

4.1 Array

Quando si ha la necessità di trattare un insieme omogeneo di dati esiste una soluzione diversa da quella di utilizzare tante variabili dello stesso tipo: definire un *array*, ovvero una variabile strutturata dove è possibile memorizzare più valori tutti dello stesso tipo. Intuitivamente, un array monodimensionale o *vettore* può essere immaginato come un contenitore suddiviso in tanti scomparti quanti sono i dati che vi si vogliono memorizzare. Ognuno di questi scomparti, detti *elementi* del vettore, contiene un unico dato ed è individuato da un numero progressivo, detto *indice*, che specifica la posizione dell'elemento all'interno del vettore stesso. L'indice può assumere valori interi da zero al numero totale di elementi meno 1. L'indice di base dell'array è sempre zero. Il numero complessivo degli elementi del vettore viene detto *lunghezza*.

In Figura 4.1, per esempio, l'elemento di indice 2 o, più brevemente, il terzo elemento del vettore contiene il carattere Q, il carattere S è contenuto nell'elemento di indice 0 e il vettore ha lunghezza 4.

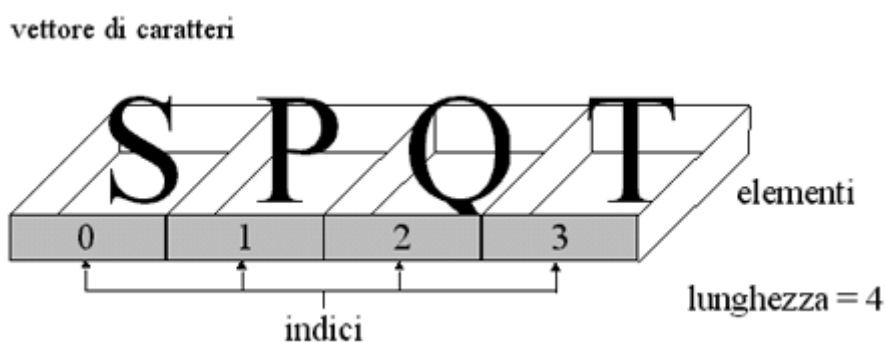


Figura 4.1 Rappresentazione intuitiva di un vettore

In definitiva, un vettore è una struttura di dati composta da un numero determinato di elementi tutti dello stesso tipo, ognuno dei quali è individuato da un indice specifico. È ora chiaro perché i vettori si dicano *variabili strutturate* mentre all'opposto tutte le variabili semplici siano anche dette *non strutturate*. Il tipo dei dati contenuti nel vettore viene detto *tipo del vettore*, ovvero si dice che il vettore è di quel particolare tipo.

Dunque per il vettore, come per qualsiasi altra variabile, devono essere definiti il nome e il tipo; inoltre si deve esplicitarne la lunghezza, cioè il numero di elementi che lo compongono. Una scrittura possibile è perciò la seguente:

```
int a[6];
```

Come sempre in C, prima deve essere dichiarato il tipo (nell'esempio `int`), poi il nome della variabile (`a`), successivamente – tra parentesi quadre – il numero degli elementi (6) che dev'essere un intero positivo. Questa dichiarazione permette di riservare in memoria centrale uno spazio strutturato come in Figura 4.2.



Figura 4.2 Struttura dell'array `a[6]`

Per accedere a un singolo elemento di `a` si deve specificare il nome del vettore seguito dall'indice dell'elemento posto tra parentesi quadre. L'array `a` è composto da sei elementi e l'indice può quindi assumere i valori: 0, 1, 2, 3, 4, e 5. Le istruzioni

```
a[0] = 71;  
a[1] = 4;
```

assegnano al primo elemento del vettore `a` il valore 71 e al secondo 4. Se `b` è una variabile intera (cioè dello stesso tipo del vettore), è possibile assegnare il suo valore a un elemento di `a` e viceversa:

```
a[3] = b;
```

In generale un singolo elemento dell'array può essere utilizzato esattamente come una variabile semplice. Nell'espressione

```
b = b + a[0] * a[5];
```

il valore di `b` è sommato al prodotto tra il primo e il sesto elemento di `a` e il risultato è assegnato a `b`.

