

# 1 Esercizi

## 1.1 Esempio Prima libreria

### 1.1.1 main

```
1 //ESEMPIO DI SVILUPPO DI UN PROGRAMMA IN PIU' FILE: File di dicharazione funzioni,ed
2 //eventualmente #define (header file), e file di definizione di funzioni
3 // Qui e' riportato un esempio semplice costituito da un unico header file ed un unico
4 //file di definizione di funzioni:
5 // Il file header e' "matriciDinamiche.h",
6 // Il file di definizione delle funzioni e': libMatriciDinamiche.c, di cui deve essere creato
7 // l'object file libMatriciDinamiche.o tramite il comando gcc -c libMatriciDinamiche.c
8 //COMPILARE CON: gcc esempioPropriaLibreria.c libMatriciDinamiche.o -o esempioPropriaLibreria
9 //PER ULTERIORI APPROFONDIMENTI VEDERE: https://www.cs.dartmouth.edu/~campbell/cs50/buildlib.html
10
11 #include <stdio.h>
12 #include "matriciDinamiche.h"
13 int main(){
14     int r,c;
15     int **M1,**M2,*M3;
16     printf("Dammi r e c:");
17     scanf("%d %d",&r,&c);
18     M1=creaMatriceDinamica1(r,c);
19     M2=creaMatriceDinamica2(r,c);
20     M3=creaMatriceDinamica3(r,c);
21     //ALTRO CODICE
22     liberaMatriceDinamica1(M1,r);
23     liberaMatriceDinamica2(M2);
24     liberaMatriceDinamica3(M3);
25     return 0;
26 }
```

### 1.1.2 lib

```
1 #include <stdlib.h>
2 #include "matriciDinamiche.h"
3
4 int **creaMatriceDinamica1(int r,int c){
5     int i;
6     int **M;
7     M= (int **) malloc(r*sizeof(int *));
8     for (i=0;i<r;++i) M[i]= (int *)malloc(c*sizeof(int));
9     // In maniera equivalente, con l'unica differenza che inizializzo a zero
10    // tutti gli elementi:
11    //for (i=0;i<r;++i) A[i]= (int *)calloc(c,sizeof(int));
```

```

12     return M;
13 }
14
15 int **creaMatriceDinamica2(int r,int c){
16     int i;
17     int *V= (int *)malloc(r*c*sizeof(int));
18     // In maniera equivalente, con l'unica differenza che inizializzo a zero
19     // tutti gli elementi:
20     // int *V= (int *)calloc(r*c,sizeof(int));
21     int **M=(int **) malloc(r*sizeof(int *));
22     for (i=0;i<r;++i) M[i]= &V[i*c];
23     return M;
24 }
25 int *creaMatriceDinamica3(int r,int c){
26     int i;
27     int *M= (int *)malloc(r*c*sizeof(int));
28     // In maniera equivalente, con l'unica differenza che inizializzo a zero
29     // tutti gli elementi:
30     // int *M= (int *)calloc(r*c,sizeof(int));
31     return M;
32 }
33
34 void liberaMatriceDinamica1(int **M, int r){
35     int i;
36     for (i=0;i<r;++i) free(M[i]);
37     free(M);
38 }
39
40 void liberaMatriceDinamica2(int **M){
41     int i;
42     free(M[0]);
43     free(M);
44 }
45 void liberaMatriceDinamica3(int *M){
46     free(M);
47 }

```

### 1.1.3 header

```

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

```

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

## 1.2 Esercizio 1

```
1 #include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
2 di questo file*/
3 /*Versione Procedurale di tipo top-down*/
4 struct risAndRes {
5     int ris;
6     int res;
7     float pippo;
8 }; //Ricordo che sto definendo la struttura ma NON sto creando alcuna variabile!!!!
9 int leggiNumero();
10 struct risAndRes estraiPrimaCifra(int);
11 void stampa(int,int);
12 int main() {
13     int x=0,c=0;
14     struct risAndRes ss; //Creo la variabile ss di tipo "struct risAndRes"
15     x = leggiNumero();
16     do {
17         ss= estraiPrimaCifra(x);
18         stampa(c,ss.res);
19         x=ss.ris;
20         ++c;
21     } while(x!=0);
22     return 0;
23 }
24
25 int leggiNumero(){
26     int x;
27
28     printf("*****\n");
29     printf("Dammi un numero intero:");
30     scanf("%d",&x);
31
32     return x;
33 }
34 struct risAndRes estraiPrimaCifra(int x){
35     struct risAndRes tmp;
36
37     tmp.ris=x/10;
38     tmp.res=x%10;
39
40     return tmp;
```

```

41 }
42
43 void stampa(int conta, int cifra){
44
45     printf("cifra corrispondente a 10 elevoto a%d =%d\n",conta,cifra);
46 }

```

### 1.3 Esercizio 2

```

1  #include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
2      di questo file*/
3  /*Versione Procedurale di tipo top-down*/
4  struct risAndRes {
5      int ris;
6      int res;
7      float pippo;
8  }; //Ricordo che sto definendo la struttura ma NON sto creando alcuna variabile!!!!
9  int leggiNumero();
10 int leggiCifra();
11 int verificaCifra(int,int);
12 void stampa(int);
13 int main() {
14     int x=0,cifra, presente=0;
15     struct risAndRes ss; //Creo la variabile ss di tipo "struct risAndRes"
16     x = leggiNumero();
17     cifra=leggiCifra();
18     presente=verificaCifra(x,cifra);
19     stampa(presente);
20     return 0;
21 }
22 int leggiCifra(){
23     int x;
24     printf("Dammi la cifra da trovare:");
25     scanf("%d",&x);
26     return x;
27 }
28
29 int leggiNumero(){
30     int x;
31
32     printf("*****\n");
33     printf("Dammi un numero intero:");
34     scanf("%d",&x);
35
36     return x;

```

```

37 }
38 struct risAndRes estraiPrimaCifra(int x){
39     struct risAndRes tmp;
40
41     tmp.ris=x/10;
42     tmp.res=x%10;
43
44     return tmp;
45 }
46
47 int verificaCifra(int x,int cifra){
48     int presente=0;
49     struct risAndRes ss;
50     do {
51         ss= estraiPrimaCifra(x);
52         if (cifra== ss.res) presente=1;
53         x=ss.ris;
54     } while(x!=0 && presente==0);
55
56     return presente;
57 }
58 void stampa(int presente){
59     if (presente==1)
60         printf("La cifra e' presente");
61     else
62         printf("La cifra non e' presente");
63 }

```

## 1.4 Esercizio 3

```

1  #include <stdio.h>
2
3  int verificaCodizione(int,int,int);
4  void stampa(int);
5  int main(){
6      int verifica=0;
7      int a,b,c;
8      printf("Dammi tre interi positivi:");
9      scanf("%d %d %d",&a,&b,&c);
10     verifica=verificaCodizione(a,b,c);
11     stampa(verifica);
12     return 0;
13 }
14
15 int verificaCodizione(int a,int b,int c){

```

```

16     int ver=0;
17     if (a*a == (b*b + c*c)) ver=1;
18     else if (b*b== (a*a+c*c)) ver=1;
19     else if (c*c == (b*b+a*a)) ver=1;
20     return ver;
21 }
22 void stampa(int verifica){
23     if (verifica==1) printf("E' un triangolo rettangolo!\n");
24     else printf("Non e' un triangolo rettangolo!\n");
25 }

```

## 1.5 Esercizio 4

```

1 #include <stdio.h>
2 //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.
3
4 int riceviInput();
5 int fattoriale(int);
6 int main() {
7     int n=0,res=0;
8     //FASE IN CUI PRENDO LE INFORMAZIONI
9     //printf("Dammi il valore di n:");
10    //scanf("%d",&n);
11    n=riceviInput();
12    //FASE IN CUI RISOLVO IL PROBLEMA
13    res=fattoriale(n);
14
15    //FASE IN CUI DO LA SOLUZIONE ALL'ESTERNO
16    printf("Il fattoriale di n=%d e' = %d\n",n,res);
17    return 0;
18 }
19 int riceviInput(){
20     int n;
21     do {
22         printf("Dammi il valore di n maggiore o uguale a 0:");
23         scanf("%d",&n);
24     } while (n<0);
25     return n;
26 }
27 //PRE-CONDIZIONE: n>=0
28 int fattoriale(int n){
29     int fatt=1;
30

