alla fine di S1
	if (s1[c1]=='\0') ris=1;

	return ris;
}

int verificaSottostringa(char s1[],char s2[]){
	int ris=0;
	int c1=0,c2=0;
	while ( ris==0 && s2[c2]!='\0'){
		while (s2[c2]!='\0' && s1[c1]!=s2[c2]) ++c2;
		if (s2[c2]!='\0') ris= subStr(s1,s2,c1+1,c2+1);
		++c2;
	}
	return ris;
}
```

```c

void stampaRisultato(int ris,char s1[],char s2[]){
	printf("La stringa: %s\n",s1);
	if (ris==1) printf("E' sottostringa della\n");
	else printf("NON e' sottostringa della\n");
	printf("stringa: %s\n",s2);
}
```

1.8 Esercizio 7

```c
/*Scrivere un programma che memorizzi in un array n numeri interi. Dopodichè
calcoli il valore medio ed elimini 'dallarray tutti gli elementi minori del valore
medio. Stampi a video 'larray così ottenuto.*/
#include <stdio.h>
#define MAXLEN 100

int leggiNumeri(int a[]);
float valoreMedia(int a[], int n);
int eliminaElementi(int a[],int n, int m);
void stampaElementi(int a[], int n);

int main(){
int a[MAXLEN];
int n;
float m;
n=leggiNumeri(a);
//n rappresenta il numero effettivo di elementi inseriti (presenti nell'array)
//MAXLEN e' la capacità massima dell'array.
m=valoreMedia(a, n);
n= eliminaElementi(a,n,m);
stampaElementi(a, n);

return 0;
}

int leggiNumeri(int a[]){
	int flag=0, n=0;
	do {
		printf("vuoi inserire un altro numero? (SI=1,NO=0)");
		scanf("%d",&flag);
		//n < MAXLEN && flag==1
		if (flag==1 && n < MAXLEN) {
			printf("Inserisci numero:");
			scanf("%d",&a[n]);
```

```
                        ++n;
                }
        } while (flag==1);
return n;
}

float valoreMedia(int a[], int n){
        float m=0;
        int i;
        for (i=0;i<n;++i) m += a[i];
        m=m/n;
        return m;
}
void shiftLeft(int a[],int n,int i);
int eliminaElementi(int a[],int n, int m){
        int i=0;
        while (i<n) {
                if (a[i]<m) {
                        shiftLeft(a,n,i);
                        n=n-1; // --n;
                }
                else ++i;
        }
        return n;
}

void shiftLeft(int a[],int n,int i){
        int j;
        for (j=i; j < n-1;++j)
                a[j]=a[j+1];
}

void stampaElementi(int a[], int n){
        int i;
        for (i=0;i<n;++i)
                printf("a[%d]=%d\n",i,a[i]);
}
```

1.9 Esercizio 8

```
/*
Scrivere un programma che memorizzi in un array n numeri interi. Dopodichè
verifichi se 'lelemento con indice i risulti sempre uguale alla somma degli
elementi con indici i-1 e i-2, più k. k è un valore ottenuto da tastiera. Se gli
elementi di indici i-1 e i-2 non esistono si assumono uguale a zero. Esempio:
```

```
k=1 [1 2 4 7 12]
k=3 [3 6 12 21 36]
*/
#include <stdio.h>

#define MAXLEN 100

int leggiNumeri(int a[]);
int verificaCondizione(int a[],int k,int n);
void stampa(int res);

int main(){
	int n,res,k;
	int a[MAXLEN];
	n=leggiNumeri(a);
	printf("Dammi k:");
	scanf("%d",&k);
	res=verificaCondizione(a,k,n);
	stampa(res);
return 0;
}

int leggiNumeri(int a[]){
	int flag=0, n=0;
	do {
		printf("vuoi inserire un altro numero? (SI=1,NO=0)");
		scanf("%d",&flag);
		//n < MAXLEN && flag==1
		if (flag==1 && n < MAXLEN) {
			printf("Inserisci numero:");
			scanf("%d",&a[n]);
			++n;
		}
	} while (flag==1);
return n;
}

int verificaCondizione(int a[],int k,int n){
	int res=1,i=0, somma=0;
	while (res==1 && i<n) {
		if (i <1) somma=k;
		else if (i<2) somma=a[0]+k;
			else somma=a[i-1] + a[i-2] +k;

		if (a[i] != somma)
			res=0;
```

```
                else ++i;
        }
        return res;
}
void stampa(int res){
        if (res==1) printf("Condizione verificata!!!\n");
        else printf("Condizione NON verificata!!\n");
}
```

1.10 Esercizio 9

```
/*
Si supponga che in un file di testo siano memorizzati i valori di una matrice di interi di
    dimensione mxn, rispettando il seguente formato: sulla prima riga sono presenti i numeri m ed n,
     sulle successive m righe ci sono gli n numeri di ciascuna riga. Ad esempio:
2 3
12 0 -1
9 -8 5
Si scriva una funzione che legga tale file ed inserisca i valori presenti nel file in una matrice M
    creata dinamicamente. Scrivere un main che: 1) Chieda il nome del file di testo. 2) Legga il
    file, se esiste, e crei opportunamente la matrice M. 3) Stampi a video la matrice M ottenuta.
*/
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 50
int **leggiMatrice(FILE *fp,int *pm,int *pn);
void stampaMatrice(int **M,int m,int n);
void liberaMatrice(int **M, int m);
int main() {
        FILE *fp;
        char nomeFile[MAXLEN];
        int m=0,n=0;
        int **M;
        printf("Dammi il nome del file:");
        scanf("%s",nomeFile);
        fp=fopen(nomeFile,"r");
        if (fp!=NULL) {
                M=leggiMatrice(fp,&m,&n);
                if (M!=NULL) {
                        stampaMatrice(M,m,n);
                        liberaMatrice(M,m);
                }
                fclose(fp);
        }
        else printf("Problemi con il file %s nella apertura in lettura!\n",nomeFile);
```

```
return 0;
}

int **leggiMatrice(FILE *fp,int *pm,int *pn){
        int **M=NULL,m=0,n=0,i,j,k;
        if (fscanf(fp,"%d %d",&m,&n)==2) {
                M = (int **)malloc(m*sizeof(int *)); //alloco array di m puntatori
                for (i=0;i<m;++i)
                        M[i]= (int *)malloc(n*sizeof(int)); //alloco array di n interi
                for (i=0;i<m;++i)
                        for (j=0;j<n;++j){
                                fscanf(fp,"%d",&k);
                                M[i][j]=k;
                                }
                *pm=m;
                *pn=n;
        }
        return M;
}
void stampaMatrice(int **M,int m,int n){
        int i,j;
        for (i=0;i<m;++i) {
                for (j=0;j<n;++j)
                        printf("%d ",M[i][j]);
                printf("\n");
        }
}
void liberaMatrice(int **M, int m){
        int i;
        for (i=0;i<m;++i) free(M[i]);
        free(M);
}
```

1.11 Esercizio 10

```
1 //SCRIVERE IN UN FILE E POI LEGGERE DA TALE FILE
2 //SCRIVO UNA SEQUENZA DI INTERI NEL FILE, UN INTERO su ogni riga.
3 //DOPODICHE' LI LEGGO E LI STAMPO A VIDEO
4 #include <stdio.h>
5 #define MAXLEN 20
6
7 int main(){
8         FILE *fp;
9         int n,i,k;
```

```
        char nomeFile[MAXLEN];
        printf("Dammi il nomme del file:");
        scanf("%s",nomeFile);
        //fp=fopen(nomeFile,"w"); //APRO IN SCRITTURA. Se esiste lo cancello. Se non esiste lo
                                //creo nuovo
        fp=fopen(nomeFile,"a");//APRO IN SCRITTURA. Se esiste non lo cancello.
                                //Se non esiste lo creo nuovo
                                //Scrivo a partire dalla fine del file
        if (fp!=NULL) {
                printf("Quanti interi vuoi scrivere?");
                scanf("%d",&n);
                for (i=0;i<n;++i) {
                        printf("Dammi il valore da scivere su file:");
                        scanf("%d",&k);
                        fprintf(fp,"%d ",k);
                }

                fclose(fp);

                fp=fopen(nomeFile,"r");
                if (fp!=NULL) {
                        while (fscanf(fp,"%d",&k)>0)
                                printf("%d\n",k);
                        fclose(fp);
                }
                else printf("Non sono riuscito ad aprire il file in lettura\n");
        }
        else {
                printf("Problema apertura file in scrittura!\n");
        }
}
```

1.12 Esercizio 11