Spesso l'array viene trattato all'interno di iterazioni; infatti risulta semplice far riferimento a suoi elementi incrementando ciclicamente il valore di una variabile intera e utilizzandola come indice. Consideriamo un'iterazione di esempio:

```
/* Inizializzazione dell'array */
for(i=0; i<=5; i++) {
    printf("Inser. intero: ");
    scanf("%d", &a[i]);
}
```

L'indice `i` dell'array `a` è inizializzato a 0 (si veda il Paragrafo 4.3) e assume a ogni iterazione successiva i valori 1, 2, 3, 4, 5. Il blocco del `for` richiede all'utente l'immissione di sei valori che vengono assegnati sequenzialmente, mediante l'istruzione `scanf`, agli elementi del vettore. Se quindi vengono inseriti in sequenza i valori 9, 18, 7, 15, 21 e 11, dopo l'esecuzione del ciclo il vettore si presenterà in memoria come in Figura 4.3.

9	18	7	15	21	11
0	1	2	3	4	5

Figura 4.3 L'array `a[6]` dopo la sua inizializzazione

Anche per ricercare all'interno dell'array valori che soddisfano certe condizioni si utilizzano abitualmente i cicli:

```
/* Ricerca del maggiore */
max = a[0];
for(i=1; i<=5; i++)
    if(a[i]>max) max = a[i];
```

L'esempio permette di determinare il maggiore degli elementi dell'array `a`: la variabile `max` viene inizializzata al valore del primo elemento del vettore, quello con indice zero. Successivamente ogni ulteriore elemento viene confrontato con `max`: se risulta essere maggiore, il suo valore viene assegnato a `max`. Il ciclo deve comprendere dunque tutti gli elementi del vettore meno il primo, perciò l'indice `i` assume valori che vanno da 1 a 5.

Nel Listato 4.1 è riportato un programma che richiede all'utente l'immissione del punteggio raggiunto da sei studenti, li memorizza nel vettore `voti` e ne determina il maggiore, il minore e la media.

```
/* Memorizza in un array di interi il punteggio raggiunto da sei
   studenti e ne determina il maggiore, il minore e la media */
#include <stdio.h>

main()
{
    int voti[6];
    int i, max, min;
    float media;

    printf("VOTI STUDENTI\n\n");
    /* Immissione voti */
    for(i=0; i<=5; i++) {
        printf("Voto %d° studente: ", i+1);
```

```

scanf("%d", &voti[i]);
}

/* Ricerca del maggiore */
max = voti[0];
for(i=1; i<=5; i++)
    if(voti[i]>max)
        max = voti[i];

/* Ricerca del minore */
min = voti[0];
for(i=1; i<=5; i++)
    if(voti[i]<min)
        min = voti[i];

/* Calcolo della media */
media = voti[0];
for(i=1; i<=5; i++)
    media = media + voti[i];
media = media/6;

printf("Maggiore: %d\n", max);
printf("Minore: %d\n", min);
printf("Media: %f\n", media);
}

```

Listato 4.1 Esempio di utilizzo di una variabile array

Si noti che la richiesta dei voti all'utente viene fatta evidenziando il numero d'ordine che corrisponde al valore dell'indice aumentato di una unità.

```
printf("Voto %d° studente: ", i+1);
```

Alla prima iterazione appare sullo schermo:

```
Voto 1° studente:
```

Considerazioni analoghe a quelle fatte per il calcolo del maggiore valgono per il minimo e la media.

Nella pratica la memorizzazione in un vettore ha senso quando i valori debbono essere utilizzati più volte, come nel caso precedente. Nell'esempio, comunque, i calcoli potevano essere effettuati all'interno della stessa iterazione con un notevole risparmio di tempo di esecuzione:

```

/* Ricerca maggiore, minore e media */
max = voti[0];
min = voti[0];
media = voti[0];
for(i = 0; i <= 5; i++) {
    if(voti[i] > max)
        max = voti[i];
    if(voti[i] < min)
        min = voti[i];
    media = media+voti[i];
}
media = media / 6;

```

✓ **NOTA**

È importante ricordare che in C l'indice inferiore del vettore è zero e quello superiore è uguale al numero di elementi meno 1: se si desidera un array di 100 si dichiara

```
voti[100];
```

ma si deve tenere presente che l'indice assume valori da 0 a 99.

Uno degli errori più ricorrenti nella programmazione in C è lo sconfinamento dei limiti degli array. Non sempre è semplice rintracciare tali errori poiché i compilatori non li segnalano. I controlli necessari in questo senso sono a carico del programmatore.