```

```

31     for (fatt=1; n>1;n=n-1) fatt=n*fatt;
32     //if (n>0) fatt=n*fattoriale(n-1);
33     return fatt;
34 }

```

## 1.6 Esercizio 5

```

1 //Scrivere un programma che riceva in input una stringa e stampi a video
2 //la lunghezza della stringa.
3 #include <stdio.h>
4 #define MAXLEN 100
5 int lenStr(char str[]);
6
7 int main() {
8     char s[MAXLEN];
9
10    int len;
11    printf("Dammi una stringa:");
12    scanf("%s",s);
13    len=lenStr(s);
14    printf("Lunghezza stringa:%d\n",len);
15    return 0;
16 }
17
18 //PRECONDIZIONE: str deve essere una stringa, cioe' array di caratteri
19 //con carattere di fine stringhe
20 int lenStr(char str[]){
21     int len=0;
22     while (str[len]!='\0') len=len+1;
23     return len;
24 }

```

## 1.7 Esercizio 6

```

1 /*
2 Scrivere un programma che riceva in input due stringhe s1 e s2 e stampi a video se la stringa s1 è
3 una sottostringa di s2 sono oppure no. Ad esempio se s1="ala e s2="calamaro allora s1 è una
4 sottostringa di s2. Se s1="ala e s2= ""calma, allora s1 non è una sottoscritta di s2.
5 */
6 #include <stdio.h>
7 #define MAXSTR 100 //Lunghezza massima di una stringa
8
9 //DICHIO LE FUNZIONI CHE MI SERVONO NEL MAIN (PROTOTIPI)
10 void leggiStringhe(char s1[],char s2[]); //Leggo i dati
11 int verificaSottostringa(char s1[],char s2[]); //Operazioni da svolgere

```

```

10 void stampaRisultato(int ris,char s1[],char s2[]); //Stampo il risultato
11 int main(){
12     char s1[MAXSTR], s2[MAXSTR]; //STUTTURE DATI DOVE MEMORIZZARE I DATI
13     int ris=0;
14     printf("PARAMETRO ATTUALE: Indirizzo di memoria di s1=%p e suo valore=%p\n", &s1,s1);
15     leggiStringhe(s1,s2); //Leggo i dati
16     ris=verificaSottostringa(s1,s2); //Operazioni da svolgere
17     printf("Indirizzo memoria di ris del main (parametro attuale):%p\n",&ris);
18     stampaRisultato(ris,s1,s2); //Stampo il risultato
19     return 0;
20 }
21
22 void leggiStringhe(char s1[],char s2[]){
23     printf("Dammi la prima stringa:");
24     scanf("%s",s1); //scanf("%s",&s1[0]);
25     printf("Dammi la seconda stringa:");
26     scanf("%s",s2); //scanf("%s",&s1[0]);
27 }
28
29 //VERIFICO SE E' UNA SOTTOSTRINGA A PARTIRE DALLA CONDIZIONE CHE IL CARATTERE
30 //PRECEDENTE E' UGUALE
31 int subStr(char s1[],char s2[],int c1,int c2){
32     //s1="ala" s2="camalaro"
33     int ris=0;
34
35     //CONFRONO SE IN SEQUENZA I CARATTERI SONO TUTTI UGUALI
36     while ((s1[c1]!='\0' && s2[c2]!='\0') && (s2[c2]==s1[c1])) {
37         ++c1;
38         ++c2;
39     }
40     //S1 e' una sottostringa se sono arrivato alla fine di S1
41     if (s1[c1]=='\0') ris=1;
42
43     return ris;
44 }
45
46 int verificaSottostringa(char s1[],char s2[]){
47     int ris=0;
48     int c1=0,c2=0;
49     while ( ris==0 && s2[c2]!='\0'){
50         while (s2[c2]!='\0' && s1[c1]!=s2[c2]) ++c2;
51         if (s2[c2]!='\0') ris= subStr(s1,s2,c1+1,c2+1);
52         ++c2;
53     }
54     return ris;
55 }

```



```

56
57
58 void stampaRisultato(int ris,char s1[],char s2[]){
59     printf("La stringa: %s\n",s1);
60     if (ris==1) printf("E' sottostringa della\n");
61     else printf("NON e' sottostringa della\n");
62     printf("stringa: %s\n",s2);
63 }

```

## 1.8 Esercizio 7

```

1  /*Scrivere un programma che memorizzi in un array n numeri interi. Dopodichè
2  calcoli il valore medio ed elimini 'dallarray tutti gli elementi minori del valore
3  medio. Stampi a video 'larray così ottenuto.*/
4  #include <stdio.h>
5  #define MAXLEN 100
6
7  int leggiNumeri(int a[]);
8  float valoreMedia(int a[], int n);
9  int eliminaElementi(int a[],int n, int m);
10 void stampaElementi(int a[], int n);
11
12 int main(){
13     int a[MAXLEN];
14     int n;
15     float m;
16     n=leggiNumeri(a);
17     //n rappresenta il numero effettivo di elementi inseriti (presenti nell'array)
18     //MAXLEN e' la capacità massima dell'array.
19     m=valoreMedia(a, n);
20     n= eliminaElementi(a,n,m);
21     stampaElementi(a, n);
22
23     return 0;
24 }
25
26 int leggiNumeri(int a[]){
27     int flag=0, n=0;
28     do {
29         printf("vuoi inserire un altro numero? (SI=1,NO=0)");
30         scanf("%d",&flag);
31         //n < MAXLEN && flag==1
32         if (flag==1 && n < MAXLEN) {
33             printf("Inserisci numero:");
34             scanf("%d",&a[n]);

```

```

35         ++n;
36     }
37     } while (flag==1);
38     return n;
39 }
40
41 float valoreMedia(int a[], int n){
42     float m=0;
43     int i;
44     for (i=0;i<n;++i) m += a[i];
45     m=m/n;
46     return m;
47 }
48 void shiftLeft(int a[],int n,int i);
49 int eliminaElementi(int a[],int n, int m){
50     int i=0;
51     while (i<n) {
52         if (a[i]<m) {
53             shiftLeft(a,n,i);
54             n=n-1; // --n;
55         }
56         else ++i;
57     }
58     return n;
59 }
60
61 void shiftLeft(int a[],int n,int i){
62     int j;
63     for (j=i; j < n-1;++j)
64         a[j]=a[j+1];
65 }
66
67 void stampaElementi(int a[], int n){
68     int i;
69     for (i=0;i<n;++i)
70         printf("a[%d]=%d\n",i,a[i]);
71 }

```

## 1.9 Esercizio 8

```

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

```

```

6 k=1 [1 2 4 7 12]
7 k=3 [3 6 12 21 36]
8 */
9 #include <stdio.h>
10
11 #define MAXLEN 100
12
13 int leggiNumeri(int a[]);
14 int verificaCondizione(int a[],int k,int n);
15 void stampa(int res);
16
17 int main(){
18     int n,res,k;
19     int a[MAXLEN];
20     n=leggiNumeri(a);
21     printf("Dammi k:");
22     scanf("%d",&k);
23     res=verificaCondizione(a,k,n);
24     stampa(res);
25     return 0;
26 }
27
28 int leggiNumeri(int a[]){
29     int flag=0, n=0;
30     do {
31         printf("vuoi inserire un altro numero? (SI=1,NO=0)");
32         scanf("%d",&flag);
33         //n < MAXLEN && flag==1
34         if (flag==1 && n < MAXLEN) {
35             printf("Inserisci numero:");
36             scanf("%d",&a[n]);
37             ++n;
38         }
39     } while (flag==1);
40     return n;
41 }
42
43 int verificaCondizione(int a[],int k,int n){
44     int res=1,i=0, somma=0;
45     while (res==1 && i<n) {
46         if (i <1) somma=k;
47         else if (i<2) somma=a[0]+k;
48         else somma=a[i-1] + a[i-2] +k;
49
50         if (a[i] != somma)
51             res=0;

```

```

52         else ++i;
53     }
54     return res;
55 }
56 void stampa(int res){
57     if (res==1) printf("Condizione verificata!!!\n");
58     else printf("Condizione NON verificata!!\n");
59 }

