
Array in C

Emilio Di Giacomo

Limite delle variabili

- L'utilizzo di variabili “semplici” come quelle viste fino ad ora non è sufficiente per risolvere problemi in cui si debbano gestire collezioni di dati di dimensione non nota a priori
- Consideriamo ad esempio il seguente problema:
 - vogliamo far inserire all'utente una sequenza di interi e poi un numero intero positivo n
 - vogliamo visualizzare all'utente i soli interi inserite che sono divisibili per n

Limite delle variabili

- Il problema può avere due varianti:
 - la lunghezza della sequenza da fare inserire all'utente è fissata prima di scrivere il programma
 - la lunghezza della sequenza da fare inserire all'utente è scelta dall'utente stesso (e quindi non nota all'atto della scrittura del codice)
- Consideriamo inizialmente la prima variante e supponiamo che la lunghezza della sequenza sia quattro:
 - in questo caso potremmo risolvere il problema con gli strumenti che conosciamo

Soluzione 1

```
#include <stdio.h>

int main(){
    int a1, a2, a3, a4;
    printf("Inserisci quattro numeri interi\n");
    scanf("%d%d%d%d", &a1, &a2, &a3, &a4);

    int n;
    printf("Inserisci un intero\n");
    scanf("%d", &n);

    printf("Numeri divisibili per %d\n", n);
    if (a1%n==0)
        printf("%d ", a1);
    if (a2%n==0)
        printf("%d ", a2);
    if (a3%n==0)
        printf("%d ", a3);
    if (a4%n==0)
        printf("%d ", a4);
    printf("\n");
}
```

Limiti della soluzione vista

- La soluzione illustrata necessita di definire 4 variabili distinte per memorizzare le stringhe della sequenza
- Inoltre, la soluzione illustrata deve ripetere le stesse cose (lettura, verifica lunghezza e stampa) per ognuna delle stringhe inserite
- Cosa succede se voglio fissare la lunghezza della sequenza a 100, o addirittura a 100.000?
 - è molto dispendioso scrivere un codice in cui definisco esplicitamente 100.000 variabili ed in cui ripeto 100.000 istruzioni “quasi” uguali!!

Variante con lunghezza non fissata

- Nel caso in cui la lunghezza della sequenza non fosse fissata nel programma, ma decisa dall'utente, sarebbe addirittura impossibile scrivere il programma usando i soli strumenti che abbiamo
 - non sappiamo quante variabili bisogna usare
 - non sappiamo quante volte ripetere le operazioni di lettura, verifica della lunghezza e stampa

Gli array

- Per risolvere problemi come quello precedente, i linguaggi di programmazione forniscono una struttura dati nota come array
- Un array è una sequenza di variabili:
 - tutte le variabili di un array hanno lo stesso tipo di dato (il tipo dell'array), e si chiamano anche elementi dell'array
 - ogni variabile ha associato un indice (numero intero non negativo)
 - la dimensione (o lunghezza) dell'array è il numero delle sue variabili

Array in C

- In C per definire un array di un certo tipo *T* la sintassi è

T *<nome>* [*<dimensione>*]

- Esempio:
 - creazione di un array *a* di tipo *int* di dimensione *5*:
int a[5];
 - creazione di un array *b* di tipo *double* di dimensione *10*:
double b[10];

Accesso agli elementi di un array

- Un array di tipo T di dimensione n è costituito da n variabili di tipo T
- Ognuna di tali variabili può memorizzare un dato di tipo T e può essere acceduta (in lettura o scrittura) indipendentemente dalle altre
- L'accesso ad una certa variabile avviene specificando un indice
 - Ogni elemento di un array è infatti associato ad un indice compreso tra 0 e $n-1$

$\langle \text{nome array} \rangle [\langle \text{espressione} \rangle]$

Accesso agli elementi di un array

- Esempio:

int a[5];

a[0] = 2;

a[1] = -15;

a[2] = 78;

a[3] = 4;

a[4] = 0;

0	2
1	-15
2	78
3	4
4	0

- Il brano precedente crea un array di interi di dimensione 5 e assegna ad ogni elemento di tale array un valore

Riconsideriamo il nostro problema

- Utilizzando gli array possiamo risolvere il problema della sequenza di interi
- Creiamo un array di dimensione opportuna per memorizzare la sequenza di interi
- Sia la lettura dei valori che la valutazione della loro divisibilità per n (con eventuale stampa) avviene scandendo gli elementi dell'array

Soluzione 2

```
#include <stdio.h>
# define DIM 4

int main(){

    int a[DIM];
    printf("Inserisci %d numeri interi\n",DIM);
    for(int i=0;i<DIM;i++)
        scanf("%d",&a[i]);

    int n;
    printf("Inserisci un intero\n");
    scanf("%d",&n);

    printf("Numeri divisibili per %d\n", n);
    for(int i=0;i<DIM;i++)
        if (a[i]%n==0)
            printf("%d ",a[i]);
    printf("\n");
}
```