Dopo la dichiarazione del tipo possono essere presenti i nomi di più variabili di quel tipo, semplici o strutturate. Per esempio

```
int i, voti[6], max, min, somma;
```

dichiara le variabili intere `i`, `max`, `min`, `somma` e la variabile array di tipo intero `voti`.

4.2 Esempi di uso di array

Per determinare la destrezza di n concorrenti sono state predisposte due prove, entrambe con una valutazione che varia da 1 a 10; il punteggio totale di ogni concorrente è dato dalla media aritmetica dei risultati delle due prove. Si richiede la visualizzazione di una tabella che contenga su ogni linea i risultati parziali e il punteggio totale di un concorrente. Nel Listato 4.2 è mostrato il programma relativo.

```
/* Carica i punteggi di n concorrenti su due prove
   Determina la classifica */
#include <stdio.h>

#define MAX_CONC 1000 /* massimo numero di concorrenti */
#define MIN_PUN 1 /* punteggio minimo per ogni prova */
#define MAX_PUN 10 /* punteggio massimo per ogni prova */

main()
{
float prova1[MAX_CONC], prova2[MAX_CONC], totale[MAX_CONC];
int i, n;

do {
printf("\nNumero concorrenti: ");
scanf("%d", &n);
}
while(n<1 || n>MAX_CONC);

/* Per ogni concorrente, richiesta punteggio nelle due prove */
for(i=0; i<n; i++) {
printf("\nConcorrente n.%d \n", i+1);

do {
printf("Prima prova: ");
scanf("%f", &prova1[i]);
}
while(prova1[i]<MIN_PUN || prova1[i]>MAX_PUN);

do {
printf("Seconda prova: ");
scanf("%f", &prova2[i]);
}
while(prova2[i]<MIN_PUN || prova2[i]>MAX_PUN);
```

```

}

/* Calcolo media per concorrente */
for(i=0; i<n; i++)
    totale[i] = (prova1[i]+prova2[i])/2;

printf("\n      CLASSIFICA\n");
for(i=0; i<n; i++)
    printf("%f %f %f \n", prova1[i], prova2[i], totale[i]);
}

```

Listato 4.2 Esempio di utilizzo di un array

Non conoscendo a priori il numero di concorrenti che parteciperanno alle gare si fa l'ipotesi che comunque non siano più di 1000, valore che memorizziamo nella costante `MAX_CONC`. In conseguenza di ciò definiamo di lunghezza `MAX_CONC` gli array che conterranno i risultati: `prova1`, `prova2` e `totale`. Richiediamo all'utente a tempo di esecuzione il numero effettivo dei concorrenti e verificiamo che non sia minore di 1 e maggiore di `MAX_CONC`:

```

do {
    printf("\nNumero concorrenti: ");
    scanf("%d", &n);
}
while(n<MIN_PUN || n>MAX_CONC);

```

In seguito richiediamo l'introduzione dei risultati della prima e della seconda prova di ogni concorrente, controllando che tale valutazione non sia minore di 1 e maggiore di 10, nel qual caso ripetiamo la richiesta. Abbiamo memorizzato in `MIN_PUN` e `MAX_PUN` i limiti inferiore e superiore del punteggio assegnabile, in maniera che, se questi venissero modificati, basterebbe intervenire sulle loro definizioni perché il programma continui a funzionare correttamente. Infine calcoliamo il punteggio totale e lo visualizziamo. Un esempio di esecuzione è mostrato in Figura 4.4.

✓ NOTA

Si noti che soltanto n elementi di ogni array vengono utilizzati veramente; quindi se, per esempio, i concorrenti sono dieci, si ha un utilizzo di memoria pari a solo il 10 per mille (valore attuale di `MAX_CONC`); in questo modo però il programma è più flessibile.

Esistono altre soluzioni più complesse che permettono di gestire la memoria dinamicamente (cioè di adattarla alle effettive esigenze del programma), anziché staticamente (cioè riservando a priori uno spazio): le vedremo in seguito.

```

Numero concorrenti: 3

Concorrente n.1
Prima prova: 8   Seconda prova: 7

Concorrente n.2
Prima prova: 5   Seconda prova: 9

Concorrente n.3
Prima prova: 8   Seconda prova: 8

      CLASSIFICA

8.000000  7.000000  7.500000

```

```
5.000000  9.000000  7.000000
8.000000  8.000000  8.000000
```

Figura 4.4 Esempio di esecuzione del programma del Listato 4.2

4.3 Inizializzazione di variabili

L'inizializzazione di una variabile può essere esplicitata direttamente al momento della sua dichiarazione, come con

```
int i = 0;
```