```

## 1.10 Esercizio 9

```

/*
Si supponga che in un file di testo siano memorizzati i valori di una matrice di interi di
dimensione  $m \times n$ , 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;

```

```

10     char nomeFile[MAXLEN];
11     printf("Dammi il nome del file:");
12     scanf("%s",nomeFile);
13     //fp=fopen(nomeFile,"w"); //APRO IN SCRITTURA. Se esiste lo cancello. Se non esiste lo
14         //creo nuovo
15     fp=fopen(nomeFile,"a");//APRO IN SCRITTURA. Se esiste non lo cancello.
16         //Se non esiste lo creo nuovo
17         //Scrivo a partire dalla fine del file
18     if (fp!=NULL) {
19         printf("Quanti interi vuoi scrivere?");
20         scanf("%d",&n);
21         for (i=0;i<n;++i) {
22             printf("Dammi il valore da scrivere su file:");
23             scanf("%d",&k);
24             fprintf(fp,"%d ",k);
25         }
26
27         fclose(fp);
28
29         fp=fopen(nomeFile,"r");
30         if (fp!=NULL) {
31             while (fscanf(fp,"%d",&k)>0)
32                 printf("%d\n",k);
33             fclose(fp);
34         }
35         else printf("Non sono riuscito ad aprire il file in lettura\n");
36     }
37     else {
38         printf("Problema apertura file in scrittura!\n");
39     }
40 }

```

## 1.12 Esercizio 11

```

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 void scriviSuFile(FILE *fp);
8 void leggiDaFile(FILE *fp);
9
10 int main(){
11     FILE *fp;

```

```

12     int n,i,k;
13     char nomeFile[MAXLEN];
14     printf("Dammi il nome 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)

```

```
58     printf("%d\n",k);
59 }
```

### 1.13 Esercizio 12

```
1  #include <stdio.h>
2  #define MAXLEN 50
3  int main(){
4      char nomeFile[MAXLEN],c;
5
6
7      FILE *fp; //fp è il puntatore ad una stream, ma lo stream non e' creato
8      //Quando creo (apro) lo stream devo specificare se apro in lettura,
9      // scrittura o entrambe le cose
10     //VEDIAMO PRIMA in lettura:
11     printf("Dammi il nome del file da leggere:");
12     scanf("%s",nomeFile);
13     //fscanf(stdin,"%s",nomeFile);
14     fp=fopen(nomeFile,"r");
15     if (fp!=NULL) {//Ho trovato il file da leggere e creato lo stream
16         //Leggo carattere per carattere
17         while (fscanf(fp,"%c",&c)>0) {
18             printf("%c",c);
19         }
20         //ALLA FINE DI TUTTE LE OPERAZIONI, DEVO CHIUDERE LO tream
21         fclose(fp);
22     }
23     else { //NON HO TROVATO IL FILE DA LEGGERE O C'E' STATO UN ERRORE IN
24         //APERTURA
25         printf("FILE NON TROVATO!!");
26     }
27     return 0;
28 }
```

### 1.14 Esercizio 13

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct elem {
5      int k;
6      struct elem *next;
7  };
8
9  struct elem *insElemInTesta(struct elem *top,int k);
```



```

10 struct elem *insElemInCoda(struct elem *top,int k);
11 struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k);
12 struct elem *eliminaLista(struct elem *top);
13 struct elem *cercaElemento(struct elem *top,int k);
14 //In effetti serve solo il precedente!!
15 struct elem *eliminaElem(struct elem *top, struct elem *pre);
16
17 int main(){
18     int k;
19     struct elem *top; //NON E' CREATA LA LISTA
20
21     top=NULL; //CREO UNA LISTA NULLA, VUOTA!!!!
22
23     top=insElemInTesta(top,k); //QUI, CON L'ASSEGNAZIONE, CAMBIA EFFETTIVAMENTE IL TOP
24                             // DELLA LISTA
25     top=insElemInCoda(top,k); //SE CAMBIA IL TOP, VIENE CAMBIATO QUI!!!
26
27     top=eliminaLista(top);
28
29     return 0;
30 }
31 //PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
32 struct elem *eliminaElem(struct elem *top, struct elem *pre){
33     struct elem *c;
34     if (pre==NULL) {
35         c=top;
36         top=top->next;
37         free(c);
38     } else {
39         c= pre->next;
40         pre->next=c->next;
41         free(c);
42     }
43     return top;
44 }
45
46 struct elem *nuovoElemento(int k){
47     struct elem *nuovo= (struct elem *)malloc(sizeof(struct elem));
48     nuovo->k=k;
49     nuovo->next=NULL;
50     return nuovo;
51 }
52 struct elem *insElemInTesta(struct elem *top,int k){
53     struct elem *nuovo= nuovoElemento(k);
54     nuovo->next=top;
55     return nuovo;

```

```

56 }
57 struct elem *insElemInCoda(struct elem *top,int k){
58     struct elem *nuovo= nuovoElemento(k);
59     if (top==NULL) return nuovo;
60     struct elem *curr=top;
61     while (curr->next!=NULL) curr=curr->next; //SEMPRE SICURO CHE curr E' DIVERSO DA NULL
62     curr->next=nuovo;
63     return top;
64 }
65 //pre è l'analogo dell'indice negli array
66 struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k){
67     struct elem *nuovo= nuovoElemento(k);
68     if (pre==NULL) {
69         nuovo->next=top;
70         top=nuovo;
71     }
72     else {
73         nuovo->next=pre->next;
74         pre->next=nuovo;
75     }
76     return top;
77 }
78 struct elem *eliminaLista(struct elem *top){
79     struct elem *c;
80     while (top != NULL){
81         c=top;
82         top=c->next;
83         free(c);
84     }
85     return top;
86 }
87 // --Se il puntatore restituito punta a NULL, l'elemento cercato è in testa (la testa
88 // potrebbe essere NULL)
89 // -- Se il puntatore restituito,p, è diverso da NULL, l'elemento cercato si trova come
90 // next, p->next (p->next potrebbe essere NULL, se non trovo l'elemento)
91 struct elem *cercaElemento(struct elem *top,int k){
92     struct elem *pre=NULL, curr=top;
93     //SCORRO LA LISTA SE SONO SU UN ELEMENTO DIVERSO DA NULL E NON HO
94     // TROVATO k!!!
95     while (curr !=NULL && curr->k!=k) {
96         pre=curr;
97         curr=curr->next;
98     }
99     /* COSI' NON VA BENE
100     while (curr->k!=k && curr !=NULL ) {
101         pre=curr;

```

```

102         curr=curr->next;
103     }
104     */
105     return pre;
106 }

```

## 1.15 Esercizio 14

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  //SUPPONIAMO DI VOLER RISOLVERE IL SEGUENTE PROBLEMA:
4  // Eliminare un elemento da una lista semplicemente concatenata di valore k, se esiste.
5  //VERSIONE 1 di questo problema: la lista non contiene ripetizioni. Ogni valore i è presente
6  //una sola volta nella lista: top--> 23 --> 18 --> 30 -->NULL
7  // (E NON COSI': top --> 23 -->18-->23-->8-->NULL (in questo caso 23 si ripete)
8  struct elem {
9      int k;
10     struct elem *next;
11 };
12
13 //Funzioni di servizio (precedentemente create)*****
14 struct elem *insElemInTesta(struct elem *top,int k);
15
16 struct elem *insElemInCoda(struct elem *top,int k);
17
18 struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k);
19 struct elem *eliminaLista(struct elem *top);
20 struct elem *cercaElemento(struct elem *top,int k);
21 //In effetti serve solo il precedente!!
22 //PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
23 struct elem *eliminaElem(struct elem *top, struct elem *pre);
24 //*****
25 struct elem *riempiLista();
26 struct elem *eliminaKSenzaRipetizioni(struct elem *top,int k);
27 void stampaLista(struct elem *top);
28 int main(){
29     int k;
30     struct elem *top; //NON E' CREATA LA LISTA
31     top=riempiLista();
32     printf("Dammi l'elemento da cancellare:");
33     scanf("%d",&k);
34     top=eliminaKSenzaRipetizioni(top,k);
35     stampaLista(top);
36     top=eliminaLista(top);
37

