Esercizi Aggiuntivi su Funzioni

Credits Prof Campi

Esercizio

- Definire un nuovo tipo di dato chiamato VT vettore di 10 interi. Scrivere in C la funzione ft che:
  - ha in ingresso un vettore A di tipo VT già caricato e restituisce, attraverso un opportuno parametro, un vettore di tipo VT che contiene gli stessi elementi del vettore di ingresso ma in ordine inverso;
  - restituisce il prodotto di tutti gli elementi di A.

```
#include <stdlib.h>
#include <stdio.h>
#define N 10
typedef int VT[N];

int ft (VT A, VT B) {
    int i, prod=1;
    for(i=0; i<N; i++) {
      B[N-i-1] = A[i];
      prod = prod * A[i];
     }
    return prod;
}

int main() { //non richiesto dal testo
    VT v ={2,-4,8,9,12,1,6,-9,5,3}, w;
    int j;
    for(j=0;j<10;j++)
            printf("%d\n",v[j]);
    printf("prodotto: %d\n", ft(v,w));
    system("pause");  return 0;
}
```

Esercizio

- Definire un nuovo tipo di dato chiamato VT vettore di 10 int. Scrivere in C la funzione ft che:
  - ha in ingresso un vettore A di tipo VT già caricato e restituisce, attraverso un opportuno parametro, una struct con 2 campi a e b di tipo int che contengono rispettivamente la somma degli elementi di A di posto pari e quella degli elementi di A di posto dispari;
  - restituisce la sommatoria di tutti gli elementi di A.

```c
#include <stdlib.h>
#include <stdio.h>
#define N 10
typedef int VT[N];
typedef struct {int a, b;} SOM;

int ft (VT A, SOM *s) {
   int i, som=0;
   s->a=0; s->b=0;
   for(i=0; i<10;i++) {
      som = som + A[i];
      if (i%2 == 0) { s->a = s->a + A[i]; }
      else { s->b = s->b + A[i]; }
   }
   return som;
}
int main() { //non richiesto dal testo
    VT v ={2,-4,8,9,12,1,6,-9,5,13};    SOM s;   int ris;
    ris=ft(v,&s);
    printf("%d %d %d\n",ris,s.a,s.b);
    system("pause");
    return 0;
}
```

Esercizio

- **Definire un nuovo tipo di dato di nome REC, struttura con due campi:**
  - **il primo campo, di nome b2, deve essse di tipo vettore di 10 componenti di tipo int;**
  - **il secondo campo, di nome b10, deve essere di tipo int.**
- **Scrivere una funzione di nome F con un parametro formale di nome X di tipo REC passato per indirizzo. La funzione deve assegnare al campo b10 di X il risultato della conversione da base 2 a base 10 del numero dato nel campo b2 di X.**

```
typedef struct{int b2[10],b10;} REC;

f(REC *X) {
    int i,peso=1;

    X->b10=0;

    for(i=9;i>=0;i--){
        X->b10=X->b10+X->b2[i]*peso;
        peso=peso*2;
    }
}
```

Esercizio

- Definire una struttura dati che descriva le squadre di un torneo di calcio.
  - Le squadre sono 6, di ogni squadra si devono memorizzare: matricola squadra, nome, città e tutti giocatori come nome e numero di maglia.
  - Quanta memoria occupa questa struttura?
- Scrivere una funzione che riceve in ingresso la struttura definita e stampa l'elenco dei portieri titolari.

```
typedef struct {
        char cognome[10];
        int numero;
} giocatore;

typedef struct {
        int matricola-squadra;
        char nome[15];
        char città[10];
        giocatore formazione[11];
} squadra;

squadra squadre-torneo[6];
```

6x(2+15+10+11*12)= 954 bytes

```
void stampaPortieri(squadra sq[]) {
        int i,j;
        for(i=0;i<6;i++)
           for(j=0;j<11;j++)
                   if(sq[i].formazione[j].numero==1)
                       printf("\n%s", sq[i].formazione[j].cognome);
}
```

Esercizio

- Scrivere una funzione che riceve in input due array di N elementi e copia nel secondo gli elementi di valore pari del primo senza lasciare buchi. La funzione restituisce il numero di elementi copiati.

```
int copiaPari(int A[], int B[]) {
        int i,j=0;
        // i usato per scorrere A
        // j punta sempre al primo elemento libero di B

        for (i=0; i<N; i++)
                if(A[i] % 2 ==0) {
                        B[j]=A[i];
                        j++;
                }

        return j;
}
```

Esercizio (cont.)

- Scrivere una funzione che riceve in input un array di N elementi e restituisce una struct composta da un intero e da un array. La funzione inserisce nell'array contenuto nella struct gli elementi pari dell'array che riceve in input e assegna all'intero contenuto nella struct il numero di elementi copiati.
- typedef struct {int v[N];int cont} arrayConCont

```
arrayConCont copiaPari(int A[]) {
        arrayConCont ac;
        int i;  // i usato per scorrere A
        ac.cont=0; // ac.cont punta sempre al primo
                        // elemento di ac.v

        for (i=0; i<N; i++)
                if(A[i] % 2 ==0) {
                        ac.v[ac.cont]=A[i];
                        ac.cont++;
                }

        return ac;
}
```

Esercizio (cont.)

- Scrivere una funzione che riceve in input un array di N elementi e una struct composta da un intero e da un array. La funzione modifica la struct inserendo nell'array gli elementi pari dell'array che riceve in input e assegna all'intero contenuto nella struct il numero di elementi copiati.
- typedef struct {int v[N];int cont} arrayConCont

```
void copiaPari(int A[], arrayConCont *ac) {
        int i;  // i usato per scorrere A
        ac->cont=0; // ac->cont punta sempre al primo
                        // elemento di ac.v

        for (i=0; i<N; i++)
                if(A[i] % 2 ==0) {
                        ac->v[ac->cont]=A[i];
                        ac->cont++;
                }
}
```

Esercizio

- Definire un tipo di dato MATR, matrice 10x10 di interi ed un tipo di dato EST, struct con un campo di nome X di tipo MATR e un campo L di tipo intero. Scrivere una funzione in linguaggio C con due parametri formali: A di tipo MATR e V di tipo puntatore ad un elemento di tipo EST. La funzione deve modificare l'area di memoria indirizzata da V inserendo nel campo X di tale area di memoria la differenza tra la matrice A e la matrice identità 10x10.
  - Si ricorda che la matrice identità 10x10 è una matrice di 10x10 componenti contenente tutti valori nulli ad eccezione della diagonale principale i cui valori sono uguali ad 1.

```
#define N 100
typedef int MATR[N][N];
typedef struct { MATR x; int L; } EXT;
void f(MATR A, EXT *V) {
    int i,j,elemMatrDiag;
    for(i=0;i<N;i++) {
        for(j=0;j<N;j++) {
                if( i == j )
                        elemMatrDiag = 1;
                else
                        elemMatrDiag = 0;

                V->x[i][j] = A[i][j] - elemMatrDiag;
        }
    }
}
```

Esercizio

- Definire un tipo MATR, matrice 20x20 di interi ed un tipo BIS, struct con un campo di nome X di tipo MATR e un intero n che indica il numero di righe di X realmente utilizzate. Scrivere una funzione con tre parametri formali: A di tipo BIS, Q di tipo intero e R di tipo puntatore ad un numero intero.
- La funzione deve restituire attraverso il parametro R il numero di occorrenze di Q tra i valori contenuti nel campo X della struttura A.

```
#define N 20
typedef int MATR[N][N];
typedef struct { MATR x; int n; } bis;

void f(BIS A; int Q; INT *R) {
   int i,j;

   for(i=0;i<N && i<A.n;i++)//n sono le righe realmente usate
         for(j=0;j<N;j++)
               if(A.X[i][j]==Q)
                     *R++;
}
```