L'istruzione dichiara la variabile `i` di tipo intero e le assegna il valore zero. Di seguito alla dichiarazione di tipo possono essere definite e inizializzate più variabili:

```
int a = 50, b = 30, c, d = 333;
```

definisce le variabili intere `a`, `b`, `c` e `d`; inizializza `a` al valore 50, `b` a 30, `d` a 333, `c` non è inizializzata. Analogamente si possono assegnare valori agli altri tipi di variabili semplici:

```
float x = 567.8927;
float y = 7e13;
char risposta = 's';
```

Per gli array l'inizializzazione è possibile solamente se sono stati dichiarati come `extern` o come `static`; quest'ultima classe di variabile verrà esaminata in seguito.

Le variabili `extern` sono quelle che vengono definite prima di `main`. L'inizializzazione si ottiene inserendo i valori tra parentesi graffe, separati da una virgola:

```
int voti[6] = {11, 18, 7, 15, 21, 9};
```

Il compilatore fa la scansione dei valori presenti tra parentesi graffe da sinistra verso destra e genera altrettanti assegnamenti consecutivi agli elementi del vettore, rispettando la loro posizione; dunque `voti[0]` assume il valore 11, `voti[1]` 18, `voti[2]` 7 ecc. Quando tutti gli elementi dell'array vengono inizializzati è possibile omettere l'indicazione del numero di elementi, e scrivere

```
int voti[] = {11, 18, 7, 15, 21, 9};
```

È infatti il compilatore stesso che conta i valori e di conseguenza determina la dimensione del vettore.

Gli array di caratteri, comunemente detti *stringhe*, possono essere inizializzati anche inserendo il loro contenuto tra doppi apici:

```
char frase[] = "Analisi, requisiti";
```

4.4 Matrici

Nei paragrafi precedenti abbiamo trattato i vettori, detti anche matrici monodimensionali. Per la memorizzazione abbiamo usato una variabile di tipo array dichiarandone il numero di componenti, per esempio:

```
int vet[3];
```

Per accedere direttamente a ciascuno degli elementi del vettore si è utilizzato un indice che varia da zero a $n-1$. Nell'esempio n è uguale a 3.

In una matrice bidimensionale i dati sono organizzati per righe e per colonne, come se fossero inseriti in una tabella. Per la memorizzazione si utilizza una variabile di tipo array specificando il numero di componenti per ciascuna delle due dimensioni che la costituiscono:

```
int mat[4][3];
```

La variabile strutturata `mat` che abbiamo dichiarato contiene 4 righe e 3 colonne per un totale di dodici elementi; per accedere a ciascuno di essi si utilizzano due indici: il primo specifica la riga il secondo la colonna. Gli indici variano rispettivamente tra 0 e $r-1$ e tra 0 e $c-1$, dove r e c sono il numero di righe e il numero di colonne. Abbiamo cioè

```
mat[0][0] mat[0][1] mat[0][2]
mat[1][0] mat[1][1] mat[1][2]
mat[2][0] mat[2][1] mat[2][2]
mat[3][0] mat[3][1] mat[3][2]
```

Per esempio, `mat[1][2]` fa riferimento all'elemento presente nella seconda riga della terza colonna. Ogni colonna della matrice bidimensionale non è altro che un vettore.

Il formato generale della dichiarazione degli array multidimensionali è il seguente:

```
tipo nome[dimensione1][dimensione2]...[dimensioneN];
```

Per esempio, al fine di memorizzare i ricavi ottenuti dalla vendita di 10 prodotti in 5 punti vendita nei dodici mesi dell'anno, potremmo utilizzare la matrice tridimensionale `marketing` così dichiarata:

```
int marketing[10][5][12]
```

Scriviamo ora un programma che richiede all'utente i valori da inserire, li memorizza nella matrice bidimensionale `mat` e la visualizza (Listato 4.3).

```
/* Caricamento di una matrice */
#include <stdio.h>

int mat[4][3];

main()
{
    int i, j;

    printf("\n \n CARICAMENTO DELLA MATRICE \n \n");
    for(i=0; i<4; i++)
        for(j=0; j<3; j++) {
            printf("Inserisci linea %d colonna %d val: ", i, j);
            scanf("%d", &mat[i][j]);
        };

    /* Visualizzazione */
    for(i=0; i<4; i++) {
        printf("\n");
        for(j=0; j<3; j++)
            printf("%5d", mat[i][j]);
    }
}
```