```

```

38 return 0;
39 }
40 //PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
41 struct elem *eliminaElem(struct elem *top, struct elem *pre){
42     struct elem *c;
43     if (pre==NULL) {
44         c=top;
45         top=top->next;
46         free(c);
47     } else {
48         c= pre->next;
49         pre->next=c->next;
50         free(c);
51     }
52     return top;
53 }
54
55 struct elem *nuovoElemento(int k){
56     struct elem *nuovo= (struct elem *)malloc(sizeof(struct elem));
57     nuovo->k=k;
58     nuovo->next=NULL;
59     return nuovo;
60 }
61 struct elem *insElemInTesta(struct elem *top,int k){
62     struct elem *nuovo= nuovoElemento(k);
63     nuovo->next=top;
64     return nuovo;
65 }
66 struct elem *insElemInCoda(struct elem *top,int k){
67     struct elem *nuovo= nuovoElemento(k);
68     if (top==NULL) return nuovo;
69     struct elem *curr=top;
70     while (curr->next!=NULL) curr=curr->next; //SEMPRE SICURO CHE curr E' DIVERSO DA NULL
71     curr->next=nuovo;
72     return top;
73 }
74 //pre è l'analogo dell'indice negli array
75 struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k){
76     struct elem *nuovo= nuovoElemento(k);
77     if (pre==NULL) {
78         nuovo->next=top;
79         top=nuovo;
80     }
81     else {
82         nuovo->next=pre->next;
83         pre->next=nuovo;

```

```

84     }
85     return top;
86 }
87 struct elem *eliminaLista(struct elem *top){
88     struct elem *c;
89     while (top != NULL){
90         c=top;
91         top=c->next;
92         free(c);
93     }
94     return top;
95 }
96 // --Se il puntatore restituito punta a NULL, l'elemento cercato è in testa (la testa
97 // potrebbe essere NULL)
98 // -- Se il puntatore restituito,p, è diverso da NULL, l'elemento cercato si trova come
99 // next, p->next (p->next potrebbe essere NULL, se non trovo l'elemento)
100 struct elem *cercaElemento(struct elem *top,int k){
101     struct elem *pre=NULL, *curr=top;
102     //SCORRO LA LISTA SE SONO SU UN ELEMENTO DIVERSO DA NULL E NON HO
103     // TROVATO k!!!
104     while (curr !=NULL && curr->k!=k) {
105         pre=curr;
106         curr=curr->next;
107     }
108     /* COSI' NON VA BENE
109     while (curr->k!=k && curr !=NULL ) {
110         pre=curr;
111         curr=curr->next;
112     }
113     */
114     return pre;
115 }
116 struct elem *riempiLista(){
117     struct elem *top=NULL;
118     int n,i,k;
119     printf("Quanti elementi vuoi inserire?");
120     scanf("%d",&n);
121     for (i=0;i<n;++i) {
122         printf("Dammi l'elemento %d:",i+1);
123         scanf("%d",&k);
124         top=insElemInTesta(top,k);
125     }
126     return top;
127 }
128 struct elem *eliminaKSenzaRipetizioni(struct elem *top,int k){
129     struct elem *pre=NULL,*tmp;

```

```

130     if (top!=NULL) {
131         pre=cercaElemento(top,k);
132         if (pre==NULL) {
133             tmp=top;
134             top=top->next;
135             free(tmp);
136         }
137         else if (pre->next!=NULL) {
138             //tmp è il nodo da eliminare
139             tmp=pre->next;
140             pre->next=tmp->next;
141             free(tmp);
142         }
143     }
144 }
145 /*VERSIONE CON CHIAMATA DI SOTTOFUNZIONE
146 if (top!=NULL) {
147     pre=cercaElemento(top,k);
148     if (pre==NULL) top=eliminaElem(top,pre);
149     else if (pre->next!=NULL) top=eliminaElem(top,pre);
150 }
151 */
152 return top;
153 }
154 void stampaLista(struct elem *top){
155     while (top!=NULL) {
156         printf("%d -->",top->k);
157         top=top->next;
158     }
159     printf("NULL\n");
160 }

```

## 1.16 Esercizio 15

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  //SUPPONIAMO DI VOLER RISOLVERE IL SEGUENTE PROBLEMA:
4  // Eliminare un elemento da una lista semplicemente concatenata di valore k, se esiste.
5  //VERSIONE 2 di questo problema: la lista ncontiene ripetizioni. Ogni valore i può essere //
6  // presente più volte.
7  struct elem {
8      int k;
9      struct elem *next;
10 };
11

```

```

12 //Funzioni di servizio (precedentemente create)*****
13 struct elem *insElemInTesta(struct elem *top,int k);
14
15 struct elem *insElemInCoda(struct elem *top,int k);
16
17 struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k);
18 struct elem *eliminaLista(struct elem *top);
19 struct elem *cercaElemento(struct elem *top,int k);
20 //In effetti serve solo il precedente!!
21 //PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
22 struct elem *eliminaElem(struct elem *top, struct elem *pre);
23 //*****
24 struct elem *riempiLista();
25 struct elem *eliminaKConRipetizioni(struct elem *top,int k);
26
27 //Funzione che richiama altre funzioni già realizzate
28 struct elem *eliminaKConRipetizioni_2(struct elem *top,int k);
29 void stampaLista(struct elem *top);
30 int main(){
31     int k;
32     struct elem *top; //NON E' CREATA LA LISTA
33     top=riempiLista();
34     printf("Dammi l'elemento da cancellare:");
35     scanf("%d",&k);
36     //top=eliminaKConRipetizioni(top,k);
37     top=eliminaKConRipetizioni_2(top,k);
38
39     stampaLista(top);
40     top=eliminaLista(top);
41
42     return 0;
43 }
44 //PRE-CONDIZIONE: l'elemento da eliminare sia sempre diverso da NULL
45 struct elem *eliminaElem(struct elem *top, struct elem *pre){
46     struct elem *c;
47     if (pre==NULL) {
48         c=top;
49         top=top->next;
50         free(c);
51     } else {
52         c= pre->next;
53         pre->next=c->next;
54         free(c);
55     }
56     return top;
57 }

```

```

58
59 struct elem *nuovoElemento(int k){
60     struct elem *nuovo= (struct elem *)malloc(sizeof(struct elem));
61     nuovo->k=k;
62     nuovo->next=NULL;
63     return nuovo;
64 }
65 struct elem *insElemInTesta(struct elem *top,int k){
66     struct elem *nuovo= nuovoElemento(k);
67     nuovo->next=top;
68     return nuovo;
69 }
70 struct elem *insElemInCoda(struct elem *top,int k){
71     struct elem *nuovo= nuovoElemento(k);
72     if (top==NULL) return nuovo;
73     struct elem *curr=top;
74     while (curr->next!=NULL) curr=curr->next; //SEMPRE SICURO CHE curr E' DIVERSO DA NULL
75     curr->next=nuovo;
76     return top;
77 }
78 //pre è l'analogo dell'indice negli array
79 struct elem *insElemInMezzo(struct elem *top,struct elem *pre,int k){
80     struct elem *nuovo= nuovoElemento(k);
81     if (pre==NULL) {
82         nuovo->next=top;
83         top=nuovo;
84     }
85     else {
86         nuovo->next=pre->next;
87         pre->next=nuovo;
88     }
89     return top;
90 }
91 struct elem *eliminaLista(struct elem *top){
92     struct elem *c;
93     while (top != NULL){
94         c=top;
95         top=c->next;
96         free(c);
97     }
98     return top;
99 }
100 // --Se il puntatore restituito punta a NULL, l'elemento cercato è in testa (la testa
101 // potrebbe essere NULL)
102 // -- Se il puntatore restituito,p, è diverso da NULL, l'elemento cercato si trova come
103 // next, p->next (p->next potrebbe essere NULL, se non trovo l'elemento)

```