Esercizio

- Definire una struttura denominata PRIMO contenente due campi A e B di tipo rispettivamente double e SECONDO dove SECONDO è una struttura composta da due campi denominati rispettivamente X ed Y di tipo int.
- Scrivere la funzione FZX che ha in ingresso un parametro passato per valore sec di tipo SECONDO ed un parametro passato per indirizzo prim di tipo PRIMO e che inserisce nel campo B di prim il contenuto di sec e nel campo A di prim il valore della divisione decimale dei campi X ed Y di sec, se il campo Y è diverso da zero. La funzione restituisce 1 se il campo Y di sec è diverso da zero, 0 altrimenti.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{int X,Y;} SECONDO;
typedef struct{double A; SECONDO B;} PRIMO;
int FZX(SECONDO sec, PRIMO *prim) {
    prim -> B = sec;
    if(sec.Y != 0) {
        prim->A = sec.X / sec.Y;
        return 1;
    }
    else return 0;
}

int main() {  //non richiesto dal testo
   PRIMO p1, p2;   SECONDO s1={3.5, 8.9}, s2={45,0.0};    int r1, r2;
   r1 = FZX(s1,&p1);    r2 = FZX(s2,&p2);
   printf("%d %f\n",r1,p1.A);    printf("%d %f\n",r2,p2.A);
   system("pause"); return 0;
}
```

Esercizio

- Scrivere una funzione che riceve in input una matrice di interi di dimensione N*N e un punto così definito

  typedef struct {int x; int y;} punto

  La funzione deve restituire il numero di multipli dell'elemento della matrice di cui sono state fornite le coordinate.

```
int contaMultipli(int M[][N], punto p) {
        int i,j,mult=0;
        if (p.x<0 || p.x>N-1 || p.y<0 || p.y>N  || M[p.x][p.y] ==0) {
                printf("errore");
                return 0;
        }

        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] % M[p.x][p.y] ==0)
                                mult++;

        return mult;
}
```

Esercizio (tdeB 24-11-2005)

- Si codifichi un programma C che legge dallo standard input una sequenza (di lunghezza arbitraria) di interi positivi terminata dal valore 0 e, al termine della sequenza, visualizza su standard output un messaggio che indica *quante* ***coppie*** *di numeri consecutivi che siano diversi, pari* ***e*** *il cui prodotto sia un quadrato perfetto* sono contenute nella sequenza. Ecco un esempio:

  2 50 13 16 8 7 8 2 18 6 6 16 4 1 25 0

  Le coppie di numeri cercati sono 4 (2 50, 8 2, 2 18, 16 4)

- Si noti che un numero può anche appartenere a due coppie. Nella soluzione **si utilizzi** la funzione:

  int dppqp(int a, int b); /* Restituisce 1 se i parametri sono **d**iversi, **p**ari, e il loro **p**rodotto è un **q**uadrato **p**erfetto, 0 altrimenti */

- La funzione dichiarata qui sopra può essere ad esempio definita come segue:

```
int dppqp (int a, int b) {
  int x = 2, p = a*b;
  if ( a%2 || b%2 || a==b ) // Se uno dei parametri è dispari o sono uguali: -> 0
    return 0;
  while ( x*x < p )      // Prova i quadrati di tutti i numeri pari iniziando
    x += 2;            // da 2 e fino a raggiungere o superare a*b
  return x*x == p;     // Se l'ultimo quadrato supera p: -> 0, altrimenti: -> 1
}
```

```
#define SENT 0

int main() {
int cont = 0,       // L'inizializzazione di cur a 1 è innocua
    prev, cur = 1;  // rappresentano il valore corrente e il precedente

  printf("\nSequenza di int separati da spazi (termina con %d)\n\n", SENT);
  do {
    prev = cur;
    scanf("%d", &cur);
    if(cur!=0 && dppqp(prev,cur))
        cont++;
  } while ( cur != SENTINELLA );

  printf("Le coppie di numeri cercati sono %d\n", cont);

  return 0;
}
```

Esercizio (tde 16-11-2007)

Le matrici quadrate N x N possono essere navigate in diversi modi. Due modi tradizionali sono il nested loop e il merge scan. Il nested loop è la navigazione riga per riga (o colonna per colonna), come esemplificato sotto (nella matrice i numeri rappresentano l'ordine di navigazione).

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

a) Si scriva una funzione che riceve in input due matrici quadrate A e B e un intero k. A e B sono matrici NxN e k è compreso tra 1 e NxN (non serve verificare, k è garantito essere compreso tra 1 e NxN). La funzione deve attraversare, secondo l'ordine del nested loop per righe, le prime k posizioni sia di A che di B e restituire il numero di interi uguali che si trovano nella stessa posizione in A e in B. (2 punti).
Esempio: con k=12 e le matrici

6	***8***	13	***87***	78	5	***9***	23
12	***45***	32	12				

5	***8***	12	***87***	56	6	***9***	11
13	***45***	11	32				

restituisce 4, perché sono uguali l'8 in posizione (1,2), l'87 in (1,4), il 9 in (1,7) e il 45 in (2,2).

```
int quantiUgualiConNestedLoop(int A[][N], int B[][N], int k)
```

Il metodo merge scan invece è quello che si muove lungo le diagonali partendo dal punto più in alto a sinistra, come esemplificato sotto (nella matrice i numeri rappresentano l'ordine di esplorazione).

1	2	4	7	11	16	22	29
3	5	8	12	17	23	30	37
6	9	13	18	24	31	38	44
10	14	19	25	32	39	45	50
15	20	26	33	40	46	51	55
21	27	34	41	47	52	56	59
28	35	42	48	53	57	60	62
36	43	49	54	58	61	63	64

b) Si scriva una funzione che riceve in input due matrici quadrate A e B e un intero k. A e B sono matrici NxN e k è compreso tra 1 e NxN (non serve verificare, k è garantito essere compreso tra 1 e NxN). La funzione deve attraversare, secondo l'ordine del merge scan, le prime k posizioni sia di A che di B e restituire il numero di interi uguali che si trovano nella stessa posizione in A e in B. (2 punti).
Esempio, con k=12 e le matrici

23	45	12	***3***	11
2	***32***	1	65	
1	9			
10				

3	43	87	***3***	9
1	***32***	3	100	
2	54			
10				

restituisce 3, perché sono uguali il 3 in posizione (1,4), il 32 in (2,2) e il 10 in (4,1).

```
int quantiUgualiConMergeScan(int A[][N], int B[][N], int k)
```

```
int quantiUgualiConNestedLoop(int A[][N], int B[][N], int k) {
  int cont =0, howMany=0, i, j;

  for (i=0;i<N && howMany<k;i++) {
    for (j=0;j<N && howMany<k;j++) {
      if (A[i][j] == B[i][j])
                    cont++;
      howMany++;
    }
  }

  return cont;
}
```

```
int quantiUgualiConMergeLoop(int A[][N], int B[][N], int k) {
  int cont=0, howMany=0, i, numElem;
  int dn=1;//numero diagonale
  while (dn<=2*N-1 && howMany<k) {
    i=0;
    if (dn <= N) {    //prima meta'
      numElem = dn;  //indica quanti elementi in diagonale
      while (i<numElem && howMany<k) {
          if (A[i][dn-1-i] == B[i][dn-1-i])
                      cont++;
                    howMany++;
          i++;
       }
    } else {  //seconda meta’
```

```
  } else {  //seconda meta‘
     numElem = N - (dn-N);
     while (i<numElem && howMany<k) {
         if(A[dn-N+i][N-1-i]==B[dn-N+i][N-1-i])
                cont++;
                howMany++;
                i++;
      }
    }
    dn++;
  }
  return cont;
}
```

oppure…

```
//soluzione alternativa merge scan
int ms(int A[][N],int B[][N],int k){
        int d,p,i,j,cont=0,quanti=0;