Listato 4.3 Esempio di utilizzo di un array bidimensionale

Per effettuare il caricamento dei dati nella matrice utilizziamo due cicli, uno più esterno che mediante la variabile `i` fa la scansione delle righe da zero a 3 (4-1) e un altro che percorre, per mezzo della variabile `j`, le colonne da zero a 2 (3-1):

```
for(i=0; i<4; i++)
    for(j=0; j<3; j++) {
        printf("Inserisci linea %d colonna %d val:", i, j);
        scanf("%d", &mat[i][j]);
    };
```

Viene riempita tutta la prima riga poi la seconda e così via; se l'utente passa i valori 2, 55, 12, 98, 34... essi verranno inseriti negli elementi `mat[0][0]`, `mat[0][1]`, `mat[0][2]`, `mat[1][0]`, `mat[1][1]`...

Si può ottenere il caricamento per colonne invertendo semplicemente i due cicli:

```
for(j=0; j<3; j++)
    for(i=0; i<4; i++) {
```

```

    printf("Inserisci linea %d colonna %d val:", i, j);
    scanf("%d", &mat[i][j]);
};

```

Il numero totale d'iterazioni è sempre uguale a 12 e gli indici `j` e `i` fanno la scansione delle colonne e delle righe della matrice senza fuoriuscire dai margini. La visualizzazione è ancora una volta ottenuta con due cicli `for` annidati uno nell'altro.

Nel programma precedente le dimensioni della matrice erano fissate a priori: modifichiamolo in modo da far decidere all'utente il numero delle righe e delle colonne, come nel Listato 4.4.

```

/* Caricamento di una matrice
   le cui dimensioni vengono decise dall'utente */

#include <stdio.h>

#define MAXLINEE 100
#define MAXCOLONNE 100
int mat[MAXLINEE][MAXCOLONNE];

main()
{
int n, m;
int i, j;

/* Richiesta delle dimensioni */
do {
    printf("\nNumero di linee: ");
    scanf("%d", &n);
}
while((n>=MAXLINEE) || (n<1));

do {
    printf("Numero di colonne: ");
    scanf("%d", &m);
}
while((m>=MAXCOLONNE) || (m<1));

printf("\n \n CARICAMENTO DELLA MATRICE \n \n");
for(i=0; i<n; i++)
    for(j=0; j<m; j++) {
        printf("Inserisci linea %d colonna %d val:", i, j);
        scanf("%d", &mat[i][j]);
    };

/* Visualizzazione */
for(i=0; i<n; i++) {
    printf("\n");
    for(j=0; j<m; j++)
        printf("%5d", mat[i][j]);
}
}

```

Listato 4.4 Inizializzazione di una matrice bidimensionale, seconda versione

La matrice viene definita con un massimo numero di linee e di colonne:

```
int mat[MAXLINEE][MAXCOLONNE];
```

dove `MAXLINEE` e `MAXCOLONNE` sono due costanti che abbiamo dichiarato precedentemente, il cui valore deve essere scelto in relazione alle massime dimensioni. Successivamente si richiede l'inserimento del valore di n , numero di linee che realmente verranno riempite:

```
do {
    printf("Numero di linee: ");
    scanf("%d", &n);
}
while((n>=MAXLINEE) || (n<1));
```

L'istruzione `scanf` viene inserita in un ciclo `do-while` in modo che se n è maggiore del numero di linee che costituiscono la matrice o è minore di 1 il valore non viene accettato e la richiesta viene ripetuta. Analogamente si procede per le colonne.

4.5 Esempi con le matrici

Passiamo adesso a un problema più complesso: date due matrici `mat1[N][P]`, `mat2[P][M]` calcolare la matrice prodotto in cui ogni elemento è dato da:

$$pmat[i][j] = \sum_{k=1}^P mat1[i][k] * mat2[k][j]$$

per $i=1..N$, $j=1..M$

Il prodotto così definito si può ottenere soltanto se il numero di colonne della prima matrice (P) è uguale al numero di righe della seconda. La matrice `pmat` è dunque costituita da N righe e M colonne.