```

104 struct elem *cercaElemento(struct elem *top,int k){
105     struct elem *pre=NULL, *curr=top;
106     //SCORRO LA LISTA SE SONO SU UN ELEMENTO DIVERSO DA NULL E NON HO
107     // TROVATO k!!!
108     while (curr !=NULL && curr->k!=k) {
109         pre=curr;
110         curr=curr->next;
111     }
112     /* COSI' NON VA BENE
113     while (curr->k!=k && curr !=NULL ) {
114         pre=curr;
115         curr=curr->next;
116     }
117     */
118     return pre;
119 }
120 struct elem *riempiLista(){
121     struct elem *top=NULL;
122     int n,i,k;
123     printf("Quanti elementi vuoi inserire?");
124     scanf("%d",&n);
125     for (i=0;i<n;++i) {
126         printf("Dammi l'elemento %d:",i+1);
127         scanf("%d",&k);
128         top=insElemInTesta(top,k);
129     }
130     return top;
131 }
132 struct elem *eliminaKConRipetizioni(struct elem *top,int k){
133     struct elem *pre=NULL, *curr,*tmp;
134     curr=top;
135     while (curr!=NULL) {
136         if (curr->k == k) {
137             if (pre==NULL) { //ELEMENTO DA ELIMINARE E' IN TESTA
138                 tmp=curr;
139                 curr=curr->next;
140                 top=curr; //UNICO CASO IN CUI CAMBIA IL TOP DELLA LISTA
141                 free(tmp);
142             }
143             else {
144                 pre->next=curr->next;
145                 free(curr);
146                 curr=pre->next;
147             }
148         }
149         else {

```

```

150         pre=curr;
151         curr=curr->next;
152     }
153 }
154 return top;
155 }
156
157
158 //Funzione che richiama altre funzioni già realizzate
159 struct elem *eliminaKConRipetizioni_2(struct elem *top,int k){
160     struct elem *pre;
161     int flag=1;
162     while (top !=NULL && flag==1) {
163         pre=cercaElemento(top,k); //QUI RICERCO SEMPRE DA TOP, E' DISPENDIOSO!!!
164                                     // COME MODIFICARE???
165                                     //ESERCIZIO DA FARE A CASA
166         if (pre==NULL)
167             top=eliminaElem(top,pre);
168         else if (pre->next!=NULL) top=eliminaElem(top,pre);
169         else flag=0; //NON TROVO ELEMENTI, ESCO DA WHILE!!!!
170     }
171     return top;
172 }
173
174 void stampaLista(struct elem *top){
175     while (top!=NULL) {
176         printf("%d -->",top->k);
177         top=top->next;
178     }
179     printf("NULL\n");
180 }

```

## 1.17 Esercizio 16

```

1  #include <stdio.h>
2  #define MAXLEN 100
3
4  void mergeSort(int A[], int n);
5
6
7  int main() {
8      int V[MAXLEN];
9      int n,i;
10     printf("Quanti numeri:");
11     scanf("%d",&n);

```

```

12     for (i=0;i<n;++i) {
13         printf("Dammi V[%d]=",i);
14         scanf("%d",&V[i]);
15     }
16     mergeSort(V,n);
17     for (i=0;i<n;++i) {
18         printf("V[%d]=%d\n",i,V[i]);
19     }
20
21     return 0;
22 }
23
24 void merge(int V[],int centro,int n);
25
26 void mergeSort(int V[], int n){
27     //CASO BASE n=1, IMPLICITO
28     if (n>1) {
29         //DIVIDO IL PROBLEMA IN SOTTOPROBLEMI
30         int centro=n/2;
31         //TROVO LE SOLUZIONI DEI SOTTOPROBLEMI
32         mergeSort(V, centro);
33         mergeSort(&V[centro], n-centro);
34         //COMBINO LE SOLUZIONI TROVATE
35         merge(V,centro,n);
36     }
37 }
38 void merge(int V[],int centro,int n){
39     int *A, *B;
40     int C[MAXLEN];
41     int m1,m2; //m1 e m2 le dimensioni dei due vettori già ordinati
42     m1=centro;
43     m2=n-centro;
44     int i=0,j=0,k=0;
45     //I DUE ARRAYS A e B GIÀ ORDINATI
46     A=&V[0];
47     B=&V[centro];
48
49     while (i<m1 && j < m2) {
50         if (A[i] < B[j]) {
51             C[k]=A[i];
52             i=i+1;
53         }
54         else {
55             C[k]=B[j];
56             j=j+1;
57         }

```

```

58         k=k+1;
59     }
60     //AL TERMINE DEL PRECEDENTE WHILE HO: 0 i>=m1 e j>=m2, OPPURE SOLO UNO E' maggiore
61     //O UGUALE
62     while ( i < m1) {
63         C[k]=A[i];
64         k=k+1;
65         i=i+1;
66     }
67
68     while ( j < m2) {
69         C[k]=B[j];
70         k=k+1;
71         j=j+1;
72     }
73
74     for (i=0;i<n;++i) V[i]=C[i];
75 }

```

## 1.18 Esercizio 17

```

1  /*
2   fattorian di n, con n >=0, che in genere è indicato con n! risulta essere
3   n!= 1 se n==0, oppure n!=n * (n-1)! altrimenti
4   */
5
6  #include <stdio.h>
7  #define MAXLEN 10
8  int fatt(int n);
9  //Funzione fattoriale di debug
10 int fatt_debug(int n);
11 int main() {
12     int n,res;
13     printf("Dammi un numero intero maggiore o uguale a zero:");
14     scanf("%d",&n);
15     //res=fatt(n);
16     res=fatt_debug(n);
17     printf("Fattoriale uguale a: %d\n",res);
18     return 0;
19 }
20
21 int fatt(int n){
22     int res=1;
23     int res_piu_piccolo;
24     if (n==0) //Passo base: risolvo il problema direttamente e

```

```

25         // "facilmente"
26         res=1;
27     else //Passo ricorsivo
28     {
29         //Prendo la soluzione dello stesso problema, ma di
30         // dimensioni più piccole, cioè n-1
31         res_piu_piccolo=fatt(n-1);
32         //res_piu_piccolo è il fattoriale di (n-1), cioè il //
33         // prodotto (n-1)*(n-2)*.....*1
34         // Risolvo il problema con dimensione n, sulla base
35         // (usando) la soluzione del problema più piccolo
36         res= n *res_piu_piccolo; //Questa è la chiave della
37                                 // ricorsione!!!!!!!!!!
38         //res= n*(n-1)*(n-2)*.....*1, quindi
39         //res e' il fattoriale di n
40     }
41     return res;
42 }
43
44 //Altro esempio:
45
46 int massimoR(int a[],int n) {
47     int res=0;
48     int max;
49     if (n=1) res=a[0];
50     else
51     {
52         max=massimoR(a,n-1);
53         if (a[n-1]>max) res=a[n-1];
54         else res=max;
55     }
56
57     return res;
58 }
59
60
61 int fatt_debug(int n){
62     int res=1;
63     int ferma; //Variabile di appoggio
64     if (n==0){ //Passo base: risolvo il problema direttamente e
65         // "facilmente"
66         res=1;
67         printf("CASO BASE n:%d  res:%d \n",n,res);
68         printf("Vai avanti?");
69         scanf("%d",&ferma);
70     }

```

```

71     else //Passo ricorsivo
72     {
73         printf("CASO ricorsivo prima della chiamata ricorsiva n:%d  res:%d \n",n,res);
74         printf("Vai avanti?");
75         scanf("%d",&ferma);
76         res= n * fatt_debug(n-1);
77         printf("CASO ricorsivo dopo la chiamata ricorsiva n:%d  res:%d \n",n,res);
78         printf("Vai avanti?");
79         scanf("%d",&ferma);
80
81     }
82     return res;
83 }

```

## 1.19 Esercizio 18

```

1 //1) Voglio eliminare da un lista semplicemente concatenata tutti gli elementi mionori
2 //di un dato k
3 //2) Suppongo di avere una lista di interi semplicemente concatenata senza ripetizioni.
4 // Voglio eleiminare un elemento di valore k, se esiste
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 struct elem {
9     int k;
10    struct elem *next;
11 };
12
13 struct elem *riempiLista();
14 struct elem *cancellaMinoriDik(struct elem *top,int k);
15 struct elem *cancellaElementok(struct elem *top,int k);
16
17 void stampaLista(struct elem *top);
18 int main(){
19     struct elem *top;
20     int k;
21     top=riempiLista();
22     stampaLista(top);
23     printf("Dammi k:");
24     scanf("%d",&k);
25     top=cancellaMinoriDik(top,k);
26     stampaLista(top);
27     return 0;
28 }
29 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

```