```
//SCRIVERE IN UN FILE E POI LEGGERE DA TALE FILE
//SCRIVO UNA SEQUENZA DI INTERI NEL FILE, UN INTERO su ogni riga.
//DOPODICHE' LI LEGGO E LI STAMPO A VIDEO
#include <stdio.h>
#define MAXLEN 20

void scriviSuFile(FILE *fp);
void leggiDaFile(FILE *fp);

int main(){
        FILE *fp;
```

```
12          int n,i,k;
13          char nomeFile[MAXLEN];
14          printf("Dammi il nomme del file:");
15          scanf("%s",nomeFile);
16          fp=fopen(nomeFile,"w"); //APRO IN SCRITTURA. Se esiste lo cancello. Se non esiste lo
17                                  //creo nuovo
18  //      fp=fopen(nomeFile,"a");//APRO IN SCRITTURA. Se esiste non lo cancello.
19                                  //Se non esiste lo creo nuovo
20                                  //Scrivo a partire dalla fine del file
21          if (fp!=NULL) {
22                  scriviSuFile(fp);
23                  //rewind(fp);
24                  fprintf(fp," \n %d \n ",100);
25                  fclose(fp);
26
27                  fp=fopen(nomeFile,"r");
28                  if (fp!=NULL) {
29                          leggiDaFile(fp);
30                          fclose(fp);
31                  }
32                  else printf("Non sono riuscito ad aprire il file in lettura\n");
33          }
34          else {
35                  printf("Problema apertura file in scrittura!\n");
36          }
37
38
39
40  }
41  //SUPPONGO CHE IL CONTROLLO SU fp !=NULL sia già stato fatto
42  void scriviSuFile(FILE *fp){
43                  int i,n,k;
44                  printf("Quanti interi vuoi scrivere?");
45                  scanf("%d",&n);
46                  for (i=0;i<n;++i) {
47                          printf("Dammi il valore da scivere su file:");
48                          scanf("%d",&k);
49                          fprintf(fp,"%d ",k);
50                  }
51
52  }
53
54
55  void leggiDaFile(FILE *fp){
56          int k;
57          while (fscanf(fp,"%d",&k)>0)
```

```
            printf("%d\n",k);
}
```

1.13 Esercizio 12

```
#include <stdio.h>
#define MAXLEN 50
int main(){
  char nomeFile[MAXLEN],c;


  FILE *fp; //fp è il puntatore ad una stream, ma lo stream non e' creato
  //Quando creo (apro) lo stream devo specificare se apro in lettura,
  // scrittura o entrambe le cose
  //VEDIAMO PRIMA in lettura:
  printf("Dammi il nome del file da leggere:");
  scanf("%s",nomeFile);
  //fscanf(stdin,"%s",nomeFile);
  fp=fopen(nomeFile,"r");
  if (fp!=NULL) {//Ho trovato il file da leggere e creato lo stream
    //Leggo carattere per carattere
    while (fscanf(fp,"%c",&c)>0) {
      printf("%c",c);
    }
    //ALLA FINE DI TUTTE LE OPERAZIONI, DEVO CHIUDERE LO tream
    fclose(fp);
  }
  else { //NON HO TROVATO IL FILE DA LEGGERE O C'E' STATO UN ERRORE IN
    //APERTURA
    printf("FILE NON TROVATO!!");
  }
  return 0;
}
```

1.14 Esercizio 13

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

struct elem {
       int k;
       struct elem *next;
};

struct elem *insElemInTesta(struct elem *top,int k);
```

```c
struct elem *insElemInCoda(struct elem *top,int k);
struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k);
struct elem *eliminaLista(struct elem *top);
struct elem *cercaElemento(struct elem *top,int k);
//In effetti serve solo il precedente!!
struct elem *eliminaElem(struct elem *top, struct elem *pre);

int main(){
	int k;
	struct elem *top; //NON E' CREATA LA LISTA

	top=NULL; //CREO UNA LISTA NULLA, VUOTA!!!!!

	top=insElemInTesta(top,k); //QUI, CON L'ASSEGNAZIONE, CAMBIA EFFETTIVAMENTE IL TOP
				//  DELLA LISTA
	top=insElemInCoda(top,k); //SE CAMBIA IL TOP, VIENE CAMBIATO QUI!!!

	top=eliminaLista(top);

return 0;
}
//PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
struct elem *eliminaElem(struct elem *top, struct elem *pre){
	struct elem *c;
	if (pre==NULL) {
		c=top;
		top=top->next;
		free(c);
	} else {
		c= pre->next;
		pre->next=c->next;
		free(c);
	}
	return top;
}

struct elem *nuovoElemento(int k){
	struct elem *nuovo= (struct elem *)malloc(sizeof(struct elem));
	nuovo->k=k;
	nuovo->next=NULL;
	return nuovo;
}
struct elem *insElemInTesta(struct elem *top,int k){
	struct elem *nuovo= nuovoElemento(k);
	nuovo->next=top;
	return nuovo;
```

```
}
struct elem *insElemInCoda(struct elem *top,int k){
	struct elem *nuovo= nuovoElemento(k);
	if (top==NULL) return nuovo;
	struct elem *curr=top;
	while (curr->next!=NULL) curr=curr->next; //SEMPRE SICURO CHE curr E' DIVERSO DA NULL
	curr->next=nuovo;
	return top;
}
//pre è l'analogo dell'indice negli array
struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k){
	struct elem *nuovo= nuovoElemento(k);
	if (pre==NULL) {
		nuovo->next=top;
		top=nuovo;
	}
	else {
		nuovo->next=pre->next;
		pre->next=nuovo;
	}
	return top;
}
struct elem *eliminaLista(struct elem *top){
	struct elem *c;
	while (top != NULL){
		c=top;
		top=c->next;
		free(c);
	}
	return top;
}
// --Se il puntatore restituito punta a NULL, l'elemento cercato è in testa (la testa
//  potrebbe essere NULL)
// -- Se il puntatore restituito,p, è diverso da NULL, l'elemento cercato si trova come
//   next, p->next (p->next potrebbe essere NULL, se non trovo l'elemento)
struct elem *cercaElemento(struct elem *top,int k){
	struct elem *pre=NULL, curr=top;
	 //SCORRO LA LISTA SE SONO SU UN ELEMENTO DIVERSO DA NULL E NON HO
	// TROVATO k!!!
	while (curr !=NULL && curr->k!=k) {
		pre=curr;
		curr=curr->next;
	}
	/* COSI' NON VA BENE
	while (curr->k!=k && curr !=NULL ) {
		pre=curr;
```

```
                curr=curr->next;
        }
        */
        return pre;
}
```

1.15 Esercizio 14

```
#include <stdio.h>
#include <stdlib.h>
//SUPPONIAMO DI VOLER RISOLVERE IL SEGUENTE PROBLEMA:
// Eliminare un elenento da una lista semplicemente concatenata di valore k, se esiste.
//VERSIONE 1 di questo problema: la lista non contiene ripetizioni. Ogni valore i è presente
//una sola volta nella lista: top--> 23 --> 18 --> 30 -->NULL
// (E NON COSI': top --> 23 -->18-->23-->8-->NULL (in questo caso 23 si ripete)
struct elem {
        int k;
        struct elem *next;
};

//Funzioni di servizio (precedentemente create)*****
struct elem *insElemInTesta(struct elem *top,int k);

struct elem *insElemInCoda(struct elem *top,int k);

struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k);
struct elem *eliminaLista(struct elem *top);
struct elem *cercaElemento(struct elem *top,int k);
//In effetti serve solo il precedente!!
//PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
struct elem *eliminaElem(struct elem *top, struct elem *pre);
//******************************
struct elem *riempiLista();
struct elem *eliminaKSenzaRipetizioni(struct elem *top,int k);
void stampaLista(struct elem *top);
int main(){
        int k;
        struct elem *top; //NON E' CREATA LA LISTA
        top=riempiLista();
        printf("Dammi l'elemento da cancellare:");
        scanf("%d",&k);
        top=eliminaKSenzaRipetizioni(top,k);
        stampaLista(top);
        top=eliminaLista(top);