        //prima metà
        for(d=0;d<N && cont<k;d++)
          i=0;j=d;
          for(p=0;p<d+1 && cont<k;p++){
            if(A[i][j]==B[i][j])
              quanti++;
            cont++;

            i++;j--;
          }
```

```
    //seconda metà
    for(d=1;d<N && cont<k;d++)
      i=d;j=N-1;
      for(p=N-1;p>=1 && cont<k;p--){

        if(A[i][j]==B[i][j])
          quanti++;
        cont++;

        i++;j--;
      }

    return quanti;
}
```

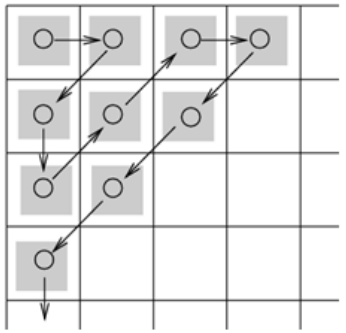
```
int quantiUgualiConMergeLoop(int A[][N], int B[][N], int k) {
        int i,j,cont=0,quanti=0;
        i=0;j=0;
        while(quanti<k){//prima metà
                //guarda casella giusta
                if(A[i][j]==B[i][j])
                        cont++;
                quanti++;
                //trova casella giusta
                i++;j--;
                if(j==-1) {
                        if(i==N-1)
                                break;
                        j=i;  i=0;
                }
        }
```

```
        while(quanti<k){//seconda metà
                //guarda casella giusta
                if(A[i][j]==B[i][j])
                        cont++;
                quanti++;
                //trova casella giusta
                i++; j--;
                if(i==N-1) {
                        i=j+2; j=N-1;
                }
        }

        return cont;
}
```

```
int merge(int A[][N],int B[][N],int k){
   int cont=0,quanti=0,i,j,diag=0;
   while(diag<N && quanti<=k) {
       i=0;j=diag;
       while(j>=0 && quanti<=k) {
            if(A[i][j]==B[i][j])
                cont++;
           quanti++;
           i++;j--;
       }
       diag++;
   }
```

```
   diag=1;
   while(diag<N-1 && quanti<=k) {
       i=diag;j=N-1;
       while(i<N && quanti<=k) {
            if(A[i][j]==B[i][j])
                cont++;
           quanti++;
           i++;j--;
       }
       diag++;
   }
   return cont;
}
```

Esercizio (tdeB 22/11/2007)



Tra i vari modi di scandire le celle di una matrice N·N, l'ordine *diagonale serpeggiante* è illustrato in figura (inizia dalla cella m[0][0]). Si codifichi la funzione

```
int snakesum(int m[][N], int n)
```

che (1) riceve come parametri una matrice quadrata `m` (N è una costante positiva già definita) e un intero `n`; (2) se `n` è *valido*, considera i primi `n` elementi di `m` in ordine diagonale serpeggiante, altrimenti restituisce -1; (3) restituisce al chiamante la somma degli `n` numeri incontrati.

1	4	1	1	3
2	1	7	5	2
5	1	4	0	1
2	3	2	9	3
1	6	8	5	8

Attenzione: `n` è valido se compreso tra 1 e N·(N+1)/2, cioè consideriamo solo la "prima metà" della matrice. Con riferimento alla matrice qui a fianco:

snakesum(m, 2) → 5
snakesum(m, 1) → 1
snakesum(m, -4) → -1
snakesum(m, 4) → 12
snakesum(m, 16) → -1
snakesum(m, 8) → 22

Per scandire la matrice occorre percorrere in n passi le varie diagonali successive in direzione alternata, lunghe ogni volta una casella in più. Effettuiamo con un ciclo in cui diminuiamo n esattamente gli n passi, accumulando la somma dei valori incontrati. Chiamando *fase* ogni successiva diagonale, introduciamo anche un contatore relativo a ogni singola fase (che chiamiamo *traccia*); la traccia ricomincerà sempre da 1 ogni volta che si invertirà la direzione di marcia e crescerà fino all'indicatore di fase (che indica la lunghezza della diagonale – la terza diagonale è lunga 3, la quarta è lunga 4, …). Lo spostamento diagonale avviene sommando agli indici i e j due contributi variabili (+1 e -1): la direzione si scambia invertendo il segno di tali fattori additivi, in occasione dell'arrivo su un "bordo" (identificato dalla condizione "fase = traccia").

```
int snakesum( int m[][N], int n ) {
  int sum = 0, fase = 1, traccia = 1, i = 0, j = 0;
  int deltai = -1, deltaj = 1;
  if ( n < 1 || n > N*(N+1)/2 )
    return -1;
  while ( n > 0 ) {
    sum += m[i][j];
    if( traccia < fase ) {        // mi muovo in diagonale, sono a metà della fase
      traccia++;                  // aumento la traccia
      i += deltai;                // spostamento lungo i
      j += deltaj;                // spostamento lungo j
    } else {                      // sul bordo! -> cambio fase, azzero la traccia
      fase++;                     // cambiamo la fase
      traccia = 1;                // resettiamo la traccia
      deltai = -deltai;           // invertiamo la direzione diagonale
      deltaj = -deltaj;
      if ( i == 0 )
          j++;          // se siamo su un "bordo i==0"
      else
          i++;          // se siamo su un "bordo j==0"
    }
    n--;
  }
  return sum;
}
```

Esercizio (tde 11-11-2011)

- I risultati della finale olimpica di nuoto della staffetta 4x100 stile libero sono rappresentati mediante una matrice di dimensioni 4x8.
- Ogni cella rappresenta il risultato di un singolo frazionista tramite la seguente struttura:
  typedef struct { int min,sec,cent;} tempo;
  typedef struct {char nome[15],cognome[15],nazionalità[15];
  tempo t; } frazione;
- La matrice è dichiarata in questo modo:
  typedef frazione risultato[4][8];
- I risultati sono disposti nella matrice secondo la corsia (e.g. corsia 1 nella prima colonna)
- Scrivere una funzione che riceve in ingresso la matrice che contiene il risultato e restituisce il numero della corsia vincente.
  int corsiaVincente(risultato ris);
- Scrivere una funzione che riceve in ingresso la matrice che contiene il risultato e una variabile che contiene il record mondiale di staffetta e che restituisce 1 se il record mondiale è stato battuto, 0 altrimenti.
  int recordBattuto(risultato ris, tempo record);

```
long converti(tempo t) {
        return t.min*6000+t.sec*100+cent;
}
long calcolaCorsia(risultato ris, int k) {
        int i,tot=0;
        for(i=0;i<4;i++) {
                tot+=converti(ris[i][k]);
        }
        return tot;
}
int corsiaVincente(risultato ris) {
        int i,min,corrente,imin=0;
        min==calcolaCorsia(ris,0);
        for(i=0;i<8;i++) {
                corrente=calcolaCorsia(ris,i);
                if(corrente<min){
                        min=corrente;imin=i;
                }
        }
        return imin+1;
}
```

```
int recordBattuto(risultato ris,tempo record) {
        int v;
        v=corsiaVincente(ris);
        if(calcolaCorsia(ris,v)<converti(record))
                return 1;
        else
                return 0;
}
```

Esercizio

- Scrivere una funzione che riceve in input una matrice di interi di dimensione N*N e restituisce un'opportuna struttura dati contenente tutti i punti della matrice contenenti un valore uguale al massimo valore contenuto nella matrice stessa
- Si utilizzi per memorizzare un punto la struttura dati

  typedef struct {int x; int y;} punto

Vedremo due soluzioni

1. Mettiamo tutti i massimi in array di punti
2. Mettiamo tutti i massimi in una struct che contiene un array di punti e un contatore che dice quanti elementi dell'array contengono effettivamente valori validi
   typedef struct{int cont; punto punti[]} massimiT;