Consideriamo le matrici di Figura 4.5: l'elemento `[2][4]` della matrice prodotto è dato da

```
pmat[2][4] = mat1[2][0]*mat2[0][4] +
             mat1[2][1]*mat2[1][4] +
             mat1[2][2]*mat2[2][4]
```

ossia

```
pmat[2][4] = 5*3 + 2*4 + 0*5 = 23
```

Venendo dunque al programma richiesto (Listato 4.5), in primo luogo si devono caricare i dati delle due matrici.

```
/* Calcolo del prodotto di due matrici */
#include <stdio.h>

#define N 4
#define P 3
#define M 5

int mat1[N][P];          /* prima matrice */
int mat2[P][M];          /* seconda matrice */
int pmat[N][M];          /* matrice prodotto */

main()
{
    int i, j, k;

    printf("\n \n CARICAMENTO DELLA PRIMA MATRICE \n \n");
    for(i=0; i<N; i++)
        for(j=0; j<P; j++) {
            printf("Inserisci linea %d colonna %d val:", i, j);
            scanf("%d", &mat1[i][j]);
        }
}
```

```

};

printf("\n \n CARICAMENTO DELLA SECONDA MATRICE \n \n");
for(i=0; i<P; i++)
    for(j=0; j<M; j++) {
        printf("Inserisci linea %d colonna %d val:", i, j);
        scanf("%d", &mat2[i][j]);
    };

/* Calcolo del prodotto */
for(i=0; i<N; i++)
    for(j=0; j<M; j++) {
        pmat[i][j] = 0;
        for(k=0; k<P; k++)
            pmat[i][j] = pmat[i][j] + mat1[i][k] * mat2[k][j];
    };

printf("\n \n PRIMA MATRICE \n ");
for(i=0; i<N; i++) {
    printf("\n");
    for(j=0; j<P; j++)
        printf("%5d", mat1[i][j]);
}

printf("\n \n SECONDA MATRICE \n ");
for(i=0; i<P; i++) {
    printf("\n");
    for(j=0; j<M; j++)
        printf("%5d", mat2[i][j]);
}

printf("\n \n MATRICE PRODOTTO \n ");
for(i=0; i<N; i++) {
    printf("\n");
    for(j=0; j<M; j++)
        printf("%5d", pmat[i][j]);
}
}

```

Listato 4.5 Calcolo del prodotto tra matrici

Per ottenere il valore dell'elemento i, j della matrice prodotto lo si inizializza a zero:

```
pmat[i][j] = 0;
```

Successivamente, con un ciclo che fa la scansione della riga i di $mat1$ e della colonna j di $mat2$, si accumula in $pmat[i][j]$ la sommatoria dei prodotti dei corrispondenti elementi di $mat1$ e $mat2$:

```
for(k=0; k<P; k++)
    pmat[i][j] = pmat[i][j] + mat1[i][k] * mat2[k][j];
```

La variabile k permette di scorrere contemporaneamente la linea i di $mat1$ e la colonna j di $mat2$; il suo valore varia da 0 a P . Il procedimento appena visto va ripetuto per ognuno degli elementi della matrice prodotto:

```
for(i=0; i<N; i++)
    for(j=0; j<M; j++) {
        pmat[i][j] = 0;
        for(k=0; k<P; k++)
            pmat[i][j] = pmat[i][j] + mat1[i][k] * mat2[k][j];
    };

```

I due cicli for fissano a ogni iterazione una certa riga di mat1 e di pmat e una certa colonna di mat2 e di pmat.
Riportiamo in Figura 4.5 un esempio di esecuzione del programma.

CARICAMENTO DELLA PRIMA MATRICE

```
Inserisci linea 0 colonna 0 val:1
Inserisci linea 0 colonna 1 val:0
Inserisci linea 0 colonna 2 val:0
Inserisci linea 1 colonna 0 val:22
Inserisci linea 1 colonna 1 val:-6
Inserisci linea 1 colonna 2 val:3
Inserisci linea 2 colonna 0 val:5
Inserisci linea 2 colonna 1 val:2
Inserisci linea 2 colonna 2 val:0
Inserisci linea 3 colonna 0 val:11
Inserisci linea 3 colonna 1 val:4
Inserisci linea 3 colonna 2 val:7
```