```

76         //la soluzione in tmp;
77         tmp=cancellaElementok(tmp->next,k);
78         //COSTRUISCO LA SOLUZIONE PER P(n);
79         top->next=tmp;
80     }
81 }
82 return top;
83 }

```

## 1.20 Esercizio 19

```

1 // Data una lista semplicemente concatenata di interi, voglio invertirla, cioè
2 //il primo elemento diventa l'ultimo, il secondo il penultimo e così via.
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 struct elem {
7     int k;
8     struct elem *next;
9 };
10
11 struct elem *riempiLista();
12 struct elem *cancellaMinoriDik(struct elem *top,int k);
13 struct elem *cancellaElementok(struct elem *top,int k);
14 struct elem *invertiLista(struct elem *top);
15
16 void stampaLista(struct elem *top);
17 int main(){
18     struct elem *top;
19     int k;
20     top=riempiLista();
21     stampaLista(top);
22     printf("Dammi k:");
23     scanf("%d",&k);
24     top=cancellaMinoriDik(top,k);
25     stampaLista(top);
26     return 0;
27 }
28 struct elem *riempiLista(){
29     struct elem *top=NULL;
30     return top;
31 }
32 //P(n)=Eliminare da un lista semplicemente concatenata di dimensione n,
33 // tutti gli elementi mionori di un dato k
34

```



```

35 struct elem *cancellaMinoriDik(struct elem *top,int k){
36     //CASO BASE, top==NULL, implicito
37     struct elem *tmp;
38     if (top!=NULL) {
39         //ASSUMMO DI SAPER RISOLVERE IL PROBLEMA P(n-1), cioè
40         //P(n)=Eliminare da un lista semplicemente concatenata di dimensione n-1,
41         // tutti gli elementi mionori di un dato k
42         //tmp mantiene la soluzione, cioè sarà il puntatore alla testa della
43         //lista in cui sono stati eliminati gli elementi minori di k,
44         //ecceto la testa iniziale (top)
45         tmp=cancellaMinoriDik(top->next,k);
46         //ADESSO COSTRUISCO LA SOLUZIONE PER P(n)
47         if (top->k<k) {
48             free(top);
49             top=tmp;
50         }
51         else top->next=tmp;
52     }
53     return top;
54 }
55 void stampaLista(struct elem *top){
56 }
57
58 //P(n)= Data una lista di interi semplicemente concatenata e senza ripetizioni,
59 // con n elementi, Elimino un elemento di valore k, se esiste
60
61 struct elem *cancellaElementok(struct elem *top,int k){
62     struct elem *tmp;
63     //Passo base, top==NULL, implicito
64
65     if (top!=NULL) {
66         //Se è in testa devo solo cancellare la testa e non devo fare più nulla,
67         // perchè è una lista SENZA ripetizioni
68         if (top->k==k) {
69             tmp=top->next;
70             free(top);
71             top=tmp;
72         }
73         else {
74             //SUPPPONGO DI SAPER RISOLVERE IL PROBLEMA P(n-1) e mettere
75             //la soluzione in tmp;
76             tmp=cancellaElementok(tmp->next,k);
77             //COSTRUISCO LA SOLUZIONE PER P(n);
78             top->next=tmp;
79         }
80     }

```

```

81     return top;
82 }
83 // P(n)=Data una lista semplicemente concatenata di n interi, voglio invertirla
84
85 struct elem *invertiLista(struct elem *top){
86     //caso base implicito (top==NULL)
87     struct elem *tmp,*curr;
88     if (top!=NULL) {
89         //Assumo che tmp contenga la soluzione per la lista senza la testa,
90         //cioè la soluzione del problema P(n-1)=Data una lista semplicemente //
91         //concatenata di n-1 interi, voglio invertirla
92         tmp=invertiLista(top->next);
93         //COSTRUISCO LA SOLUZIONE PER P(n)
94         if (tmp!=NULL) {
95             curr=tmp;
96             while (curr->next!=NULL) curr=curr->next;
97             curr->next=top;
98             top->next=NULL;
99             top=tmp;
100         }
101     }
102     return top;
103 }

```

## 1.21 Esercizio 20

```

1 //Dato un array di n elementi interi, calcolare il massimo
2 #include <stdio.h>
3 #define MAXELEM 50
4 int riempiArray(int a[]);
5 int calcolaMassimo(int a[], int n);
6 int calcolaMassimo2(int a[], int n);
7
8 int main() {
9     int a[MAXELEM];
10    int n,m,m2;
11    n=riempiArray(a);
12    if (n>0) {
13        m=calcolaMassimo(a,n);
14        m2=calcolaMassimo2(a,n);
15
16
17    printf("Il massimo e':%d %d\n",m,m2);
18    }
19    else printf("Non ho elementi!\n");

```

```

20     return 0;
21 }
22
23 int riempiArray(int a[]){
24     int n,i;
25     printf("Dammi il numero di elementi:");
26     scanf("%d",&n);
27     for (i=0;i<n;++i) {
28         printf("Dammi a[%d]=",i);
29         scanf("%d",&a[i]);
30     }
31     return n;
32 }
33 //PRECONDIZIONE: n>0
34 //a è l'array, n è il numero di elementi nell'array
35 int calcolaMassimo(int a[], int n){
36     int m;
37     if (n==1) {
38         m=a[0];
39     }
40     else {
41         //SUPPONGO di Avere la soluzione per lo stesso problema
42         //ma con un elemento in meno, P(n-1)
43         //Ad esempio, tolgo l'ultimo elemento
44         m=calcolaMassimo(a,n-1);
45         //m è il massimo tra tutti gli elementi, eccetto l'ultimo a[n-1]
46         //Quindi prendo m e costruisco la soluzione per P(n)
47         if (a[n-1]>m) m=a[n-1];
48         //Questo m è il massimo tra tutti gli elementi!!!!
49     }
50     return m;
51 }
52 //PRECONDIZIONE: n>0
53 //a è l'array, n è il numero di elementi nell'array
54 int calcolaMassimo2(int a[], int n){
55     int m;
56     if (n==1) m=a[0];
57     else {
58         //SUPPONGO di Avere la soluzione per lo stesso problema
59         //ma con un elemento in meno, P(n-1)
60         //Ad esempio, tolgo il primo elemento
61         m=calcolaMassimo2(&a[1],n-1);
62         //m è il massimo tra tutti gli elementi, eccetto il primo
63         //Quindi prendo m e costruisco la soluzione per P(n)
64         if (a[0]>m) m=a[0];
65         //Questo m è il massimo tra tutti gli elementi!!!!

```

```

66     }
67     return m;
68 }

```

## 1.22 Esercizio 21

```

1  /*
2   Ho una lista , un puntatore ad un nodo e voglio spostarlo In testa.
3  */
4  #include <stdio.h>
5  #include <stdlib.h>
6  struct elem {
7      int k;
8      struct elem *next;
9  };
10
11 struct elem *newElem(int k);
12 struct elem *insInCodaR(struct elem *top, int k);
13 struct elem *insInOrdineR(struct elem *top, int k);
14 struct elem *insInOrdine(struct elem *top, int k);
15 struct elem *spostoElemInTesta(struct elem *top, struct elem *p);
16 void stampaLista(struct elem *top);
17 void stampaListaR1(struct elem *top);
18 void stampaListaR2(struct elem *top);
19
20
21 int main(){
22     int n,i,k;
23     struct elem *top=NULL;
24     printf("Quanti elementi vuoi inserire:");
25     scanf("%d",&n);
26     for (i=0;i<n;++i) {
27         printf("Dammi l'elemento da inserire:");
28         scanf("%d",&k);
29         top=insInOrdineR(top,k);
30     }
31     stampaListaR1(top);
32     stampaListaR2(top);
33     return 0;
34 }
35
36 //P(n)= Ho una lista di n elementi ed un puntatore p ad un
37 //elemento e voglio mettere p in testa
38 struct elem *spostoElemInTesta(struct elem *top, struct elem *p){
39     //P(n=0)

```