```
int trovaMassimi(int M[][N], punto massimi[]) {
        int i,j,cont=0,max;
        max=M[0][0];
        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] >max)
                                max=M[i][j];
        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] == max) {
                                massimi[cont].x=i;
                                massimi[cont].y=j;
                                cont++;
                        }
        return cont;
}
```

```
typedef struct{int cont; punto punti[]} massimiT;
void trovaMassimi(int M[][N], massimiT *massimi) {
        int i,j,max;
        massimi -> cont=0//equivale a (*massimi).cont=0
        max=M[0][0];
        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] >max)
                                max=M[i][j];
        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] == max) {
                                massimi->punti[massimi -> cont].x=i;
                                massimi->punti[massimi -> cont].y=j;
                                massimi-> cont++;
                        }
}
```

```
typedef struct{int cont; punto punti[]} massimiT;
massimiT trovaMassimi(int M[][N]) {
        massimiT massimi;
        int i,j,max;
        massimi.cont=0
        max=M[0][0];
        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] >max)
                                max=M[i][j];
        for (i=0; i<N; i++)
                for (j=0; j<N; j++)
                        if(M[i][j] == max) {
                                massimi.punti[massimi.cont].x=i;
                                massimi.punti[massimi.cont].y=j;
                                (massimi.cont)++;
                        }
        return massimi;
}
```

Esercizio

- **Simulare l'esecuzione di ciascuna delle 4 chiamate a sottoprogrammi, limitandosi alla descrizione dei valori stampati tramite le *printf*.**

```
#include <stdio.h>

int varA = 12, varB = 25;

int proc1(int *par) {
        *par = *par - 7;
        return(23);
}

void proc2(int *varX, int varY) {
        int varB;

        *varX = varY;
        varB = *varX + varY;
}

int proc3(int varA) {
        int varZ = 17;

   varA = varA + varZ;
        return(varA);
}
```

```
int varA = 12, varB = 25;

int main() {
  int varC = 63;
  printf("\n0) varA=%d,varB=%d,varC=%d",varA,varB,varC);
  varC = proc1(&varA);   /* passo 1 */
  printf("\n1) varA=%d,varB=%d,varC=%d ",varA,varB,varC);
  proc2(&varA, varC);    /* passo 2 */
  printf("\n2) varA=%d,varB=%d,varC=%d ",varA,varB,varC);
  varB = proc3(varC);    /* passo 3 */
  printf("\n3) varA=%d,varB=%d,varC=%d ",varA,varB,varC);
  varB = proc1(&varB);   /* passo 4 */
  printf("\n4) varA=%d,varB=%d,varC=%d ",varA,varB,varC);
  return 0;
}
```

Variabili visibili dal main:	varA	varB	varC
situazione iniziale	12	25	63
dopo *varC = proc1(&varA);*	*5*	*25*	*23*
dopo *proc2(&varA, varC);*	*23*	*25*	*23*
dopo *varB = proc3(varC);*	*23*	*40*	*23*
dopo *varB = proc1(&varB);*	*23*	*23*	*23*

Esercizio

- **Scrivere un programma in linguaggio C che legga dallo standard input due sequenze di interi positivi ognuna terminata dal numero 0 (escluso).**
- **Relativamente alla sequenza più lunga (SE FOSSERO UGUALI) si deve stampare a terminale: l'elemento massimo con il suo numero d'ordine e l'elemento minimo con il suo numero d'ordine e infine l'intera sequenza in ordine inverso.**

```
#include <stdio.h>

#define MAX_LEN 100

void analizza(int l[],int len,int *max,int *imax,
                          int *min,int *imin,int c[],int *lenC)
void leggi_seq(int v[], int *len);
void scrivi_seq(int v[], int len);
void trova_max(int v[], int len, int *max, int *index);
void trova_min(int v[], int len, int *min, int *index);
void inverti_seq(int len, int v_in[], int v_out[]);
```

```
int main() {
   int a[MAX_LEN],b[MAX_LEN],c[MAX_LEN];
   int len_a, len_b, len_c, max,min, ind_max, ind_min;
   leggi_seq(a, &len_a);
   leggi_seq(b, &len_b);
   if (len_a > len_b) {
      analizza(a,len_a,&max,&ind_max,&min,&ind_min,c,&len_c);
    } else {
      analizza(b,len_b,&max,&ind_max,&min,&ind_min,c,&len_c);
    }
    printf("\n la sequenza piu' lunga ha: \n");
    printf("\n max = %d in pos %d",max,ind_max);
    printf("\n min  = %d in pos %d",min,ind_min);
    printf("\n la sequenza invertita e': \n");
    scrivi_seq(c,len_c);
    return 0;
}
```

```
void analizza(int l[],int len,int *max,int *imax,int *min,int *imin,
                                                    int c[], int *lenC) {
      trova_max(l,len,max,imax);
      trova_min(l,len,min,imin);
      inverti_seq(len,l,c);
      lenC = len;
}

void leggi_seq(int v[], int *len) {
        int i = 0;
        printf("\nInserire una seq. di interi terminata con 0.");
        do {
                printf("\n inserisci elemento: ");
                scanf("%d",&v[i]);
                i++;
        } while ((v[i-1]!=0)&&(i < MAX_LEN));
        if (i == MAX_LEN) *len = MAX_LEN;
        else *len = i-1;
}
```

```
void scrivi_seq(int *v, int len) {
        int i;
        printf("\nla sequenza e': \n");
        for (i = 0; i < len; i++)
                printf("%5d",v[i]);
}

void inverti_seq(int len, int v_in[], int v_out[]) {
        int i;
        for (i = len-1; i>=0; i--) {
                v_out[len-1-i] = v_in[i];
         }
}
```

```
void trova_max(int v[], int len, int *max, int *index) {
        int i,aux,ind_aux=0;
        aux = v[0];

        for (i = 1; i < len; i++) {
                if (v[i] > aux) {
                  aux = v[i];
                  ind_aux = i;
                }
        }

        *max = aux;
        *index = ind_aux;
}
```

```
void trova_min(int v[], int len, int *min, int *index) {
        int i,aux,ind_aux=0;
        aux = v[0];

        for (i = 1; i < len; i++) {
            if (v[i] < aux) {
                    aux = v[i];
                    ind_aux = i;
                }
        }

        *min = aux;
        *index = ind_aux;
}
```

Esercizio

- Scrivere un programma in linguaggio C che definendo opportune funzioni, risolva il seguente problema. Date due stringhe S1 e S2, stabilire se S2 compare come sottosequenza di S1 e, in caso affermativo, a partire da quale indice. Creare quindi una nuova stringa S3 costruita come S1 meno la prima occorrenza della stringa S2. Non è permesso utilizzare le funzioni di libreria del linguaggio relative alla manipolazione di stringhe.
- Esempio: S1 = “melograno”, S2 = “grano”;

  restituisce 5 (S2 è inclusa in S1 a partire dal 5° carattere) e

  S3 = “melo”.
- Se fosse S1 = “sperpero”, S2 = “per”;

  il risultato è 2 e S3 = “spero”.

Le funzionalità che il programma deve possedere saranno realizzate definendo tre funzioni e precisamente:

```
int lung_str(char *s);

int inclusione_str(char *s1, char *s2);
// restituisce -1 se s2 non è contenuta in s1; altrimenti
// l'indice della cella in cui inizia l'occorrenza di s2 in s1;

void sottrai_str(char *s1, char *s2, char *s3);
// s3 conterrà tutti i caratteri di s1 tranne la prima
// occorrenza della stringa s2.
```