```

```
return 0;
}
//PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
struct elem *eliminaElem(struct elem *top, struct elem *pre){
	struct elem *c;
	if (pre==NULL) {
		c=top;
		top=top->next;
		free(c);
	} else {
		c= pre->next;
		pre->next=c->next;
		free(c);
	}
	return top;
}

struct elem *nuovoElemento(int k){
	struct elem *nuovo= (struct elem *)malloc(sizeof(struct elem));
	nuovo->k=k;
	nuovo->next=NULL;
	return nuovo;
}
struct elem *insElemInTesta(struct elem *top,int k){
	struct elem *nuovo= nuovoElemento(k);
	nuovo->next=top;
	return nuovo;
}
struct elem *insElemInCoda(struct elem *top,int k){
	struct elem *nuovo= nuovoElemento(k);
	if (top==NULL) return nuovo;
	struct elem *curr=top;
	while (curr->next!=NULL) curr=curr->next; //SEMPRE SICURO CHE curr E' DIVERSO DA NULL
	curr->next=nuovo;
	return top;
}
//pre è l'analogo dell'indice negli array
struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k){
	struct elem *nuovo= nuovoElemento(k);
	if (pre==NULL) {
		nuovo->next=top;
		top=nuovo;
	}
	else {
		nuovo->next=pre->next;
		pre->next=nuovo;
```

```
	}
	return top;
}
struct elem *eliminaLista(struct elem *top){
	struct elem *c;
	while (top != NULL){
		c=top;
		top=c->next;
		free(c);
	}
	return top;
}
// --Se il puntatore restituito punta a NULL, l'elemento cercato è in testa (la testa
//  potrebbe essere NULL)
// -- Se il puntatore restituito,p, è diverso da NULL, l'elemento cercato si trova come
//   next, p->next (p->next potrebbe essere NULL, se non trovo l'elemento)
struct elem *cercaElemento(struct elem *top,int k){
	struct elem *pre=NULL, *curr=top;
	 //SCORRO LA LISTA SE SONO SU UN ELEMENTO DIVERSO DA NULL E NON HO
	// TROVATO k!!!
	while (curr !=NULL && curr->k!=k) {
		pre=curr;
		curr=curr->next;
	}
	/* COSI' NON VA BENE
	while (curr->k!=k && curr !=NULL ) {
		pre=curr;
		curr=curr->next;
	}
	*/
	return pre;
}
struct elem *riempiLista(){
	struct elem *top=NULL;
	int n,i,k;
	printf("Quanti elementi vuoi inserire?");
	scanf("%d",&n);
	for (i=0;i<n;++i) {
		printf("Dammi l'elemento %d:",i+1);
		scanf("%d",&k);
		top=insElemInTesta(top,k);
	}
	return top;
}
struct elem *eliminaKSenzaRipetizioni(struct elem *top,int k){
	struct elem *pre=NULL,*tmp;
```

```
        if (top!=NULL) {
                pre=cercaElemento(top,k);
                if (pre==NULL) {
                        tmp=top;
                        top=top->next;
                        free(tmp);
                }
                else if (pre->next!=NULL) {
                                //tmp è il nodo da eliminare
                                tmp=pre->next;
                                pre->next=tmp->next;
                                free(tmp);
                        }

        }
        /*VERSIONE CON CHIAMATA DI SOTTOFUNZIONE
        if (top!=NULL) {
                pre=cercaElemento(top,k);
                if (pre==NULL) top=eliminaElem(top,pre);
                else if (pre->next!=NULL) top=eliminaElem(top,pre);
        }
        */
        return top;
}
void stampaLista(struct elem *top){
        while (top!=NULL) {
                printf("%d -->",top->k);
                top=top->next;
        }
        printf("NULL\n");
}
```

1.16 Esercizio 15

```
#include <stdio.h>
#include <stdlib.h>
//SUPPONIAMO DI VOLER RISOLVERE IL SEGUENTE PROBLEMA:
// Eliminare un elenento da una lista semplicemente concatenata di valore k, se esiste.
//VERSIONE 2 di questo problema: la lista ncontiene ripetizioni. Ogni valore i può essere //
// presente più volte.
struct elem {
        int k;
        struct elem *next;
};

```

```c
//Funzioni di servizio (precedentemente create)*****
struct elem *insElemInTesta(struct elem *top,int k);

struct elem *insElemInCoda(struct elem *top,int k);

struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k);
struct elem *eliminaLista(struct elem *top);
struct elem *cercaElemento(struct elem *top,int k);
//In effetti serve solo il precedente!!
//PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
struct elem *eliminaElem(struct elem *top, struct elem *pre);
//******************************
struct elem *riempiLista();
struct elem *eliminaKConRipetizioni(struct elem *top,int k);

//Funzione che richiama altre funzioni già realizzate
struct elem *eliminaKConRipetizioni_2(struct elem *top,int k);
void stampaLista(struct elem *top);
int main(){
	int k;
	struct elem *top; //NON E' CREATA LA LISTA
	top=riempiLista();
	printf("Dammi l'elemento da cancellare:");
	scanf("%d",&k);
	//top=eliminaKConRipetizioni(top,k);
	top=eliminaKConRipetizioni_2(top,k);

	stampaLista(top);
	top=eliminaLista(top);

return 0;
}
//PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
struct elem *eliminaElem(struct elem *top, struct elem *pre){
	struct elem *c;
	if (pre==NULL) {
		c=top;
		top=top->next;
		free(c);
	} else {
		c= pre->next;
		pre->next=c->next;
		free(c);
	}
	return top;
}
```

```c

struct elem *nuovoElemento(int k){
	struct elem *nuovo= (struct elem *)malloc(sizeof(struct elem));
	nuovo->k=k;
	nuovo->next=NULL;
	return nuovo;
}
struct elem *insElemInTesta(struct elem *top,int k){
	struct elem *nuovo= nuovoElemento(k);
	nuovo->next=top;
	return nuovo;
}
struct elem *insElemInCoda(struct elem *top,int k){
	struct elem *nuovo= nuovoElemento(k);
	if (top==NULL) return nuovo;
	struct elem *curr=top;
	while (curr->next!=NULL) curr=curr->next; //SEMPRE SICURO CHE curr E' DIVERSO DA NULL
	curr->next=nuovo;
	return top;
}
//pre è l'analogo dell'indice negli array
struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k){
	struct elem *nuovo= nuovoElemento(k);
	if (pre==NULL) {
		nuovo->next=top;
		top=nuovo;
	}
	else {
		nuovo->next=pre->next;
		pre->next=nuovo;
	}
	return top;
}
struct elem *eliminaLista(struct elem *top){
	struct elem *c;
	while (top != NULL){
		c=top;
		top=c->next;
		free(c);
	}
	return top;
}
// --Se il puntatore restituito punta a NULL, l'elemento cercato è in testa (la testa
//  potrebbe essere NULL)
// -- Se il puntatore restituito,p, è diverso da NULL, l'elemento cercato si trova come
//    next, p->next (p->next potrebbe essere NULL, se non trovo l'elemento)
```

```
struct elem *cercaElemento(struct elem *top,int k){
	struct elem *pre=NULL, *curr=top;
	 //SCORRO LA LISTA SE SONO SU UN ELEMENTO DIVERSO DA NULL E NON HO
	// TROVATO k!!!
	while (curr !=NULL && curr->k!=k) {
		pre=curr;
		curr=curr->next;
	}
	/* COSI' NON VA BENE
	while (curr->k!=k && curr !=NULL ) {
		pre=curr;
		curr=curr->next;
	}
	*/
	return pre;
}
struct elem *riempiLista(){
	struct elem *top=NULL;
	int n,i,k;
	printf("Quanti elementi vuoi inserire?");
	scanf("%d",&n);
	for (i=0;i<n;++i) {
		printf("Dammi l'elemento %d:",i+1);
		scanf("%d",&k);
		top=insElemInTesta(top,k);
	}
	return top;
}
struct elem *eliminaKConRipetizioni(struct elem *top,int k){
	struct elem *pre=NULL, *curr,*tmp;
	curr=top;
	while (curr!=NULL) {
		if (curr->k == k) {
			if (pre==NULL) { //ELEMENTO DA ELIMINARE E' IN TESTA
				tmp=curr;
				curr=curr->next;
				top=curr; //UNICO CASO IN CUI CAMBIA IL TOP DELLA LISTA
				free(tmp);
			}
			else {
				pre->next=curr->next;
				free(curr);
				curr=pre->next;
			}
		}
		else {
```

```
                pre=curr;
                curr=curr->next;
            }
        }
        return top;
}


//Funzione che richiama altre funzioni già realizzate
struct elem *eliminaKConRipetizioni_2(struct elem *top,int k){
        struct elem *pre;
        int flag=1;
        while (top !=NULL && flag==1) {
            pre=cercaElemento(top,k); //QUI RICERCO SEMPRE DA TOP, E' DISPENDIOSO!!!
                                      // COME MODIFICARE???
                                      //ESERCIZIO DA FARE A CASA
            if (pre==NULL)
                top=eliminaElem(top,pre);
            else if (pre->next!=NULL) top=eliminaElem(top,pre);
            else flag=0; //NON TROVO ELEMENTI, ESCO DA WHILE!!!!
        }
        return top;
}

void stampaLista(struct elem *top){
        while (top!=NULL) {
            printf("%d -->",top->k);
            top=top->next;
        }
        printf("NULL\n");
}
```