CARICAMENTO DELLA SECONDA MATRICE

```
Inserisci linea 0 colonna 0 val:2
Inserisci linea 0 colonna 1 val:0
Inserisci linea 0 colonna 2 val:4
Inserisci linea 0 colonna 3 val:0
Inserisci linea 0 colonna 4 val:3
Inserisci linea 1 colonna 0 val:0
Inserisci linea 1 colonna 1 val:1
Inserisci linea 1 colonna 2 val:5
Inserisci linea 1 colonna 3 val:1
Inserisci linea 1 colonna 4 val:4
Inserisci linea 2 colonna 0 val:21
Inserisci linea 2 colonna 1 val:1
Inserisci linea 2 colonna 2 val:2
Inserisci linea 2 colonna 3 val:2
Inserisci linea 2 colonna 4 val:5
```

PRIMA MATRICE

1	0	0
22	-6	3
5	2	0
11	4	7

SECONDA MATRICE

2	0	4	0	3
0	1	5	1	4
21	1	2	2	5

MATRICE PRODOTTO

2	0	4	0	3
107	-3	64	0	57
10	2	30	2	23
169	11	78	18	84

4.6 Esercizi

- * 1. Scrivere un programma che, inizializzati in due vettori a e b della stessa lunghezza n valori interi, calcoli la somma incrociata degli elementi: $a[1] + b[n]$, $a[2] + b[n-1]$, ... la memorizzi nel vettore c e visualizzi quindi a , b e c .
- * 2. Modificare il programma, esaminato nel presente capitolo, che determina il maggiore, il minore e la media degli elementi di un array in modo che vengano diminuiti in media il numero di confronti effettuati nel ciclo durante l'esecuzione.
- 3. Scrivere un programma che inizializzi e quindi visualizzi un vettore con i valori alternati 0, 1, 0, 1, 0, 1, 0, 1, ... Ripetere l'esercizio con i valori 0, -3, 6, -9, 12, -15, 18, -21,
- 4. Scrivere un programma che, letti gli elementi di un vettore $v1$ e un numero k , determini l'elemento di $v1$ più prossimo a k .
- 5. Scrivere un programma che, letti gli elementi di due vettori $v1$ e $v2$ di lunghezza 5, determini il vettore w di lunghezza 10 ottenuto alternando gli elementi di $v1$ e $v2$. Visualizzare $v1$, $v2$ e w .
Per esempio: se $v1$ e $v2$ sono i vettori di caratteri

$v1$	B	N	S	I	O					
$v2$	E	I	S	M	!	si deve ottenere il vettore				
w	B	E	N	I	S	S	I	M	O	!

- 6. Scrivere un programma che, letti gli elementi di due vettori $v1$ e $v2$ di lunghezza n , inizializzi un terzo vettore w di lunghezza n con i valori
- $$\begin{aligned} w(i) &= 1 && \text{se } v1(i) > v2(i); \\ w(i) &= 0 && \text{se } v1(i) = v2(i); \\ w(i) &= -1 && \text{se } v1(i) < v2(i). \end{aligned}$$
- Visualizzare quindi $v1$, $v2$ e w .
- 7. Scrivere un programma che, inizializzato un vettore di `char` con una stringa di lettere dell'alfabeto e punteggiatura, visualizzi il numero complessivo delle vocali e delle consonanti del vettore.
 - 8. Scrivere un programma di inizializzazione che richieda un elemento controlli, prima di inserirlo nel vettore, se è già presente, nel qual caso chieda che l'elemento sia digitato di nuovo.
 - 9. Scrivere un programma che inizializzi e quindi visualizzi una matrice di `int` in cui ciascun elemento è dato dalla somma dei propri indici.
 - 10. [Matrici simmetriche] Una matrice quadrata $n \times n$ di un tipo qualsiasi si dice simmetrica se gli elementi simmetrici rispetto alla diagonale principale (dal vertice alto sinistro al vertice basso destro) sono due a due uguali. Scrivere un programma che, letta una matrice quadrata di interi, controlli se è simmetrica.
 - 11. Scrivere un programma che, inizializzata una matrice $n \times n$, visualizzi la matrice che si ottiene da quella data scambiando le righe con le colonne.
 - * 12. Modificare il programma che calcola il prodotto di due matrici bidimensionali esaminato nel presente capitolo, in modo che sia l'utente a scegliere le dimensioni degli array. Il programma deve verificare la correttezza delle dimensioni inserite.
 - 13. Scrivere un programma che, letta una matrice di interi o reali, individui la colonna con somma degli elementi massima.