```
#include <stdio.h>
#define MAX_LEN 100
int lung_str(char *s);
int inclusione_str(char *s1, char *s2);
void sottrai_str(char *s1, char *s2, char *s3);

int main() {
  char stringa1[MAX_LEN] , stringa2[MAX_LEN], stringa3[MAX_LEN];
  int res;
  printf("\n inserisci la prima stringa: ");   scanf("%s",stringa1);
  printf("\n inserisci la seconda stringa: ");   scanf("%s",stringa2);
  res = inclusione_str(stringa1,stringa2);
  if (res == -1)
        printf("\n%s non e' contenuta in %s !",stringa2,stringa1);
  else {printf("\n%s in %s inizia in pos %d",stringa2,stringa1,1+res);
        sottrai_str(stringa1,stringa2,stringa3);
        printf("\nla stringa differenza e': s3 = %s",stringa3); }
  return 0;
}
```

```
int lung_str(char *s) {
        int i = 0;

        while (s[i]  != '\0') {
                i++;
        }

        return i;
}
```

```
int inclusione_str(char *s1, char *s2) {
  int len1,len2,trovato,i,j,new_i;
  len1 = lung_str(s1);    len2 = lung_str(s2);
  if (len1 < len2) return -1;
  for (i = 0; i <= len1-len2; i++) {
        j = 0;
        trovato = 1;
        new_i = i;
        while ((j < len2)&&(trovato)) {
              trovato = (s1[new_i] == s2[j]);
              j++;
              new_i++;
        }
        if ((j == len2)&&(trovato))
              return i;
  }
  return -1;
}
```

```
void sottrai_str(char *s1, char *s2, char *s3) {
        int index,len1,len2,i;

        index = inclusione_str(s1,s2);
        len1 = lung_str(s1);
        len2 = lung_str(s2);

        for (i = 0; i < index; i++)
                s3[i] = s1[i];

        for (i = index + len2; i < len1; i++)
                s3[i-len2] = s1[i];

        s3[i-len2] = s1[i];  // per aggiungere il char '\0'
}
```

Esercizio

- Si considerino due matrici di interi A e B, di uguali dimensioni (R e C, costanti, che indicano il numero di righe e colonne). Diciamo che A *domina* B se, confrontando i valori in posizioni corrispondenti, risulta che il numero dei valori in A maggiori dei corrispondenti valori in B è più grande del numero di quelli di B maggiori dei corrispondenti in A e inoltre gli elementi corrispondenti non sono mai uguali (se due elementi corrispondenti sono uguali la dominanza non è definita).
- Si codifiche la funzione **domina(...)** che riceve le matrici come parametri e restituisce 1 se la prima domina la seconda, -1 se la seconda domina la prima, 0 altrimenti.

```
int domina ( int A[][C], int B[][C] ) {
        int bilancio = 0, i, j;
        for ( i=0; i<R; i++ ) {
                for ( j=0; j<C; j++ ) {
                        if ( A[i][j] == B[i][j] )
                                return 0;
                        if ( A[i][j] >  B[i][j] )
                                bilancio++;
                        else
                                bilancio--;
                }
        }
        if ( bilancio > 0 ) return  1;
        if ( bilancio < 0 ) return -1;
        return  0;
}
```

Esercizio (tdeB 18-7-2007)

- *SimpleScrabble* è una versione del gioco *Scrabble* dove il valore delle parole dipende solo dalle lettere che le compongono. Ogni lettera ha un valore (da 1 a 10) ed è disponibile in un certo numero di esemplari (da 1 a 12). Il valore di una parola è la somma dei valori delle sue lettere. Gli array **valore** e **numero**, definiti nell'ambiente globale come segue, indicano il valore e il numero di esemplari disponibili di ognuna delle 26 lettere, in ordine alfabetico. Consideriamo per semplicità solo le lettere maiuscole.
- **int** numero[26] = {9,2,2,4,12,2,3,2,9,1,1,4,2,6,8,2,1,6,4,6,4,2,2,1,2,1};
- **int** valore[26] = {1,2,3,2,1,4,2,4,1,8,5,1,3,1,1,3,10,1,1,1,1,4,4,8,4,10};
- Ci sono quindi 9 A da 1 punto, 2 B da 2 pt, 2 C da 3 pt, 4 D da 2 pt, 12 E da 1 pt, …, 1 Z da 10 pt.
- Una parola è valida se è di almeno 2 lettere ed è ottenibile con le lettere a disposizione (CAB vale 3+1+2=6 pt, KARAOKE non è valida, perché disponiamo di una sola K).
- Una funzione **valida(**...**)**, che restituisce 1 se la parola è valida, 0 altrimenti
- Una funzione **punteggio(…)** calcola e restituisce il valore della parola ricevuta come parametro, assegnando 0 alle parole *non valide*.
- Si codifichino in C le funzioni **valida(…)** e **punteggio(**...**)**

```
int valida ( char parola[] ) {
    int i, p, cont, lun = strlen(parola);
    if ( lun < 2 )  return 0;
    for ( i=0 ; i<26 ; i++ ) {
        for(cont=0,p=0; p<lun; p++) if ( parola[p] == 'A'+i ) cont++;
        if ( cont > numero[i] ) return 0;
    }
    return 1;
}

int punteggio ( char parola[] ) {
    int punt = 0, lun;
    if ( valida(parola) )
         for ( lun = strlen(parola)-1 ; lun >= 0 ; lun-- )
              punt += valore[parola[lun]-'A'];
    return punt;
}
```

Oppure

```
int valida(char parola[]){
        int i=0,j,lun,cont;
        lun=strlen(parola);
        if(lun<2)
                return 0;
        for(i=0;i<lun;i++){//per ogni lettera della parola
                //conto e confronto
                cont=1;
                for(j=i+1;j<lun;j++){
                        if(parola[j]==parola[i])
                                cont++;
                }
                //cont contiene quanti car ci sono
                if(cont>numero[parola[i]-'A'])
                        return 0;
        }
        return 1;
}
```

```
int punteggio(char parola[]){
        int lun,i,punteggio=0;

        if(!valida(parola))
                return 0;

        lun=strlen(parola);

        for(i=0;i<lun;i++)
                punteggio+=valore[parola[i]-'A'];

        return punteggio;
}
```

Esercizio (tdeB 22-11-2007)

- Definiamo *totalmente diverse (TD)* due parole che non hanno lettere in comune. Ad esempio "fuoco" e "aria" sono TD, mentre "fuoco" e "acqua" no (entrambe hanno 'c', anche se in posizioni diverse).
- Si definisca una funzione **… td(…)** che riceve come parametri due stringhe (che supponiamo valide e ben formate) e restituisce **1** se le stringhe rappresentano due parole TD, **0** altrimenti
- Si scriva un programma C che legge da stdin una sequenza (di lunghezza arbitraria, anche enorme) di parole, ognuna di massimo 50 caratteri, e al termine della sequenza indichi su stdout il numero di parole della sequenza e il numero di coppie di parole TD **consecutive**. La sequenza è terminata dalla parola "fine" (che non deve essere considerata parte della sequenza)

```
int td( char s1[], char s2[] ) {
  int i = 0, j;
  while( s1[i] != '\0' ) {
    j = 0;
    while( s2[j] != '\0' ) {
      if( s1[i] == s2[j] )
        return 0;
      j++;
    }
    i++;
 }
 return 1;
}
```

Oppure

```
int td( char s1[], char s2[] ) {
  int i, j;
  for( i=0; s1[i]!='\0'; i++ )
    for( j=0; s1[i]!='\0'; j++ )
      if( s1[i] == s2[j] )
        return 0;
  return 1;
}
```

Oppure

```
int td( char s1[], char s2[] ) {
  int i, j;
  for( i=strlen(s1)-1; i>=0; i-- )
    for( j=strlen(s2)-1; j>=0; j-- )
      if( s1[i] == s2[j] )
        return 0;
  return 1;
}
```