1.17 Esercizio 16

```
#include <stdio.h>
#define MAXLEN 100

void mergeSort(int A[], int n);


int main() {
        int V[MAXLEN];
        int n,i;
        printf("Quanti numeri:");
        scanf("%d",&n);
```

```
	for (i=0;i<n;++i) {
		printf("Dammi V[%d]=",i);
		scanf("%d",&V[i]);
	}
	mergeSort(V,n);
	for (i=0;i<n;++i) {
		printf("V[%d]=%d\n",i,V[i]);
	}

return 0;
}

void merge(int V[],int centro,int n);

void mergeSort(int V[], int n){
	//CASO BASE n=1, IMPLICITO
	if (n>1) {
		//DIVIDO IL PROBLEMA IN SOTTOPROBLEMI
		int centro=n/2;
		//TROVO LE SOLUZIONI DEI SOTTOPROBLEMI
		mergeSort(V, centro);
		mergeSort(&V[centro], n-centro);
		//COMBINO LE SOLUZIONI TROVATE
		merge(V,centro,n);
	}
}
void merge(int V[],int centro,int n){
	int *A, *B;
	int C[MAXLEN];
	int m1,m2; //m1 e m2 le dimensioni dei due vettori già ordinati
	m1=centro;
	m2=n-centro;
	int i=0,j=0,k=0;
	//I DUE ARRRAY A e B GIA' ORDINATI
	A=&V[0];
	B=&V[centro];

	while (i<m1 && j < m2) {
		if (A[i] < B[j]) {
			C[k]=A[i];
			i=i+1;
		}
		else {
			C[k]=B[j];
			j=j+1;
		}
```

```
                k=k+1;
        }
        //AL TERMINE DEL PRECEDENTE WHILE HO: O i>=m1 e j>=m2, OPPURE SOLO UNO E' maggiore
        //O UGUALE
        while ( i < m1) {
                C[k]=A[i];
                k=k+1;
                i=i+1;
        }

        while ( j < m2) {
                C[k]=B[j];
                k=k+1;
                j=j+1;
        }

        for (i=0;i<n;++i) V[i]=C[i];
}
```

1.18 Esercizio 17

```
/*
 fatttorian di n, con n >=0, che in genere è indicato con n! risulta essere
 n!= 1 se n==0, oppure n!=n * (n-1)! altrimenti
*/

#include <stdio.h>
#define MAXLEN 10
int fatt(int n);
//Funzione fattoriale di debug
int fatt_debug(int n);
int main() {
        int n,res;
        printf("Dammi un numero intero maggiore o uguale a zero:");
        scanf("%d",&n);
        //res=fatt(n);
        res=fatt_debug(n);
        printf("Fattoriale uguale a: %d\n",res);
        return 0;
}

int fatt(int n){
        int res=1;
        int res_piu_piccolo;
        if (n==0) //Passo base: risolvo il problema direttamente e
```

```
                // "facilmente"
                res=1;
        else  //Passo ricorsivo
                {
                //Prendo la soluzione dello stesso problema, ma di
                // dimensioni più piccole, cioè n-1
                res_piu_piccolo=fatt(n-1);
                //res_piu_piccolo è il fattoriale di (n-1), cioè il //
                // prodotto (n-1)*(n-2)*.....*1
                // Risolvo il problema con dimensione n, sulla base
                // (usando) la soluzione del problema più piccolo
                res= n *res_piu_piccolo; //QUesta è la chiave della
                                        // ricorsione!!!!!!!!!!
                //res= n*(n-1)*(n-2)*.....*1, quindi
                //res e' il fattoriale di n
                }
return res;
}

//Altro esempio:

int massimoR(int a[],int n) {
        int res=0;
        int max;
        if (n=1) res=a[0];
        else
        {
                max=massimoR(a,n-1);
                if (a[n-1]>max) res=a[n-1];
                        else res=max;
        }

return res;
}


int fatt_debug(int n){
        int res=1;
        int ferma; //Variabile di appogio
        if (n==0){ //Passo base: risolvo il problema direttamente e
                  // "facilmente"
                res=1;
                printf("CASO BASE n:%d  res:%d \n",n,res);
                printf("Vai avanti?");
                scanf("%d",&ferma);
        }
```

```
        else  //Passo ricorsivo
        {
        printf("CASO ricorsivo prima della chiamata ricorsiva n:%d  res:%d \n",n,res);
        printf("Vai avanti?");
        scanf("%d",&ferma);
        res= n * fatt_debug(n-1);
        printf("CASO ricorsivo dopo la chiamata ricorsiva n:%d  res:%d \n",n,res);
        printf("Vai avanti?");
        scanf("%d",&ferma);

        }
return res;
}
```

1.19 Esercizio 18

```
//1) Voglio eliminare da un lista semplicemente concatenata tutti gli elementi mionori
//di un dato k
//2) Suppongo di avere una lista di interi semplicemente concatenata senza ripetizioni.
// Voglio eleiminare un elemento di valore k, se esiste

#include <stdio.h>
#include <stdlib.h>
struct elem {
        int k;
        struct elem *next;
};

struct elem *riempiLista();
struct elem *cancellaMinoriDik(struct elem *top,int k);
struct elem *cancellaElementok(struct elem *top,int k);

void stampaLista(struct elem *top);
int main(){
        struct elem *top;
        int k;
        top=riempiLista();
        stampaLista(top);
        printf("Dammi k:");
        scanf("%d",&k);
        top=cancellaMinoriDik(top,k);
        stampaLista(top);
return 0;
}
struct elem *riempiLista(){
```

```
30          struct elem *top=NULL;
31          return top;
32  }
33  //P(n)=Eliminare da un lista semplicemente concatenata di dimensione n,
34  // tutti gli elementi mionori di un dato k
35
36  struct elem *cancellaMinoriDik(struct elem *top,int k){
37          //CASO BASE, top==NULL, implicito
38          struct elem *tmp;
39          if (top!=NULL) {
40                  //ASSUMMO DI SAPER RISOLVERE IL PROBLEMA P(n-1), cioè
41                  //P(n)=Eliminare da un lista semplicemente concatenata di dimensione n-1,
42                  // tutti gli elementi mionori di un dato k
43                  //tmp mantiene la soluzione, cioè sarà il puntatore alla testa della
44                  //lista in cui sono stati eliminati gli elementi minori di k,
45                  //ecceto la testa iniziale (top)
46                  tmp=cancellaMinoriDik(top->next,k);
47                  //ADESSO COSTRUISCO LA SOLUZIONE PER P(n)
48                  if (top->k<k) {
49                          free(top);
50                          top=tmp;
51                          }
52                  else top->next=tmp;
53          }
54          return top;
55  }
56  void stampaLista(struct elem *top){
57  }
58
59  //P(n)= Data una lista di interi semplicemente concatenata e senza ripetizioni,
60  // con n elementi, Elimino un elemento di valore k, se esiste
61
62  struct elem *cancellaElementok(struct elem *top,int k){
63          struct elem *tmp;
64          //Passo base, top==NULL, implicito
65
66          if (top!=NULL) {
67                  //Se è in testa devo solo cancellare la testa e non devo fare più nulla,
68                  // perchè è una lista SENZA ripetizioni
69                  if (top->k==k) {
70                          tmp=top->next;
71                          free(top);
72                          top=tmp;
73                  }
74                  else {
75                          //SUPPPONGO DI SAPER RISOLVERE IL PROBLEMA P(n-1) e mettere
```

```
                        //la soluzione in tmp;
                        tmp=cancellaElementok(tmp->next,k);
                        //COSTRUISCO LA SOLUZIONE PER P(n);
                        top->next=tmp;
                }
        }
        return top;
}
```

