```
/*
 * Corso di Fondamenti di Informatica
 * Esercizio:
 * calcolo del determinante di una matrice
 */

/* NOTA: il determinante di una matrice quadrata A è la somma dei prodotti
 * degli elementi di una riga (o colonna) per i determinanti dei rispettivi
 * minori complementari e per i rispettivi complementi algebrici.
 * Il minore complementare dell'elemento della i-esima riga e j-esima colonna
 * è la matrice ottenuta eliminando dalla matrice originale tale riga e tale
 * colonna; il complemento algebrico è pari a -1 elevato a (i+j). */

/* #define DEBUG */
/* se questa direttiva non viene commentata via, vengono eseguite le parti di
 * codice comprese tra #ifdef DEBUG e #endif */

#include <stdio.h>

#define MAX_DIM 12
/* dimensione massima della matrice (quadrata) */

typedef double ElementiMatrice[MAX_DIM][MAX_DIM];
/* tipo usato per memorizzare gli elementi di matrici quadrate di dimensioni
 * non superiori a MAX_DIM x MAX_DIM, usando tutto o parte dell'array
 * bidimensionale. Il primo indice rappresenta la riga, il secondo la
 * colonna. */

typedef struct
{
  ElementiMatrice Dati;
  /* contiene gli elementi della matrice (e altri dati non significativi) */
  int Dim;
  /* dimensione della matrice, che identifica la parte utile del campo Dati */

} Matrice;
/* usato per memorizzare una matrice quadrata di dimensioni non superiori a
 * MAX_DIM x MAX_DIM. Per matrici di dimensione inferiore a MAX_DIM, solo parte
 * del campo Dati e' utilizzata. Gli elementi utili sono quelli della matrice
 * quadrata compresa tra l'elemento [0][0] e l'elemento [Dim-1][Dim-1] */
```

```
double CalcolaDet(Matrice MatrDaElab);
/* data una matrice, ne calcola il determinante operando in modo ricorsivo */

Matrice CalcolaMinore(Matrice MatrOrig, int RigaDaElim, int ColDaElim);
/* data una matrice quadrata di dimensione N, genera la matrice di dimensione
 * N-1 ottenuta eliminando da MatrOrig la riga di indice RigaDaElim e la
 * colonna di indice ColDaElim (minore complementare) */

void LeggiMatrice(Matrice* pMatrDaLegg);
/* legge da tastiera gli elementi di una matrice quadrata di dimensione non
 * superiore a MAX_DIM */

void StampaMatrice(Matrice MatrDaStamp);
/* stampa gli elementi di una matrice quadrata di dimensione non superiore a
 * MAX_DIM */


int main ()
{
  Matrice MatriceOrig;   /* matrice di cui calcolare il determinante */

  printf("\n\nCalcolo del determinante di una matrice.\nInserimento della matrice.");

  printf("\nNumero di elementi per riga/colonna [1-%d]?  ", MAX_DIM);
  scanf ("%d", &MatriceOrig.Dim);

  LeggiMatrice(&MatriceOrig);

  printf("\n\nIl determinante della matrice");
  StampaMatrice(MatriceOrig);
  printf("\n\nvale:  %f\n\n", CalcolaDet(MatriceOrig));

  return(0);
}


double CalcolaDet(Matrice MatrDaElab)
{
  int ContEl;
  /* usato per scandire gli elementi di una riga della matrice */
  int ComplAlg;
```

```
  /* complemento algebrico: e' un coefficiente che assume il valore +1 o -1 */
  Matrice MatrRidotta;
  /* contiene uno dei minori complementari */
  double Determinante;
  /* il determinante della matrice */

#ifdef DEBUG
  printf("\n\n### Sto Calcolando il determinante della matrice %d x %d", MatrDaElab.Dim, MatrDaElab.Dim);
  StampaMatrice(MatrDaElab);
#endif

  if ( 1 == MatrDaElab.Dim )
  /* caso elementare della ricorsione */
  {
    Determinante = MatrDaElab.Dati[0][0];
  }
  else
  {
    /* uso la prima riga di MatrDaElab per calcolarne il determinante */
    Determinante = 0;
    ComplAlg = +1;

    for (ContEl = 0; ContEl < MatrDaElab.Dim; ++ContEl)
    {
      MatrRidotta = CalcolaMinore(MatrDaElab, 0, ContEl);
      Determinante = Determinante + ComplAlg * MatrDaElab.Dati[0][ContEl] *
                     CalcolaDet(MatrRidotta);
      /* chiamata ricorsiva a CalcolaDet */

      ComplAlg = -1 * ComplAlg;
    }
  }

  return (Determinante);
}


Matrice CalcolaMinore(Matrice MatrOrig, int RigaDaElim, int ColDaElim)
{
  int ROrig, COrig;
  /* scandiscono righe e colonne della matrice originale */
```

```
  int RRid, CRid;
  /* scandiscono righe e colonne della matrice ridotta */
  Matrice MatrRid;
  /* la matrice ridotta da restituire */

  MatrRid.Dim = MatrOrig.Dim - 1;
  RRid = 0;

  for ( ROrig = 0; ROrig < MatrOrig.Dim; ++ROrig )
  {
    if ( RigaDaElim != ROrig )   /* se non è la riga da eliminare */
    {
      CRid = 0;

      /* copia gli elementi della riga, escluso quello nella colonna da
       * eliminare */
      for ( COrig = 0; COrig < MatrOrig.Dim; ++COrig )
      {
        if ( ColDaElim != COrig )  /* se non è la colonna da eliminare */
        {
          /* copia l'elemento corrente */
          MatrRid.Dati[RRid][CRid] = MatrOrig.Dati[ROrig][COrig];
          ++CRid;
        }
      }

      ++RRid;
    }
  }

  return(MatrRid);
}


void LeggiMatrice(Matrice* pMatrDaLegg)
{
  int ContR, ContC;
  /* contatori usati per scorrere righe e colonne */

  printf("\n");
  for ( ContR = 0; ContR < pMatrDaLegg->Dim; ++ContR )
```

```
  /* scandisce le righe */
  {
    for ( ContC = 0; ContC < pMatrDaLegg->Dim; ++ContC )
    /* scandisce le colonne */
    {
      printf("Valore elemento (%d, %d) della matrice?  ", ContR+1, ContC+1);
      scanf("%lf", &(pMatrDaLegg->Dati[ContR][ContC]));
    }
  }
}


void StampaMatrice(Matrice MatrDaStamp)
{
  int ContR, ContC;
  /* contatori usati per scorrere righe e colonne */

  printf("\n");
  for ( ContR = 0; ContR < MatrDaStamp.Dim; ++ContR )
  {
    printf("\n");
    for ( ContC = 0; ContC < MatrDaStamp.Dim; ++ContC )
    {
      printf("\t");
      printf("%f", MatrDaStamp.Dati[ContR][ContC]);
    }
  }

  printf("\n");
}
```