```
int main() {
  int contP = 0, contTD = 0;
  char p[51], pprec[51], fine[] = "fine";    // 51 perché c'è anche '\0'

  scanf("%s", p);
  strcpy(pprec, p);       // inizializzando pprec come p non falsiamo contTD

  while( strcmp(p, fine) ) {
    contP++;
    contTD = contTD + td(pprec, p);
    strcpy(pprec, p);
    scanf("%s", p);
  }

  printf("parole: %d   coppie TD consecutive: %d\n", contP, contTD);

  return 0;
}
```

Esercizio (tdeB 21/9/2007)

- Una matrice quadrata di NxN punti del piano cartesiano è definita come segue:

  typedef struct { float x, y; } Punto;

  typedef Punto Matrice[N][N];
- Gli N punti della diagonale, quelli di ogni riga e quelli di ogni colonna definiscono linee spezzate di N-1 lati. Diciamo che una matrice di punti è *regolare* se la lunghezza della spezzata definita dalla *diagonale principale* è maggiore della lunghezza di *tutte* le spezzate definite dalle righe e dalle colonne della matrice.
- Sapendo di poter disporre di una funzione di prototipo **float dist(Punto a, Punto b)** che calcola la distanza euclidea tra due punti, si codifichi in C la funzione **regolare(**...**)** che, data una matrice, restituisce 1 se è regolare, 0 altrimenti;

  int regolare(Matrice m);

  int regolare(Punto m[][N]);

```
int regolare( Matrice m ) {
  int i, j;  float lunD = 0.0, lun;
  for ( i = 0 ; i < N-1 ; i++ )
    lunD += dist( m[i][i], m[i+1][i+1] );      /* La diagonale principale */
  for ( i = 0 ; i < N ; i++ ) {
    lun = 0.0;              /* Azzero per la nuova riga(colonna) */
    for ( j = 0 ; j < N-1; j++ ) {
      lun += dist( m[i][j], m[i][j+1] );     /* i-esimo segmento riga    */
    }
    if ( lun > lunD ) /* Basta una sola lunghezza maggiore */
      return 0;
  }
  for ( i = 0 ; i < N ; i++ ) {
    lun = 0.0;              /* Azzero per la nuova riga(colonna) */
    for ( j = 0 ; j < N-1; j++ ) {
      lun += dist( m[j][i], m[j+1][i] );     /* i-esimo segmento colonna */
    }
    if ( lun > lunD ) /* Basta una sola lunghezza maggiore */
      return 0;
  }
  return 1;
}
```

Si proponga una modifica alla funzione codificata per fare il confronto con la spezzata definita dalla **diagonale secondaria** (invece che con quella definita dalla diagonale principale).
Si cerchi di individuare la **minima** modifica necessaria. La diagonale principale va dall'elemento ⟨0,0⟩ all'elemento ⟨N-1,N-1⟩, la diagonale secondaria va dall'elemento ⟨0,N-1⟩ all'elemento ⟨N-1,0⟩. [***1,5** punti*]

È sufficiente calcolare diversamente lunD, nel primo ciclo for, lasciando inalterato tutto il resto:

lunD += dist(m[i][**N-1-i**],m[i+1][**N-2-i**]); /*La diagonale secondaria*/

Esercizio

- Scrivere in linguaggio C, un programma che letta una sequenza di N numeri complessi dallo standard input (rappresentati con parte reale e parte immaginaria) stampi video la sequenza ordinata in maniera crescente secondo il valore dei loro moduli.

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

typedef struct { float re;  float im; } complex;

void scambio(complex *p1, complex *p2);
float modulo2(complex x);
int cmp_complex(complex c1, complex c2);
void ordina_vettore(int N, complex v_in[], complex v_out);

int main() {
    complex A[MAX_LEN], B[MAX_LEN];
    int N;
     //……..
    return 0;
}
```

```
void scambio(complex *p1, complex *p2) {
    complex aux;
    aux = *p1;
    *p1 = *p2;
    *p2 = aux;
}

float modulo2(complex x) {
  return sqrt((x.re * x.re)+(x.im * x.im));
}

int cmp_complex(complex c1, complex c2) {
  if (modulo2(c1) > modulo2(c2)) return 1 ;
  if (modulo2(c1) < modulo2(c2)) return -1 ;
  return 0;
}
```

```
void ordina_vettore(int N, complex v_in[], complex v_out) {
   int i, j;

   for (i = 0; i < N - 1; i++) { v_out[i] = v_in[i]; }
   // Questa copia tra vettori si rende necessaria, visto che non
   // si desidera che dopo la chiamata a questo sottoprogramma il
   // vettore V_in[] nel chiamante risulti modificato.

   for (i = 0; i < N - 1; i++) {
     for (j = 0; j < N - i - 1; j++) {
       if (cmp_complex(v_in[j],v_in[j + 1]) > 0) {
           scambio(&v_in[j], &v_in[j + 1]);
       } // end if
     } // end for
   } // end for

} // end ordina_vettore
```

Esercizio

- Sia dato un vettore di N ≥ 2 elementi. Ogni elemento del vettore è un numero intero. Si chiede di progettare e codificare in C una funzione che verifichi se, a partire da un elemento di indice i ≥ 0 fino alla fine del vettore, ogni elemento abbia valore inferiore alla somma di tutti gli elementi collocati alla sua destra, cioè alla somma di tutti gli elementi aventi indice superiore al suo (beninteso questa verifica non s'applica all'ultimo elemento, che non ha niente alla sua destra).
- Per esempio, il vettore seguente (con N = 5) soddisfa alla condizione in questione, a partire dall'elemento di indice 1 (ma non a partire da quello di indice 0).
  indice 0 1 2 3 4
  elemento 9 3 0 1 4
- La funzione ha come argomento il vettore e l'indice i da cui partire con la verifica. Essa restituisce 1 se il vettore soddisfa alla condizione così descritta, 0 altrimenti.
- Si supponga di disporre già di una funzione,
  int somma(int vett [], int k)
  che restituisce la somma di tutti gli elementi del vettore vett, a partire da quello di indice k (compreso) fino alla fine del vettore (compresa).

```
#include <stdio.h>
#define N 1000

int verifica_inf(int vett [], int i);
int somma(int vett [], int i);

int main( ) {/*……*/ return 0; }

int verifica_inf (int vett [], int i) {
        int j, flag=1;
        j = i;
        while ((j <= N - 2) && (flag == 1)) {
                if (vett [j] < somma(vett, j + 1)) {  j++; } else { flag = 0; }
        }
        return flag;
}

int somma(int vett [], int i) {
        int k, sum = 0;
        for (k=i; k < N; k++)
                sum = sum + vett[k];
}
```

Esercizio

Supponendo di disporre della seguente struttura dati per rappresentare un polinomio a coefficienti reali,

```
#define MAX_LEN 100
typedef struct { float coeff[MAX_LEN]; int deg; } pol;
  // coeff[0]: coefficiente meno significativo.
```

Scrivere un programma in linguaggio C che leggendo due polinomi dallo standard input, calcoli la somma e il prodotto dei due polinomi facendo uso di sottoprogrammi aventi i seguenti prototipi:

```
void leggi_pol(pol *p);
void scrivi_pol(pol *p);
pol somma_pol(pol *p1, pol *p2);
pol prod_pol(pol *p1, pol *p2);
```

```
#include <stdio.h>
#define MAX_LEN 100
typedef struct { float coeff[MAX_LEN]; int deg; } pol;

void leggi_pol(pol *p);
void scrivi_pol(pol *p);
pol somma_pol(pol *p1, pol *p2);
pol prod_pol(pol *p1, pol *p2);

int main() {
        pol a,b,c,d;
        leggi_pol(&a);
        leggi_pol(&b);
        scrivi_pol(&a);

        c = somma_pol(&a,&b);
        printf("\nla somma dei due polinomi inseriti vale: \n");
        scrivi_pol(&c);

        d = prod_pol(&a,&b);
        printf("\nil prodotto vale: \n");
        scrivi_pol(&d);
        return 0;
}
```

```
void leggi_pol(pol *p) {
        int i;
        do {
                printf("\nInserisci il grado del polinomio: ");
                scanf("%d",&p->deg);
        } while ((p->deg < 0)||(p->deg > MAX_LEN / 2 - 1));

        for (i = 0; i <= p->deg; i++) {
                printf("\ncoeff[%i]= ",i);
                scanf("%f",&p->coeff[i]);
        }
}

void scrivi_pol(pol *p) {
        int i;
        printf("\n%3.1f X^%d",p->coeff[p->deg],p->deg);
        for (i = p->deg - 1; i>=0; i--) {
                printf(" + %3.1f X^%d ",p->coeff[i],i);
        }
}
```