1.20 Esercizio 19

```
// Data una lista semplicemente concatenata di interi, voglio invertirla, cioè
//il primo elemento diventa l'ultimo, il secondo il penultimo e così via.

#include <stdio.h>
#include <stdlib.h>
struct elem {
        int k;
        struct elem *next;
};

struct elem *riempiLista();
struct elem *cancellaMinoriDik(struct elem *top,int k);
struct elem *cancellaElementok(struct elem *top,int k);
struct elem *invertiLista(struct elem *top);

void stampaLista(struct elem *top);
int main(){
        struct elem *top;
        int k;
        top=riempiLista();
        stampaLista(top);
        printf("Dammi k:");
        scanf("%d",&k);
        top=cancellaMinoriDik(top,k);
        stampaLista(top);
return 0;
}
struct elem *riempiLista(){
        struct elem *top=NULL;
        return top;
}
//P(n)=Eliminare da un lista semplicemente concatenata di dimensione n,
// tutti gli elementi mionori di un dato k

```

```c
struct elem *cancellaMinoriDik(struct elem *top,int k){
	//CASO BASE, top==NULL, implicito
	struct elem *tmp;
	if (top!=NULL) {
		//ASSUMMO DI SAPER RISOLVERE IL PROBLEMA P(n-1), cioè
		//P(n)=Eliminare da un lista semplicemente concatenata di dimensione n-1,
		// tutti gli elementi mionori di un dato k
		//tmp mantiene la soluzione, cioè sarà il puntatore alla testa della
		//lista in cui sono stati eliminati gli elementi minori di k,
		//ecceto la testa iniziale (top)
		tmp=cancellaMinoriDik(top->next,k);
		//ADESSO COSTRUISCO LA SOLUZIONE PER P(n)
		if (top->k<k) {
			free(top);
			top=tmp;
			}
		else top->next=tmp;
	}
	return top;
}
void stampaLista(struct elem *top){
}

//P(n)= Data una lista di interi semplicemente concatenata e senza ripetizioni,
// con n elementi, Elimino un elemento di valore k, se esiste

struct elem *cancellaElementok(struct elem *top,int k){
	struct elem *tmp;
	//Passo base, top==NULL, implicito

	if (top!=NULL) {
		//Se è in testa devo solo cancellare la testa e non devo fare più nulla,
		// perchè è una lista SENZA ripetizioni
		if (top->k==k) {
			tmp=top->next;
			free(top);
			top=tmp;
		}
		else {
			//SUPPPONGO DI SAPER RISOLVERE IL PROBLEMA P(n-1) e mettere
			//la soluzione in tmp;
			tmp=cancellaElementok(tmp->next,k);
			//COSTRUISCO LA SOLUZIONE PER P(n);
			top->next=tmp;
		}
	}
```

```
	return top;
}
// P(n)=Data una lista semplicemente concatenata di n interi, voglio invertirla

struct elem *invertiLista(struct elem *top){
	//caso base implicito (top==NULL)
	struct elem *tmp,*curr;
	if (top!=NULL) {
		//Assumo che tmp contenga la soluzione per la lista senza la testa,
		//cioè la soluzione del problema P(n-1)=Data una lista semplicemente //
		//concatenata di n-1 interi, voglio invertirla
		tmp=invertiLista(top->next);
		//COSTRUISCO LA SOLUZIONE PER P(n)
		if (tmp!=NULL) {
			curr=tmp;
			while (curr->next!=NULL) curr=curr->next;
			curr->next=top;
			top->next=NULL;
			top=tmp;
		}
	}
	return top;
}
```

1.21 Esercizio 20

```
//Dato un array di n elementi interi, calcolare il massimo
#include <stdio.h>
#define MAXELEM 50
int riempiArray(int a[]);
int calcolaMassimo(int a[], int n);
int calcolaMassimo2(int a[], int n);

int main() {
	int a[MAXELEM];
	int n,m,m2;
	n=riempiArray(a);
	if (n>0) {
		m=calcolaMassimo(a,n);
		m2=calcolaMassimo2(a,n);


	printf("Il massimo e':%d %d\n",m,m2);
	}
	else printf("Non ho elementi!\n");
```

```
	return 0;
}

int riempiArray(int a[]){
	int n,i;
	printf("Dammi il numero di elementi:");
	scanf("%d",&n);
	for (i=0;i<n;++i) {
		printf("Dammi a[%d]=",i);
		scanf("%d",&a[i]);
	}
	return n;
}
//PRECONDIZIONE: n>0
//a è l'array, n è il numero di elementi nell'array
int calcolaMassimo(int a[], int n){
	int m;
	if (n==1) {
		m=a[0];
	}
	else {
		//SUPPONGO di Avere la soluzione per lo stesso problema
		//ma con un elemento in meno, P(n-1)
		//Ad esempio, tolgo l'ultimo elemento
		m=calcolaMassimo(a,n-1);
		//m è il massimo tra tutti gli elementi, eccetto l'ultimo a[n-1]
		//Quindi prendo m e costruisco la soluzione per P(n)
		if (a[n-1]>m) m=a[n-1];
		//Questo m è il massimo tra tutti gli elementi!!!!!
	}
	return m;
}
//PRECONDIZIONE: n>0
//a è l'array, n è il numero di elementi nell'array
int calcolaMassimo2(int a[], int n){
	int m;
	if (n==1) m=a[0];
	else {
		//SUPPONGO di Avere la soluzione per lo stesso problema
		//ma con un elemento in meno, P(n-1)
		//Ad esempio, tolgo il primo elemento
		m=calcolaMassimo2(&a[1],n-1);
		//m è il massimo tra tutti gli elementi, eccetto il primo
		//Quindi prendo m e costruisco la soluzione per P(n)
		if (a[0]>m) m=a[0];
		//Questo m è il massimo tra tutti gli elementi!!!!!
```

```
        }
        return m;
}
```