```

40     struct elem *tmp;
41     if (top!=NULL && p!=NULL) {
42         if (p!= top) {
43             //Soluzione per P(n-1)
44             tmp= spostoElemInTesta(top->next,p);
45             //Assumo che p è in testa alla nuova lista
46             //tmp
47             //Allora trovo soluzione per P(n)
48             top->next=tmp->next;
49             tmp->next=top;
50             top=tmp;
51         }
52     }
53
54
55
56     return top;
57 }
58
59 struct elem *newElem(int k){
60     struct elem *nuovo=NULL;
61     nuovo=(struct elem *)malloc(sizeof(struct elem));
62     nuovo->k=k;
63     nuovo->next=NULL;
64     return nuovo;
65 }
66
67 //Il problem P(n)= "Inserire il valore k in coda ad un lista di n elementi"
68 struct elem *insInOrdineR(struct elem *top, int k){
69     // P(n) =Ho una lista ordinata (crescente) di n elementi e voglio
70     // inserire un elemento k in modo tale che
71     //la lista resti ordinata (sempre crescente).
72
73     struct elem *tmp;
74     //P(n=0)
75     if (top==NULL) {
76         top=newElem(k);
77     } else {
78         if (k< top->k) {
79             //Risolvo banalmente senza chiamata
80             //ricorsiva
81             tmp=newElem(k);
82             tmp->next=top;
83             top=tmp;
84         } else
85         {

```

```

86         //Problema P(n-1) lo so risolvere:
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         //cioè 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

```
1 //Lezione Lab Prog Gr. 2B del 2021.04.14
2 #include <stdio.h>
3 #define MAXLEN 10
4 //NON HO DEFINITO NESSUNA VARIABILE, MA HO ASSOCIATO ALL'IDENTIFICATORE miaStruc
5 // (si può scegliere qualunque nome all'interno di quelli ammissibili)
6 // UNA struct come quella definita
7 struct miaStruct {
8     int n; //Capo della struct: nome: n tipo: int
9     char c[MAXLEN]; //Capo della struct: nome: c tipo: stringa
10    float x; //Capo della struct: nome: x tipo: float
11 };
12
13 void stampaMiaStruct( struct miaStruct ss);
14 struct miaStruct riempiMiaStruct();
15 int main(){
16     struct miaStruct rr; //QUI e' definita la variabile rr di tipo struct miaStruct
17     rr=riempiMiaStruct();
18
19     stampaMiaStruct(rr); //ANCHE IN QUESTO CASO PASSAGGIO PER VALORE
20     struct miaStruct rr2; //QUI ho definito un' altra variabile di tipo struct
21     rr2=riempiMiaStruct();
22
23     stampaMiaStruct(rr2); //ANCHE IN QUESTO CASO PASSAGGIO PER VALORE
24 return 0;
25 }
26 //ANCHE IN QUESTO CASO PASSAGGIO PER VALORE:
27 //E' CREATO IL PARAMETRO FORMALE ss (che è una struct miaStruct)
28 // Al parametro formale è assegnato il valore del parametro attuale (ad esempio rr del main),
29 // cioè ai campi di ss sono assegnati i valori dei rispettivi campi del parametro attuale,
30 // ad esempio quelli di rr
31 void stampaMiaStruct( struct miaStruct ss){
32     printf("c: %s x: %f n:%d\n",ss.c,ss.x,ss.n);
33 }
34
35 struct miaStruct riempiMiaStruct(){
36     struct miaStruct ss; //Variabile struct locale alla funzione
37     printf("Dammi carattere, intero, float:");
38     scanf("%s %d %f",ss.c,&(ss.n),&(ss.x));
39     return ss;
40 }
```

### 1.24 Esercizio 23

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



```

2 #include <stdio.h>
3 #include <stdlib.h> //necessario per utilizzo malloc, calloc e free
4 //SUPPONIAMO DI VOLER RIEMPIRE E STAMPARE UN ARRAY CREATO CON MEMORIA DINAMICA
5
6
7 //FUNZIONE RESTITUISCE L'ARRAY CREATO DINAMICAMENTE, CIOE' RESTITUISCE L'INDIRIZZO
8 //DI MEMORIA DEL PRIMO ELEMENTO (QUESTO E' BENE FARLO SOLO PER MEMORIA ALLOCATA
9 //DINAMICAMENTE)
10 //HA COME ARGOMENTO l'indirizzo di memoria di una variabile in cui metterò
11 //il valore delle dimensioni dell'array creato dinamicamente
12 int *riempiArrayDinamico(int *pn);
13
14 //Dato l'array A di dimensioni n, stampo l'array
15 void stampaArray(int *A,int n);
16
17 //Libero la memoria per l'array allocato dinamicamente
18 void liberaArray(int *A);
19
20 int main(){
21     int *V; //Puntatore ad un intero
22     int n; // n variabile che mantiene l'informazione
23         //sulle dimensioni dell'array elementi
24     V=riempiArrayDinamico(&n);
25     if (V!=NULL) {
26         stampaArray(V,n); //Stampo array
27         liberaArray(V); //Libero la memoria creata dinamicamente
28     }
29     else printf("ARRAY NON CREATO PER MEMORIA INSUFFICIENTE!!!");
30 return 0;
31 }
32
33 int *riempiArrayDinamico(int *pn){
34     int n,i;
35     //size_t dipende dal sistema operativo e versione c,
36     //in genere definito come unsigned int
37     //void* calloc (size_t num, size_t size); Alloco num elementi di dimensione size byte.
38     //
39     //                                Memoria contigua e azzero la memoria allocata
40     //void* malloc (size_t size);      Alloco size byte (memoria contigua)
41     printf("Quanti elementi vuoi inserire?");
42     scanf("%d",&n);
43     //int *A=(int *)calloc(n,sizeof(int)); //int *A=(int *)calloc(n,sizeof(n))
44     //UNA ALTERNATIVA
45     int *A= (int *) malloc(n*sizeof(int)); //Con malloc devo stare attento alla
46     //inizializzazione, perche' non ho una
47     //inizializzazione a zero "automatica"
48
49     if (A!=NULL) {

```

```

48         for (i=0;i<n;++i) {
49             printf("dammi A[%d]=",i);
50             scanf("%d",&A[i]);
51         }
52         n=0;
53     }
54     *pn=n; // Accedo al contenuto presente nell'indirizzo di memoria mantenut in pn
55           // e gli assegno il valore di n
56     return A; //Restituisco il valore contenuto in A
57 }
58 void stampaArray(int *A,int n){
59     int i;
60     for (i=0;i<n;++i)
61         printf("A[%d]=%d\n",i,A[i]);
62 }
63 void liberaArray(int *A){
64     //void free(void *ptr)
65     free(A);
66 }
67 }

```

## 1.25 Esercizio 24

```

1 //Una variabile e' sempre definita da: un tipo, nome, un valore ed un indirizzo di memoria
2 #include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
3                      di questo file*/
4 void fun(int);
5 int main() {
6     int x=0;
7     printf("x address: %p x value=%d\n",&x, x);
8     //p --> x
9     int *p=&x;
10    printf("p address: %p p value=%p\n",&p, p);
11
12    printf("Dammi x:");
13    scanf("%d",&x);
14    if (x>0) {
15        int x=0;
16        printf("Indirizzo nuovo x:%p\n",&x);
17        //Ho nascosto l'accesso alla variabile x definita precedentemente
18        *p=10;
19    }
20    printf("x address: %p x value=%d\n",&x, x);
21    return 0;
22 }

```

## 1.26 Esercizio 25

```
1 //Importanza della inizializzazione di una variabile.
2 //Verifica sperimentale che le variabile automatiche sono caricate in memoria
3 //quando il loader carica il programma in memoria
4 #include <stdio.h> /*header file: aggiungo, includo, il file di testo stdio.h all'inizio
5                      di questo file*/
6 void fun(int);
7
8 int main() {
9     int x;
10    printf("Dammi x:");
11    scanf("%d",&x); //x=1, x=2
12    fun(x);
13    printf("Dammi x:");
14    scanf("%d",&x); //x=2, x=1
15    fun(x);
16    return 0;
17 }
18
19
20 void fun(int x){
21     //La variabile a non e' inizializzata!!!
22     int a;
23     printf("indirizzo a:%p", &a);
24     if (x==1) a=10; else a=a+1;
25     printf("a:%d\n",a); //x=1,Stampo 10, x=2,Stampo 11
26                          //x=2, Stampo ???, x=1, Stampo 10
27 }
```