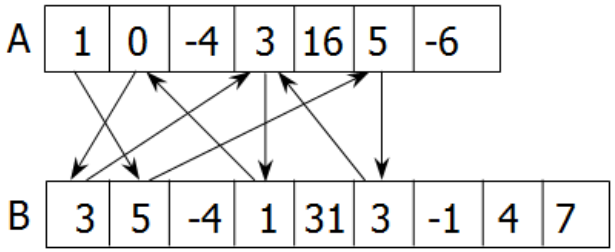
```
pol somma_pol(pol *p1, pol *p2) {
        pol res;
        int i;
        if (p1->deg > p2->deg) {
                res.deg = p1->deg;

                for (i = p2->deg +1; i<= p1->deg; i++)
                        p2->coeff[i] = 0.0;

                for (i = 0; i<= p1->deg; i++)
                        res.coeff[i] = p1->coeff[i] + p2->coeff[i];
        } else {
                res.deg = p2->deg;

                for (i = p1->deg +1; i<= p2->deg; i++)
                        p1->coeff[i] = 0.0;

                for (i = 0; i<= p2->deg; i++)
                        res.coeff[i] = p1->coeff[i] + p2->coeff[i];
        }
        return res;
}
```

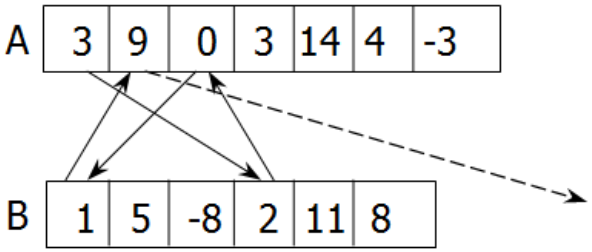
```
pol prod_pol(pol *p1, pol *p2) {
        pol res;  int i,j,k;
        if (p1->deg > p2->deg) {
                for (i = p2->deg +1; i<= p1->deg; i++)
                        p2->coeff[i] = 0.0;
        } else {
                for (i = p1->deg +1; i<= p2->deg; i++)
                        p1->coeff[i] = 0.0;
        }
        res.deg = p1->deg + p2->deg;
        for (k = 0; k <= res.deg; k++) {
                res.coeff[k] = 0;
                for (i = 0; i <= p1->deg; i++) {
                        for (j = 0; j <= p2->deg; j++) {
                                if (i+j == k)
                                        res.coeff[k] += p1->coeff[i] * p2->coeff[j];
                        }
                }
        }

        return res;
}
```

Esercizio (tdeB 21/7/2008)

Si considerino due vettori di interi A e B. I valori degli elementi di ciascun vettore possono essere interpretati come indici per l'altro vettore (naturalmente rappresentano un riferimento "errato" se il valore è negativo o superiore alla dimensione dell'altro vettore). Seguendo i riferimenti validi, si può così "oscillare" da un vettore all'altro, finché un riferimento errato non interrompe l'oscillazione.

La funzione `int ciclici(int A[], int a, int B[], int b)` riceve in input due vettori e le loro lunghezze, e restituisce 1 se, **iniziando da A[0]**, l'oscillazione continua indefinitamente, 0 se invece si interrompe per un riferimento errato. I due esempi seguenti mostrano un caso ciclico e uno aciclico.





Suggerimenti: si utilizzi un'opportuna struttura dati supplementare (oltre ai due vettori) per riconoscere la situazione di "ciclo" (la "visita" dei vettori inizia a tornare su elementi già visitati); si presti anche attenzione ad evitare che, nel caso ciclico, la funzione continui a seguire il ciclo senza terminare.

Alloco subito un vettore di valori booleani, inizializzati a false, di dimensione pari al vettore A. Metterò man mano a true i valori corrispondenti agli elementi visitati del vettore A. Non serve fare l'analogo per B, perché se vi è un ciclo esso comprende almeno un elemento di A, e quindi sarà individuato comunque.

Aggiorno progressivamente due indici ia e ib relativi ai vettori A e B rispettivamente, iniziando con ia = 0 per accedere innanzitutto a A[0]. Ad ogni assegnamento di ia e ib verifico che il riferimento non sia errato. Se è errato, posso restituire subito 0. Per ogni assegnamento di ia verifico anche se A[ia] è già stato visitato; in tal caso, restituisco 1.

Non possono darsi altre situazioni, dunque la funzione termina sempre.

```
int ciclici( int A[], int a, int B[], int b ) {
        int i, ia, ib;
        int v[N];
        for ( i = 0; i < a; i++ )
                v[i] = 0;
        ia = 0;
        while ( 1 ) {
                if( v[ia]==1 ) {        /* ciclico, siamo già passati per A[ia] */
                        return 1;
                }
                v[ia] = 1;
                ib = A[ia];
                if( ib < 0 || ib >= a ) {
                        return 0;
                }
                ia = B[ib];
                if( ia < 0 || ia >= b ) {
                        return 0;
                }
        }
}
```

Esercizio (tde 11-11-2011)

- Un numero si dice doppiamente primo se è primo e tutte le sue cifre sono numeri primi.
- Si scriva una funzione f che riceve una matrice NxN di interi e restituisce 1 se tutti i numeri doppiamente primi che contiene sono circondati solo da numeri che non sono doppiamente primi nelle otto direzioni possibili per le caselle centrali, nelle cinque o tre possibili per le caselle sul bordo della matrice, 0 altrimenti.

  int f(int M[][N]);
- Un insieme di celle di una matrice si dice connesso se esiste un percorso che partendo da una delle celle permette di raggiungere le altre muovendosi di un passo alla volta nelle otto direzioni attraversando solo celle dell'insieme stesso.
- Si scriva una funzione g che riceve una matrice NxN di interi e restituisce 1 se tutti i numeri doppiamente primi che contiene appartengono a un unico insieme connesso di celle, 0 altrimenti.