1.22 Esercizio 21

```
/*
   Ho una lista , un puntatore ad un nodo e voglio spostarlo In testa.
*/
#include <stdio.h>
#include <stdlib.h>
struct elem {
        int k;
        struct elem *next;
};

struct elem *newElem(int k);
struct elem *insInCodaR(struct elem *top, int k);
struct elem *insInOrdineR(struct elem *top, int k);
struct elem *insInOrdine(struct elem *top, int k);
struct elem *spostoElemInTesta(struct elem *top, struct elem *p);
void stampaLista(struct elem *top);
void stampaListaR1(struct elem *top);
void stampaListaR2(struct elem *top);


int main(){
        int n,i,k;
        struct elem *top=NULL;
        printf("Quanti elementi vuoi inserire:");
        scanf("%d",&n);
        for (i=0;i<n;++i) {
                printf("Dammi l'elemento da inserire:");
                scanf("%d",&k);
                top=insInOrdineR(top,k);
        }
        stampaListaR1(top);
        stampaListaR2(top);
return 0;
}

//P(n)= Ho una lista di n elementi ed un puntatore p ad un
//elemento e voglio mettere p in testa
struct elem *spostoElemInTesta(struct elem *top, struct elem *p){
        //P(n=0)
```

```
	struct elem *tmp;
	if (top!=NULL && p!=NULL) {
		if (p!= top) {
			//Soluzione per P(n-1)
			tmp= spostoElemInTesta(top->next,p);
			//Assumo che p è in testa alla nuova lista
			//tmp
			//Allora trovo soluzione per P(n)
			top->next=tmp->next;
			tmp->next=top;
			top=tmp;
		}
	}



	return top;
}

struct elem *newElem(int k){
	struct elem *nuovo=NULL;
	nuovo=(struct elem *)malloc(sizeof(struct elem));
	nuovo->k=k;
	nuovo->next=NULL;
	return nuovo;
}

//Il problem P(n)= "Inserire il valore k in coda ad un lista di n elementi"
struct elem *insInOrdineR(struct elem *top, int k){
	// P(n) =Ho una lista ordinata (crescente) di n elementi e voglio
	// inserire un elemento k in modo tale che
	//la lista resti ordinata (sempre crescente).

	struct elem *tmp;
	//P(n=0)
	if (top==NULL) {
		top=newElem(k);
	} else {
		if (k< top->k) {
			//Risolvo banalmente senza chiamata
			//ricorsiva
			tmp=newElem(k);
			tmp->next=top;
			top=tmp;
		} else
		{
```

```
86              //Problema P(n-1) lo so risovere:
87              tmp=insInOrdineR(top->next,k);
88              //Data la soluzione per P(n-1) risolvo per P(n)
89              //Adesso, quindi, assumo che k è stato messo
90              //nella posizione corretta nella lista senza il top,
91              //cioe' top->next
92              //Quindi:
93              top->next=tmp;
94              }
95      }
96      return top;
97  }
98  struct elem *insInOrdine(struct elem *top, int k){
99      struct elem *nuovo=newElem(k);
100     struct elem *prec,*curr;
101     curr=top;
102     prec=NULL;
103     while (curr!=NULL && curr->k < k){
104             prec=curr;
105             curr=curr->next;
106     }
107     if (curr==NULL) {
108             if (prec!=NULL)
109                     prec->next=nuovo;
110             else top=nuovo;
111     }
112     else {
113             if (prec!=NULL) {
114                     nuovo->next=curr;
115                     prec->next=nuovo;
116             }
117             else {
118                     nuovo->next=curr;
119                     top=nuovo;
120             }
121     }
122
123     return top;
124 }
125
126 struct elem *insInCodaR(struct elem *top, int k){
127     struct elem *tmp;
128     //PASSO BASE (CASO SEMPLICE) n=0, corrisponde top==NULL
129     if (top==NULL) {
130             top=newElem(k);
131     }
```

```
132         //CASO RICORSIVO
133         else { //QUI sono con un problema ad n elementi (n non lo conosco,
134                 // so solo che n>0
135                 //Problema P(n-1)="Inserire Il valore k in coda ad una lista di n-1 elementi", assumo
                        di saperlo risolvere
136                 tmp=insInCodaR(top->next,k);
137                 //All'uscita di questa chiamata ho inserito l'elemento
138                 //k in coda nella lista senza il primo elemento
139                 //Tale nuova lista è in tmp
140                 //Data la soluzione (tmp) per il problema con n-1 elementi
141                 //Risolvo il problema con n elementi:
142                 top->next=tmp;
143                 ///Piu' semplicemente:
144                 //top->next=insInCoda(top->next,k);
145         }
146
147         return top;
148 }
149 void stampaLista(struct elem *top){
150         while (top) {
151                 printf("%d -->",top->k);
152                 top=top->next;
153         }
154         printf("NULL\n");
155 }
156
157 void stampaListaR1(struct elem *top){
158         if (top==NULL) {
159                 printf("NULL\n");
160         }
161         else {
162                 printf("%d -->", top->k);
163                 stampaListaR1(top->next);
164         }
165 }
166 void stampaListaR2(struct elem *top){
167         if (top==NULL) {
168                 printf("NULL ");
169         }
170         else {
171                 stampaListaR2(top->next);
172                 printf("<-- %d ", top->k);
173
174         }
175 }
```

1.23 Esercizio 22

```c
//Lezione Lab Prog Gr. 2B del 2021.04.14
#include <stdio.h>
#define MAXLEN 10
//NON HO DEFINITO NESSUNA VARIABILE, MA HO ASSOCIATO ALL'IDENTIFICATORE miaStruc
// (si può scegliare qualunque nome all'interno di quelli ammissibili)
// UNA struct come quella definita
struct miaStruct {
	int n; //Capo della struct: nome: n tipo: int
	char c[MAXLEN];//Capo della struct: nome: c tipo: stringa
	float x; //Capo della struct: nome: x tipo: float
};

void stampaMiaStruct( struct miaStruct ss);
struct miaStruct riempiMiaStruct();
int main(){
	struct miaStruct rr; //QUI e' definita la variabile rr di tipo struct miaStruct
	rr=riempiMiaStruct();

	stampaMiaStruct(rr); //ANCHE IN QUESTO CASO PASSAGGIO PER VALORE
	struct miaStruct rr2;//QUi ho definito un' altra variable di tipo struct
	rr2=riempiMiaStruct();

	stampaMiaStruct(rr2); //ANCHE IN QUESTO CASO PASSAGGIO PER VALORE
return 0;
}
//ANCHE IN QUESTO CASO PASSAGGIO PER VALORE:
//E' CREATO IL PARAMETRO FORMALE ss (che è una struct miaStruct)
// Al parametro formale è assegnato il valore del parametro attuale (ad esempio rr del main),
// cioè ai campi di ss sono assegnati i valori dei rispettivi campi del parametro attuale,
// ad esempio queli di rr
void stampaMiaStruct( struct miaStruct ss){
	printf("c: %s x: %f n:%d\n",ss.c,ss.x,ss.n);
}

struct miaStruct riempiMiaStruct(){
	struct miaStruct ss; //Varialie struct locale alla funzione
	printf("Dammi carattere, intero, float:");
	scanf("%s %d %f",ss.c,&(ss.n),&(ss.x));
	return ss;
}
```

1.24 Esercizio 23

```c
//Lezione Lab Prog Gr. 2B del 2021.04.14
```

```c
#include <stdio.h>
#include <stdlib.h> //necessario per utilizzo malloc, calloc e free
//SUPPONIAMO DI VOLER RIEMPIRE E STAMPARE UN ARRAY CREATO CON MEMORIA DINAMICA


//FUNZIONE RESTITUISCE L'ARRAY CREATO DINAMICAMENTE, CIOE' RESTITUISCE L'INDIRIZZO
//DI MEMORIA DEL PRIMO ELEMENTO (QUESTO E' BENE FARLO SOLO PER MEMORIA ALLOCATA
//DINAMICAMENTE)
//HA COME ARGOMENTO l'indirizzo di memoria di una variabile in cui mettero
//il valore delle dimensioni dell'array creato dinamicamente
int *riempiArrayDinamico(int *pn);

//Dato l'array A di dimensioni n, stampo l'array
void stampaArray(int *A,int n);

//Libero la memoria per l'array allocato dinamicamente
void liberaArray(int *A);

int main(){
	int *V; //Puntatore ad un intero
	int n; // n variabile che mantiene l'informazione
		//sulle dimensioni dell'array elementi
	V=riempiArrayDinamico(&n);
	if (V!=NULL) {
		stampaArray(V,n); //Stampo array
		liberaArray(V); //Libero la memoria creata dinamicamente
		}
		else printf("ARRAY NON CREATO PER MEMORIA INSUFFICIENTE!!!");
return 0;
}

int *riempiArrayDinamico(int *pn){
	int n,i;
	//size_t dipende dal sistema operativo e versione c,
	//in genere definito come unsigned int
	//void* calloc (size_t num, size_t size); Alloco num elementi di dimensione size byte.
	//                                  Memoria contigua e azzero la memoria allocata
	//void* malloc (size_t size);     Alloco size byte (memoria contigua)
	printf("Quanti elementi vuoi inserire?");
	scanf("%d",&n);
	//int *A=(int *)calloc(n,sizeof(int)); //int *A=(int *)calloc(n,sizeof(n))
	//UNA ALTERNATIVA
	int *A= (int *) malloc(n*sizeof(int)); //Con malloc devo stare attento alla
	                                       //inizializzazione, perche' non ho una
	                                       //inizializzazione a zero "automatica"
	if (A!=NULL) {
```

```
                for (i=0;i<n;++i) {
                printf("dammi A[%d]=",i);
                scanf("%d",&A[i]);
                }
                n=0;
        }
        *pn=n;   // Accedo al contenuto presente nell'indirizzo di memoria mantenut in pn
                 // e gli assegno il valore di n
        return A; //Restituisco il valore contenuto in A
}
void stampaArray(int *A,int n){
        int i;
        for (i=0;i<n;++i)
                printf("A[%d]=%d\n",i,A[i]);

}
void liberaArray(int *A){
        //void free(void *ptr)
        free(A);
}
```

1.25 Esercizio 24

```
//Una variabile e' sempre definita da: un tipo, nome, un valore ed un indirizzo di memoria
#include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
                    di questo file*/
void fun(int);
int main() {
        int x=0;
        printf("x address: %p x value=%d\n",&x, x);
        //p --> x
        int *p=&x;
        printf("p address: %p p value=%p\n",&p, p);

        printf("Dammi x:");
        scanf("%d",&x);
        if (x>0) {
                int x=0;
                printf("Indirizzo nuovo x:%p\n",&x);
                //Ho nascosto l'accesso alla variabile x definita precedentemente
                *p=10;
        }
        printf("x address: %p x value=%d\n",&x, x);
return 0;
}
```