Soluzione 2: commenti

- La soluzione 2 è più compatta della soluzione 1
 - non c'è bisogno di ripetere più volte le stesse istruzioni
- Il codice inoltre è facilmente modificabile nel caso la lunghezza della sequenza cambi:
 - per passare da 4 a 100 stringhe è sufficiente cambiare il valore della costante *DIM*
- È inoltre possibile adattare facilmente il codice al caso in cui la lunghezza della sequenza viene scelta dall'utente

Soluzione 3

```
#include <stdio.h>

int main(){
    printf("Quanti interi vuoi inserire?\n");
    int dim;
    scanf("%d",&dim);
    int a[dim];
    printf("Inserisci %d numeri interi\n",dim);
    for(int i=0;i<dim;i++)
        scanf("%d",&a[i]);

    int n;
    printf("Inserisci un intero\n");
    scanf("%d",&n);

    printf("Numeri divisibili per %d\n", n);
    for(int i=0;i<dim;i++)
        if (a[i]%n==0)
            printf("%d ",a[i]);
    printf("\n");
}
```

Ancora sugli indici

- Abbiamo visto che gli elementi di un array di lunghezza n a sono associati agli indici da 0 ad $n-1$
- Che cosa succede se tentiamo di accedere ad una posizione non valida, cioè ad un indice minore di 0 o maggiore di $n-1$?
- In fase di compilazione non si ha nessun problema
 - l'espressione usata come indice non viene valutata ma ne viene soltanto controllato il tipo
- In fase di esecuzione l'istruzione viene eseguita e si accede ad una cella memoria che non fa parte dell'array
 - l'esito dell'operazione è imprevedibile

Come funzionano gli indici

- Che cosa succede?
- Quando scriviamo

int a[10];

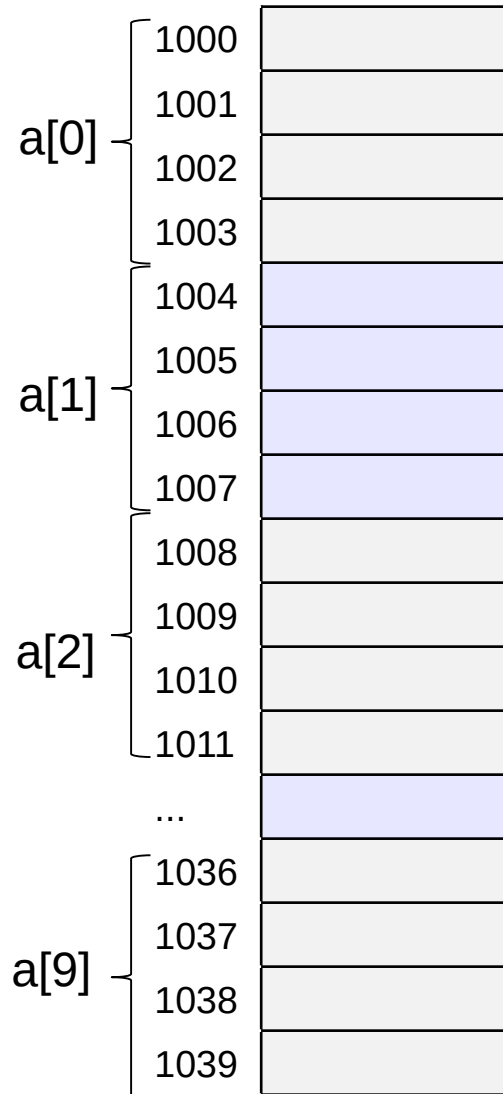
- viene riservata una quantità di memoria in celle consecutive sufficiente a contenere *10 int*
 - 40 byte se *sizeof(int)* è pari a 4
- nella variabile *a* viene memorizzato l'indirizzo di memoria della prima cella riservata per l'array