- * 14. Scrivere un programma che richieda all'utente i voti delle otto prove sostenute durante l'anno da diciotto studenti di una classe e calcoli la media di ogni studente, la media di ogni prova e la media globale. Il programma dovrà infine visualizzare l'intera matrice e la media globale. [Suggerimento: si utilizzi una matrice di 19 linee e 9 colonne dove nelle prime otto vengono memorizzati in ciascuna linea i voti di uno studente e nella nona la rispettiva media; nella diciannovesima linea viene invece memorizzata la media per prova.]

15. Memorizzare in un array tridimensionale i numeri estratti al gioco del lotto su tutte le ruote per dieci estrazioni consecutive. Verificare su quali ruote e in quali estrazioni si ripete un certo numero passato in ingresso dall'utente.

Soluzioni

1.

Il ciclo che effettua la somma deve essere realizzato in modo che l'elemento del secondo array sia simmetrico rispetto al primo.

```
for(i=0; i<n; i++)
    c[i] = a[i] + b[n-i-1];
```

2.

```
max = voti[0];
min = voti[0];
media = voti[0];
for(i = 0; i <= 5; i++) {
    if(voti[i]>max)
        max = voti[i];
    else
        if(voti[i]<min)
            min = voti[i];
    media = media+voti[i];
}
```

13.

Devono essere definite le dimensioni della matrice.

```
#define N 10
#define P 10
#define M 10

int mat1[N][P];
int mat2[P][M];
int pmat[N][M];
```

Si devono richiedere all'utente le reali dimensioni e si deve controllare che il loro valore non superi le dimensioni delle matrici. I valori da richiedere sono soltanto tre in quanto le colonne della prima matrice devono essere in numero uguale alle righe della seconda.

```
/* Richiesta delle dimensioni */
do {
    printf("Numero di linee I matrice: ");
    scanf("%d", &n);
}
while((n>=N) || (n<1));

do {
    printf("Numero colonne I matrice / righe II matrice: ");
    scanf("%d", &p);
}
while((p>=P) || (p<1));
```

```

do {
    printf("Numero di colonne II matrice: ");
    scanf("%d", &m);
}
while((m>=M) || (m<1));

```

Anteriormente devono essere state dichiarate le variabili `n`, `m` e `p`.

```
int n, m, p;
```

Sostituire `N`, `M` e `P` con `n`, `m` e `p` nel resto del programma.

16.

```

/* Calcolo media voti per studente e per prova
   Nell'esemplificazione utilizziamo 3 studenti e 4 prove */
#include <stdio.h>

```

```

#define n 4
#define m 5
float voti[n][m];

```

```
main()
```

```
{
int i, j;
```

```

printf("\n \n CARICAMENTO DEI VOTI \n \n");
for(i=0; i<n-1; i++)
    for(j=0; j<m-1; j++) {
        printf("Ins. studente %d prova %d: ", i+1, j+1);
        scanf("%f", &voti[i][j]);
    };

```

```

/* Calcolo medie per studente */
for(i=0; i<n-1; i++) {
    voti[i][m-1] = 0;
    for(j = 0; j < m-1; j++)
        voti[i][m-1] = voti[i][m-1] + voti[i][j];
    voti[i][m-1] = voti[i][m-1] / (m-1);
}

```

```

/* Calcolo medie per prova */
for(j=0; j<m; j++) {
    voti[n-1][j] = 0;
    for(i=0; i<n-1; i++)
        voti[n-1][j] = voti[n-1][j] + voti[i][j];
    voti[n-1][j] = voti[n-1][j]/(n-1);
}

```

```

printf("\n \n VISUALIZZAZIONE DELLA MATRICE \n ");
for(i=0; i<n; i++) {
    printf("\n");
    for(j=0; j<m; j++)
        printf("%8.3f", voti[i][j]);
}
putchar('\n'); putchar('\n');
}

```

Esempio di esecuzione

Ins. studente 1 prova 1: 4
Ins. studente 1 prova 2: 5
Ins. studente 1 prova 3: 4
Ins. studente 1 prova 4: 7
Ins. studente 2 prova 1: 8
Ins. studente 2 prova 2: 10
Ins. studente 2 prova 3: 8
Ins. studente 2 prova 4: 10
Ins. studente 3 prova 1: 6
Ins. studente 3 prova 2: 7
Ins. studente 3 prova 3: 8
Ins. studente 3 prova 4: 6

VISUALIZZAZIONE DELLA MATRICE

4.000	5.000	4.000	7.000	5.000
8.000	10.000	8.000	10.000	9.000
6.000	7.000	8.000	6.000	6.750
6.000	7.333	6.667	7.667	6.917