1.26 Esercizio 25

```c
//Importanza della inizializzazione di una variabile.
//Verifica sperimentale che le variabile automatiche sono caricate in memoria
//quando il loader carica il programma in memoria
#include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
                    di questo file*/
void fun(int);

int main() {
       int x;
       printf("Dammi x:");
       scanf("%d",&x); //x=1, x=2
       fun(x);
       printf("Dammi x:");
       scanf("%d",&x);//x=2, x=1
       fun(x);
return 0;
}


void fun(int x){
   //La variabile a non e' inizializzata!!!
       int a;
       printf("indirizzo a:%p", &a);
       if (x==1) a=10; else a=a+1;
              printf("a:%d\n",a); //x=1,Stampo 10, x=2,Stampo 11
                                  //x=2, Stampo ???, x=1, Stampo 10
}
```

1.27 Esercizio 26

```c
/*Alcune note sulla creazione di matrici in maniera automatica e dinamica*/
#include <stdio.h>
#include <stdlib.h> //Da includere se uso malloc o calloc
#define MAXR 10
#define MAXC 10
//Creazione matrici dinamiche (tre modi diversi)
int **creaMatriceDinamica1(int r,int c);
int **creaMatriceDinamica2(int r,int c);
int *creaMatriceDinamica3(int r,int c);

//Esempio riempimento matrice automatica
void riempiMatrice(int M[][MAXC], int *pr,int *pc);
//Esempio riempimento matrice dinamica
void riempiMatriceDinamica1(int **M, int maxR,int maxC,int *pr,int *pc);
```

```c
void riempiMatriceDinamica2(int **M, int maxR,int maxC,int *pr,int *pc);
void riempiMatriceDinamica3(int *M, int maxR,int maxC,int *pr,int *pc);

//Esempio stampa di una matrice automatica
void stampaMatrice(int M[][MAXC], int r,int c);
//Esempio stampa di una matrice dinamica (tre modi diversi, uno per ogni tipo di matrice)
void stampaMatriceDinamica1(int **M, int r,int c);
void stampaMatriceDinamica2(int **M,int r,int c);
void stampaMatriceDinamica3(int *M,int r,int c);

//Liberare memoria per una matrice dinamica (tre modi diversi, uno per ogni tipo di matrice)
void liberaMatriceDinamica1(int **M, int r);
void liberaMatriceDinamica2(int **M);
void liberaMatriceDinamica3(int *M);


int main(){
	int r,c,maxR,maxC;
	int M[MAXR][MAXC]; //CREAZIONE MATRICE IN MEMORIA AUTOMATICA
	int **M1,**M2;
	int *M3;
	/* //PER CAPIRE COME E' ORGANIZZATO M
	//UN UNICO ARRAY di rxc elementi. M definito come un
	//puntatore ad una array di MAXC elementi
	for (r=0;r<MAXR;++r)
		printf("r=%d M value:%p &M[r][0]=%p M[r]=%p \n",r, M, &M[r][0],M[r]);
	//Con l'algebra dei puntatori mi sposto di MAXC blocchi alla volta
	for (r=0;r<MAXR;++r)
		printf("%p %p\n", M+r,&M[r][0]);
			/*
			*/
	printf("Matrice memoria automatica:\n");
	riempiMatrice(M, &r,&c); //r e c manterranno le effettive dimensioni della matrice
	stampaMatrice(M,r,c);
	printf("Dammi i valori di maxR e maxC per le matrici dinamiche:");
	scanf("%d %d",&maxR,&maxC);

	printf("Matrice memoria dinamica 1\n");
	M1=creaMatriceDinamica1(maxR,maxC);
	riempiMatriceDinamica1(M1,maxR,maxC,&r,&c);
	stampaMatriceDinamica1(M1,r,c);
	liberaMatriceDinamica1(M1, r);

	printf("Matrice memoria dinamica 2\n");
	M2=creaMatriceDinamica2(maxR,maxC);
	riempiMatriceDinamica2(M2, maxR,maxC,&r,&c);
```

```
61        stampaMatriceDinamica2(M2,r,c);
62        liberaMatriceDinamica2(M2);
63
64        printf("Matrice memoria dinamica 3\n");
65        M3=creaMatriceDinamica3(maxR,maxC);
66        riempiMatriceDinamica3(M3, maxR,maxC,&r,&c);
67        stampaMatriceDinamica3(M3,r,c);
68        liberaMatriceDinamica3(M3);
69
70
71  return 0;
72  }
73
74  int **creaMatriceDinamica1(int r,int c){
75        int i;
76        int **M;
77        M= (int **) malloc(r*sizeof(int *));
78        for (i=0;i<r;++i) M[i]= (int *)malloc(c*sizeof(int));
79        // In maniera equivalente, con l'unica differenza che inizializzo a zero
80        // tutti gli elementi:
81        //for (i=0;i<r;++i) A[i]= (int *)calloc(c,sizeof(int));
82        return M;
83  }
84
85  int **creaMatriceDinamica2(int r,int c){
86        int i;
87        int *V= (int *)malloc(r*c*sizeof(int));
88        // In maniera equivalente, con l'unica differenza che inizializzo a zero
89        // tutti gli elementi:
90        // int *V= (int *)calloc(r*c,sizeof(int));
91        int **M=(int **) malloc(r*sizeof(int *));
92        for (i=0;i<r;++i) M[i]= &V[i*c];
93        return M;
94  }
95  int *creaMatriceDinamica3(int r,int c){
96        int i;
97        int *M= (int *)malloc(r*c*sizeof(int));
98        // In maniera equivalente, con l'unica differenza che inizializzo a zero
99        // tutti gli elementi:
100       // int *M= (int *)calloc(r*c,sizeof(int));
101       return M;
102 }
103
104 void riempiMatrice(int M[][MAXC], int *pr,int *pc){
105       int r,c,i,j;
106       printf("Quante righe vuoi inserire? (max r:%d):",MAXR);
```

```
107         scanf("%d",&r);
108         printf("Quante colonne vuoi inserire? (max c:%d)",MAXC);
109         scanf("%d",&c);
110         if (r>MAXR) r=MAXR;
111         if (c>MAXC) c=MAXC;
112
113         for (i=0;i<r;++i)
114                 for (j=0;j<c;++j) M[i][j]=i*c+j;
115         *pr=r; // Con *pr accedo al contenuto della variabile puntata da pr
116         *pc=c;
117 }
118 void riempiMatriceDinamica1(int **M, int maxR,int maxC,int *pr,int *pc){
119
120         int r,c,i,j;
121         printf("Quante righe vuoi inserire? (max r:%d):",maxR);
122         scanf("%d",&r);
123         printf("Quante colonne vuoi inserire? (max c:%d)",maxC);
124         scanf("%d",&c);
125         if (r>maxR) r=maxR;
126         if (c>maxC) c=maxC;
127         for (i=0;i<r;++i)
128                 for (j=0;j<c;++j)
129                         //M[i] e' un puntatore alla i-sima riga della matrice
130                         //Puo' essere visto come il nome del array corrispondente alla i-sima riga
131                         M[i][j]=i*c+j;
132         *pr=r;
133         *pc=c;
134
135 }
136 void riempiMatriceDinamica2(int **M, int maxR,int maxC,int *pr,int *pc){
137         int r,c,i,j;
138         printf("Quante righe vuoi inserire? (max r:%d):",maxR);
139         scanf("%d",&r);
140         printf("Quante colonne vuoi inserire? (max c:%d)",maxC);
141         scanf("%d",&c);
142         if (r>maxR) r=maxR;
143         if (c>maxC) c=maxC;
144         for (i=0;i<r;++i)
145                 for (j=0;j<c;++j)
146                         M[i][j]=i*c+j;
147         *pr=r;
148         *pc=c;
149
150 }
151 void riempiMatriceDinamica3(int *M, int maxR,int maxC,int *pr,int *pc){
152         int r,c,i,j;
```

```
153         printf("Quante righe vuoi inserire? (max r:%d):",maxR);
154         scanf("%d",&r);
155         printf("Quante colonne vuoi inserire? (max c:%d)",maxC);
156         scanf("%d",&c);
157         if (r>maxR) r=maxR;
158         if (c>maxC) c=maxC;
159         for (i=0;i<r;++i)
160                 for (j=0;j<c;++j)
161                         M[i*c+j]=i*c+j;
162         *pr=r;
163         *pc=c;
164
165 }
166
167 void stampaMatrice(int M[][MAXC], int r,int c){
168         int i,j;
169         for (i=0;i<r;++i){
170                 printf("\n");
171                 for (j=0;j<c;++j)
172                         printf("M[%d][%d]= %d ",i,j,M[i][j]);
173         }
174         printf("\n");
175
176 }
177
178 void stampaMatriceDinamica1(int **M, int r,int c){
179         int i,j;
180         for (i=0;i<r;++i){
181                 printf("\n");
182                 for (j=0;j<c;++j)
183                         printf("M[%d][%d]= %d ",i,j,M[i][j]);
184         }
185         printf("\n");
186
187 }
188 void stampaMatriceDinamica2(int **M,int r,int c){
189         int i,j;
190         for (i=0;i<r;++i){
191                 printf("\n");
192                 for (j=0;j<c;++j)
193                         printf("M[%d][%d]= %d ",i,j,M[i][j]);
194         }
195         printf("\n");
196
197 }
198 void stampaMatriceDinamica3(int *M,int r,int c){
```

```
        int i,j;
        for (i=0;i<r;++i){
                printf("\n");
                for (j=0;j<c;++j)
                        printf("M[%d][%d]= %d ",i,j,M[i*c+j]);
        }
        printf("\n");

}
void liberaMatriceDinamica1(int **M, int r){
        int i;
        for (i=0;i<r;++i) free(M[i]);
        free(M);
}

void liberaMatriceDinamica2(int **M){
        int i;
        free(M[0]);
        free(M);
}
void liberaMatriceDinamica3(int *M){
        free(M);
}
```