  int g(int M[][N]);

```
int primo(int n) {
        int j;
        for (j = 2; j <= n/2; j++) {
                if (n % j == 0)
                        return 0;
        }
        return 1;
}

int DP(int n) {
        int cifra,num=n;

        if(primo(n)==0)
                return 0;

        for(num=n;num!=0; num=num/10) {
                cifra=num%10;
                if(primo(cifra)==0)
                        return 0;
        }
        return 1;
}
```

```
int f(int M[][N]) {
  int i,j,k,t;
  for(i=0; i < N;i++) {
            for(j=0; j < M;j++) {
                        if(dp(M[i][j])){
                                    for(k=i-1; k <= i+1;k++) {
                                                for(t=j-1; t <= j+1;t++) {
                                                            if(k < 0 || k >= N || t < 0 || t >= M)
                                                              ; //non fare nulla
                                                            else if(!(k==i && t==j) && dp(m[k][t]))
                                                                        return 0;
                                                }
                                    }
                        }
            }
    }
    return 1;
}
```

```
int g(int M[][N]) {
 int i,j,k,t,primo=1,aux[N][N];       //aux serve per marcare i numeri
                                      // ha tante caselle quante M
 int cambiato=1;
 //scorro tutta la matrice mettendo in aux un 2 in corrispondenza del primo numero
 //doppiamente primo che trovo, un 1 in corrispondenza di tutti gli altri numeri
 //doppiamente primi, 0 in corrispondenza di numeri non doppiamente primi
 for(i=0; i < N;i++) {
          for(j=0; j < N;j++) {
                   if(dp(M[i][j])) {
                            if(primo){
                                     primo=0;
                                     aux[i][j]=2;
                            }
                            else
                                     aux[i][j]=1;
                   }
                   else
                            aux[i][j]=0;
          }
  }
```

```
while(cambiato==1) { //propago il 2 in aux trasformando in 2 tutti gli 1 vicini a un 2
   cambiato=0;
   for(i=0; i < N;i++)
         for(j=0; j < N;j++)
               if(aux[i][j]==1)
                     for(k=i-1; k <= i+1;k++)
                           for(t=j-1; t <= j+1;t++)
                                 if(k < 0 || k >= N || t < 0 || t >= N)
                                       ; //non fare nulla
                                 else if(!(k==i && t==j) && aux[k][t]==2) {
                                       aux[i][j]=2;
                                       cambiato=1;
                                 }
 }
 // se è rimasto un 1 in aux vuol dire che esiste un doppiamente primo non connesso
 for(i=0; i < N;i++)
         for(j=0; j < N;j++)
               if(aux[i][j]==1)
                     return 0;
 return 1;
}
```

Esercizio (tde 19-11-2012)

- Si dica cosa stampa il seguente codice:

```
#define N 5
#include<stdio.h>
#include<conio.h>

int f(int a, int b);

int main(){
        int matrice[N][N],i,j;
        for(i=0; i<N; i++)
                for(j=0; j<N; j++)
                        if( j==0 || i+(j%j)==0 ){
                                matrice[i][j]=f(i+j,(i+j)%N);
                        } else
                                matrice[i][j]=i+j;
        for(i=0; i<N; i++) {
                for(j=0; j<N; j++)
                        printf("%d ",matrice[i][j]);
                printf("\n");
        }
        getch();
        return 1;
}
```

```
int f(int a, int b){
        int i;
        for(i=0; i<b; i++)
                a = 3*a;
        return a;
}
```

Stampa:

0	3	18	81	324
3	2	3	4	5
18	3	4	5	6
81	4	5	6	7
324	5	6	7	8

Esercizio (tde 19-11-2012)

- Si considerino le seguenti dichiarazioni di tipi e variabili, che definiscono le strutture dati per rappresentare il livello delle sostanze inquinanti di Milano.

```
typedef struct {    int giorno,mese,anno; } data;
typedef struct {    data giorno;
                    float livelloPM10;
                    float livelloBenzene;
                    float livelloAnidrideSolforosa; } rilievoGiornaliero;
typedef struct {    char quartiere[100]; /* nome del quartiere */
                    rilievoGiornaliero rilievi[3653]; /* 3653 rilievi per 10 anni */
} rilieviQuartiere; /* dati monitorati nell'anno per un singolo quartiere */
typedef rilieviQuartiere rilieviGlobali[100];
```

- Si definisca una funzione di prototipo

  int f(rilieviGlobali rg, float sogliaPM10, float sogliaB, float sogliaAS);

- che restituisce il massimo numero di giorni in cui in uno stesso quartiere il livello di almeno due sostanze ha superato le rispettive soglie. Si consiglia di spezzare la soluzione del problema in più funzioni.

```
int supera(int livello,int soglia){
        if(livello>soglia)
                return 1;
        else
                return 0;
}

int superaDue(rilievoGiornaliero r, float sogliaPM10, float sogliaB, float sogliaAS){
        if(supera(r.livelloPM10,sogliaPM10)+supera(r.livelloBenzene,sogliaB)+
                                        supera(r.livelloAnidrideSolforosa,sogliaAS)>=2)
                return 1;
        else
                return 0;
}
```

```
int contaQuartiere(rilieviQuartiere rq, float sogliaPM10, float sogliaB, float sogliaAS){
    int i,n=0;
    for(i=0;i<3653;i++){
        if(superaDue(rq.rilievi[i], sogliaPM10, sogliaB, sogliaAS))
            n++;
    }
    return n;
}

int f(rilieviGlobali rg, float sogliaPM10, float sogliaB, float sogliaAS){
    int i, max, imax=0, temp;

    max=contaQuartiere(rg[0], sogliaPM10, sogliaB, sogliaAS);

    for(i=1;i<100;i++){
        temp=contaQuartiere(rg[i], sogliaPM10, sogliaB, sogliaAS);
        if(temp>max){
            max=temp;
            imax=i;
        }

}
```

Esercizio (tde 19-11-2012)

- Dopo che il codificatore ha composto il codice, il decodificatore fa il suo primo tentativo, cercando di indovinare il codice. Il codificatore, appena il suo avversario ha completato il tentativo, fornisce degli aiuti comunicando: il numero di cifre giuste al posto giusto, cioè le cifre del tentativo che sono effettivamente presenti nel codice al posto tentato con degli 1 e il numero di cifre giuste al posto sbagliato (cioè le cifre del tentativo che sono effettivamente presenti nel codice ma non al posto tentato) con dei 2.
- Non bisogna comunicare quali cifre sono giuste o sbagliate ma solo quante.
- Si scriva una funzione che riceve in input tre array di dimensione 4 che rappresentano 1) il codice da decodificare, 2) il tentativo di indovinarlo e 3) gli aiuti dati in risposta al tentativo.
- void f(int codice[], int tentativo[], int risposta[])
- La funzione compara l'array tentativo con l'array codice e riempie l'array risposta in base alle regole del mastermind (nell'array risposta le caselle che non devono essere riempite siano messe tutte a 0).

- Esempi
- Se codice=”2 4 6 3” e tentativo=”1 9 5 7”, risposta dovrà contenere “0 0 0 0”
- Se codice=”2 4 6 3” e tentativo=”2 6 1 7”, risposta dovrà contenere “1 2 0 0”
- Se codice=”2 4 6 3” e tentativo=”7 5 6 3”, risposta dovrà contenere “1 1 0 0”
- Se codice=”2 4 6 3” e tentativo=”2 4 6 3”, risposta dovrà contenere “1 1 1 1”
- Se codice=”2 4 6 3” e tentativo=”6 3 4 2”, risposta dovrà contenere “2 2 2 2”
- Se codice=”2 4 6 3” e tentativo=”7 3 9 3”, risposta dovrà contenere “1 0 0 0”
- Se codice=”2 4 4 3” e tentativo=”6 3 4 2”, risposta dovrà contenere “1 2 2 0”
- Se codice=”2 4 4 3” e tentativo=”2 9 2 0”, risposta dovrà contenere “1 0 0 0”

```
void f(int codice[], int tentativo[], int risposta[]){
        int i, j, quantePosEsatta, quantePosErrata;
        int trovatoC[4]={0,0,0,0};
        int trovatoT[4]={0,0,0,0};

        //conto quanti in posizione esatta
        for(i=0;i<4;i++){
                if(codice[i]==tentativo[i]){
                        quantePosEsatta++;
                        trovatoC[i]=1;
                        trovatoT[i]=1;
                }
        }
```

```
//conto quanti in posizione errata
for(i=0;i<4;i++){
    if(trovatoC[i]==0) {
        for(j=0;j<4;j++){
            if(codice[i]==tentativo[j] && trovatoT[j]==0){
                quantePosErrata++;
                trovatoC[i]=1;
                trovatoT[j]=1;
                break;
            }
        }
    }
}
```

```
    for(i=0;i<quantePosEsatta;i++){
        risposta[i]=1;
    }

    for(;i<quantePosErrata+quantePosEsatta;i++){
        risposta[i]=2;
    }

    for(;i<4;i++){ //NO INIZIALIZZAZIONE
        risposta[i]=0;
    }
}
```