## 1.27 Esercizio 26

```
1 /*Alcune note sulla creazione di matrici in maniera automatica e dinamica*/
2 #include <stdio.h>
3 #include <stdlib.h> //Da includere se uso malloc o calloc
4 #define MAXR 10
5 #define MAXC 10
6 //Creazione matrici dinamiche (tre modi diversi)
7 int **creaMatriceDinamica1(int r,int c);
8 int **creaMatriceDinamica2(int r,int c);
9 int *creaMatriceDinamica3(int r,int c);
10
11 //Esempio riempimento matrice automatica
12 void riempiMatrice(int M[][MAXC], int *pr,int *pc);
13 //Esempio riempimento matrice dinamica
14 void riempiMatriceDinamica1(int **M, int maxR,int maxC,int *pr,int *pc);
```

```

15 void riempiMatriceDinamica2(int **M, int maxR,int maxC,int *pr,int *pc);
16 void riempiMatriceDinamica3(int *M, int maxR,int maxC,int *pr,int *pc);
17
18 //Esempio stampa di una matrice automatica
19 void stampaMatrice(int M[][MAXC], int r,int c);
20 //Esempio stampa di una matrice dinamica (tre modi diversi, uno per ogni tipo di matrice)
21 void stampaMatriceDinamica1(int **M, int r,int c);
22 void stampaMatriceDinamica2(int **M,int r,int c);
23 void stampaMatriceDinamica3(int *M,int r,int c);
24
25 //Liberare memoria per una matrice dinamica (tre modi diversi, uno per ogni tipo di matrice)
26 void liberaMatriceDinamica1(int **M, int r);
27 void liberaMatriceDinamica2(int **M);
28 void liberaMatriceDinamica3(int *M);
29
30
31 int main(){
32     int r,c,maxR,maxC;
33     int M[MAXR][MAXC]; //CREAZIONE MATRICE IN MEMORIA AUTOMATICA
34     int **M1,**M2;
35     int *M3;
36     /* //PER CAPIRE COME E' ORGANIZZATO M
37     //UN UNICO ARRAY di rxc elementi. M definito come un
38     //puntatore ad una array di MAXC elementi
39     for (r=0;r<MAXR;++r)
40         printf("r=%d M value:%p &M[r][0]=%p M[r]=%p \n",r, M, &M[r][0],M[r]);
41     //Con l'algebra dei puntatori mi sposto di MAXC blocchi alla volta
42     for (r=0;r<MAXR;++r)
43         printf("%p %p\n", M+r,&M[r][0]);
44     /*
45     */
46     printf("Matrice memoria automatica:\n");
47     riempiMatrice(M, &r,&c); //r e c manterranno le effettive dimensioni della matrice
48     stampaMatrice(M,r,c);
49     printf("Dammi i valori di maxR e maxC per le matrici dinamiche:");
50     scanf("%d %d",&maxR,&maxC);
51
52     printf("Matrice memoria dinamica 1\n");
53     M1=creaMatriceDinamica1(maxR,maxC);
54     riempiMatriceDinamica1(M1,maxR,maxC,&r,&c);
55     stampaMatriceDinamica1(M1,r,c);
56     liberaMatriceDinamica1(M1, r);
57
58     printf("Matrice memoria dinamica 2\n");
59     M2=creaMatriceDinamica2(maxR,maxC);
60     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
96 int *creaMatriceDinamica3(int r,int c){
97     int i;
98     int *M= (int *)malloc(r*c*sizeof(int));
99     // In maniera equivalente, con l'unica differenza che inizializzo a zero
100    // tutti gli elementi:
101    // int *M= (int *)calloc(r*c,sizeof(int));
102    return M;
103 }
104
105 void riempiMatrice(int M[] [MAXC], int *pr,int *pc){
106     int r,c,i,j;
107     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){

```

```

199     int i,j;
200     for (i=0;i<r;++i){
201         printf("\n");
202         for (j=0;j<c;++j)
203             printf("M[%d] [%d]= %d ",i,j,M[i*c+j]);
204     }
205     printf("\n");
206
207 }
208 void liberaMatriceDinamica1(int **M, int r){
209     int i;
210     for (i=0;i<r;++i) free(M[i]);
211     free(M);
212 }
213
214 void liberaMatriceDinamica2(int **M){
215     int i;
216     free(M[0]);
217     free(M);
218 }
219 void liberaMatriceDinamica3(int *M){
220     free(M);
221 }

```

## 1.28 Esercizio 27

```

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

```



```

20     int flag;
21     struct schedaPersona schedario[MAXLEN];
22     int nPersone=0; int i;
23     char cog[MAXS];
24     do {
25         printf("Cosa vuoi fare? 0: esci, 1: inserisci, 2: elimina 3:modifica 4:stampa");
26         scanf("%d",&flag);
27         switch (flag) {
28             case 1: nPersone=inseriscoPersona(schedario,nPersone);
29                     break;
30             case 2: printf("Dammi il cognome della persona da eliminare");
31                     scanf("%s",cog);
32                     i=trovaPersoneCognome(schedario,nPersone,cog);
33                     if (i>=0)
34                         nPersone=eliminaPersona(schedario,nPersone,i);
35                     break;
36             case 3:
37                 printf("Dammi il cognome della persona da modificare");
38                 scanf("%s",cog);
39                 i=trovaPersoneCognome(schedario,nPersone,cog);
40                 if (i>=0)
41                     modificaPersona(schedario,nPersone,i);
42                 break;
43             case 4: stampaArray(schedario,nPersone);
44         }
45     } while (flag>0);
46     return 0;
47 }
48
49 struct schedaPersona creoPersona();
50 int inseriscoPersona(struct schedaPersona *schedario, int n){
51     schedario[n]=creoPersona();
52     n=n+1;
53     return n;
54 }
55
56 struct schedaPersona creoPersona(){
57     struct schedaPersona s;
58     printf("Dammi il cognome:");
59     scanf("%s",s.cognome);
60     printf("Dammi il nome:");
61     scanf("%s",s.nome);
62     printf("Dammi l'eta':");
63     scanf("%d",&(s.eta));
64     printf("Dammi il luogo di nascita:");
65     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,
84             schedario[i].nome,schedario[i].eta,schedario[i].luogoDiNascita,schedario[i].reddito
85             );
86     }
87 }

```

## 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() {

```

```

21     int flag;
22     struct schedaPersona *schedario[MAXLEN]; //Array di puntatori a struct schedaPersona
23
24     int nPersone=0; int i;
25     char cog[MAXS];
26     for (i=0;i<MAXLEN;++i) schedario[i]=NULL;
27     do {
28         printf("Cosa vuoi fare? 0: esci, 1: inserisci, 2: elimina 3:modifica 4:stampa");
29         scanf("%d",&flag);
30         switch (flag) {
31             case 1: nPersone=inseriscoPersona(schedario,nPersone);
32                     break;
33             case 2: printf("Dammi il cognome della persona da eliminare");
34                     scanf("%s",cog);
35                     i=trovaPersoneCognome(schedario,nPersone,cog);
36                     if (i>=0)
37                         nPersone=eliminaPersona(schedario,nPersone,i);
38                     break;
39             case 3:
40                 printf("Dammi il cognome della persona da modificare");
41                 scanf("%s",cog);
42                 i=trovaPersoneCognome(schedario,nPersone,cog);
43                 if (i>=0)
44                     modificaPersona(schedario,nPersone,i);
45                 break;
46             case 4: stampaArray(schedario,nPersone);
47         }
48     } while (flag>0);
49     return 0;
50 }
51
52 struct schedaPersona *creoPersona();
53 int inseriscoPersona(struct schedaPersona **schedario, int n){
54     schedario[n]=creoPersona();
55     n=n+1;
56     return n;
57 }
58
59 struct schedaPersona *creoPersona(){
60     struct schedaPersona *s=(struct schedaPersona *)malloc(sizeof(struct schedaPersona));
61     printf("Dammi il cognome:");
62     scanf("%s",s->cognome);
63     printf("Dammi il nome:");
64     scanf("%s",s->nome);
65     printf("Dammi l'eta':");
66     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,
87             schedario[i]->nome,schedario[i]->eta,schedario[i]->luogoDiNascita,schedario[i]->
88             reddito);
89     }
```