1.28 Esercizio 27

```
//SCHEDARIO
#include <stdio.h>
#define MAXLEN 1000
#define MAXS 20

struct schedaPersona {
        char cognome[MAXS];
        char nome[MAXS];
        char luogoDiNascita[MAXS];
        int eta;
        float reddito;
};

int inseriscoPersona(struct schedaPersona *schedario, int n);
int eliminaPersona(struct schedaPersona *schedario,int n, int i);
void modificaPersona(struct schedaPersona *schedario,int n, int i);
int trovaPersoneCognome(struct schedaPersona *schedario,int n,char *cog);
void stampaArray(struct schedaPersona *schedario,int n);
int main() {
```

```
	int flag;
	struct schedaPersona schedario[MAXLEN];
	int nPersone=0; int i;
	char cog[MAXS];
	do {
		printf("Cosa vuoi fare? 0: esci, 1: inserisci, 2: elmina 3:modifica 4:stampa");
		scanf("%d",&flag);
		switch (flag) {
			case 1: nPersone=inseriscoPersona(schedario,nPersone);
				break;
			case 2: printf("Dammi il cognome della persona da eliminare");
				scanf("%s",cog);
				i=trovaPersoneCognome(schedario,nPersone,cog);
				if (i>=0)
					nPersone=eliminaPersona(schedario,nPersone,i);
				break;
			case 3:
				printf("Dammi il cognome della persona da modificare");
				scanf("%s",cog);
				i=trovaPersoneCognome(schedario,nPersone,cog);
				if (i>=0)
					modificaPersona(schedario,nPersone,i);
				break;
			case 4: stampaArray(schedario,nPersone);
		}
	} while (flag>0);
return 0;
}

struct schedaPersona creoPersona();
int inseriscoPersona(struct schedaPersona *schedario, int n){
	schedario[n]=creoPersona();
	n=n+1;
	return n;
}

struct schedaPersona creoPersona(){
	struct schedaPersona s;
	printf("Dammi il cognome:");
	scanf("%s",s.cognome);
	printf("Dammi il nome:");
	scanf("%s",s.nome);
	printf("Dammi l'eta':");
	scanf("%d",&(s.eta));
	printf("Dammi il luogo di nascita:");
	scanf("%s",s.luogoDiNascita);
```

```
66          printf("Dammi il reddito:");
67          scanf("%f",&(s.reddito));
68          return s;
69  }
70
71  int eliminaPersona(struct schedaPersona *schedario,int n, int i){
72          return n;
73  }
74  void modificaPersona(struct schedaPersona *schedario,int n, int i){
75  }
76
77  int trovaPersoneCognome(struct schedaPersona *schedario,int n,char *cog){
78          return n;
79  }
80  void stampaArray(struct schedaPersona *schedario,int n){
81          int i;
82          for (i=0;i<n;++i) {
83                  printf("Cognome:%s Nome:%s Eta:%d Luogo:%s Reddito:%f\n",schedario[i].cognome,
                        schedario[i].nome,schedario[i].eta,schedario[i].luogoDiNascita,schedario[i].reddito
                        );
84          }
85  }
```

1.29 Esercizio 28

```
1   //SCHEDARIO
2   #include <stdio.h>
3   #include <stdlib.h>
4   #define MAXLEN 1000
5   #define MAXS 20
6
7   struct schedaPersona {
8           char cognome[MAXS];
9           char nome[MAXS];
10          char luogoDiNascita[MAXS];
11          int eta;
12          float reddito;
13  };
14
15  int inseriscoPersona(struct schedaPersona **schedario, int n);
16  int eliminaPersona(struct schedaPersona **schedario,int n, int i);
17  void modificaPersona(struct schedaPersona **schedario,int n, int i);
18  int trovaPersoneCognome(struct schedaPersona **schedario,int n,char *cog);
19  void stampaArray(struct schedaPersona **schedario,int n);
20  int main() {
```

```
	int flag;
	struct schedaPersona *schedario[MAXLEN]; //Array di puntatori a struct schedaPersona

	int nPersone=0; int i;
	char cog[MAXS];
	for (i=0;i<MAXLEN;++i) schedario[i]=NULL;
	do {
		printf("Cosa vuoi fare? 0: esci, 1: inserisci, 2: elmina 3:modifica 4:stampa");
		scanf("%d",&flag);
		switch (flag) {
			case 1: nPersone=inseriscoPersona(schedario,nPersone);
				break;
			case 2: printf("Dammi il cognome della persona da eliminare");
				scanf("%s",cog);
				i=trovaPersoneCognome(schedario,nPersone,cog);
				if (i>=0)
					nPersone=eliminaPersona(schedario,nPersone,i);
				break;
			case 3:
				printf("Dammi il cognome della persona da modificare");
				scanf("%s",cog);
				i=trovaPersoneCognome(schedario,nPersone,cog);
				if (i>=0)
					modificaPersona(schedario,nPersone,i);
				break;
			case 4: stampaArray(schedario,nPersone);
		}
	} while (flag>0);
return 0;
}

struct schedaPersona *creoPersona();
int inseriscoPersona(struct schedaPersona **schedario, int n){
	schedario[n]=creoPersona();
	n=n+1;
	return n;
}

struct schedaPersona *creoPersona(){
	struct schedaPersona *s=(struct schedaPersona *)malloc(sizeof(struct schedaPersona));
	printf("Dammi il cognome:");
	scanf("%s",s->cognome);
	printf("Dammi il nome:");
	scanf("%s",s->nome);
	printf("Dammi l'eta':");
	scanf("%d",&(s->eta));
```

```
67          printf("Dammi il luogo di nascita:");
68          scanf("%s",s->luogoDiNascita);
69          printf("Dammi il reddito:");
70          scanf("%f",&(s->reddito));
71          return s;
72  }
73
74  int eliminaPersona(struct schedaPersona **schedario,int n, int i){
75          return n;
76  }
77  void modificaPersona(struct schedaPersona **schedario,int n, int i){
78  }
79
80  int trovaPersoneCognome(struct schedaPersona **schedario,int n,char *cog){
81          return n;
82  }
83  void stampaArray(struct schedaPersona **schedario,int n){
84          int i;
85          for (i=0;i<n;++i) {
86                  printf("Cognome:%s Nome:%s Eta:%d Luogo:%s Reddito:%f\n",schedario[i]->cognome,
                        schedario[i]->nome,schedario[i]->eta,schedario[i]->luogoDiNascita,schedario[i]->
                        reddito);
87          }
88
89  }
```