Come funzionano gli indici

int a[10];

a

1000



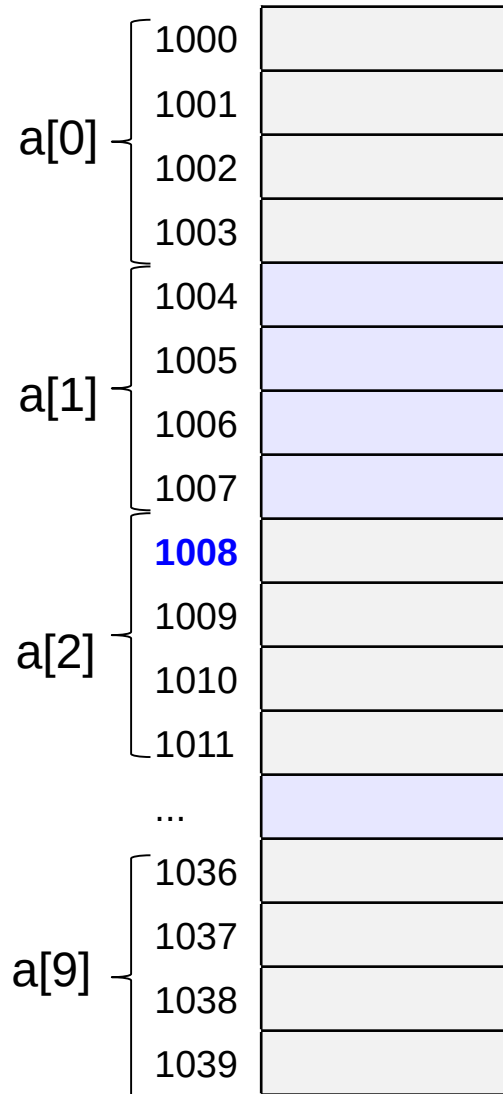
Come funzionano gli indici

- Quando accediamo ad un elemento dell'array si determina l'indirizzo cui accedere calcolando un offset rispetto all'indirizzo memorizzato in *a*
 - l'offset dipende dall'indice
- Esempio, se scriviamo
a[2]=7;
- l'offset è pari all'indice (*2*) per *sizeof(int)* (*4* nel nostro esempio), cioè *8*
- l'indirizzo cui si accede è l'indirizzo memorizzato in *a* più l'offset, cioè *1008*

Come funzionano gli indici

a

1000



Come funzionano gli indici

- Se usiamo un indice non valido si procede allo stesso modo:
 - si calcola l'offset e si somma all'indirizzo memorizzato in *a*
- Esempio, se scriviamo:
 $a[12]=7$ //indice non valido
- l'offset è *$12*4=48$* e si accede all'indirizzo *1048*
 - la cella *1048* non è una di quelle riservate per l'array
 - non sappiamo che cosa contenga
 - essa viene però scritta ugualmente con effetti imprevedibili

Inizializzazione di Array

- In C quando un array viene creato il contenuto dei suoi elementi non è noto
 - le celle di memoria riservate per l'array contengono dati che sono il risultato di precedenti elaborazioni
- È pertanto necessario inizializzare gli elementi di un array prima di utilizzarli

```
int a[10];
```

```
for(int i=0;i<10;i++)
```

```
    a[i]=0;
```

Inizializzazione di Array

- In C è anche possibile inizializzare gli elementi di un array contestualmente alla sua creazione
- Esempio

int a [5] = {2, 17, -4, -10, 35};

- *{2, 17, -4, -10, 35}* è detta lista di inizializzazione o letterale array
- L'inizializzazione con un letterale array deve avvenire contestualmente alla definizione dell'array

int[] a={1,2,3} //OK

int[] b;

b={4,5,6} // Errore!!

Inizializzazione di Array

- Diversi casi nell'uso dei letterali array
 - *int a[5] = {2, 17, -4, -10, 35};*
 - il numero di elementi nella lista è uguale alla dimensione
 - *int a[5] = {2, 17};*
 - il numero di elementi nella lista è minore della dimensione
 - gli elementi mancanti vengono inizializzati a 0
 - *int a[5] = {2, 17, -4, -10, 35, 56};*
 - il numero di elementi nella lista è maggiore della dimensione
 - errore in fase di compilazione

Inizializzazione di Array

- Diversi casi nell'uso dei letterali array
 - *int a[] = {2, 17, -4, -10, 35};*
 - la dimensione non viene indicata
 - essa viene posta pari al numero di elementi nella lista

Inizializzazione di Array

- Diversi casi nell'uso dei letterali array
 - *int a[] = {2, 17, -4, -10, 35};*
 - la dimensione non viene indicata
 - essa viene posta pari al numero di elementi nella lista

Copia di array

- Molto spesso nei programmi che usano array è necessario copiare un array in un altro
- Non è possibile copiare *a* in *b* tramite un assegnazione diretta:

b=a; // errore in fase di compilazione

- È necessario copiare elemento per elemento

Copia di array

- Esempio:

```
int a[5];  
... // popola l'array a  
int b[5];  
for(int i=0;i<10;i++)  
    b[i]=a[i];
```

0	2
1	-15
2	78
3	4
4	0

Passare gli array alle funzioni

- Immaginiamo di voler scrivere una funzione che stampi gli elementi di un array
- Tale funzione deve ricevere un array come parametro
- È possibile definire un parametro formale di tipo array
- Esempio:

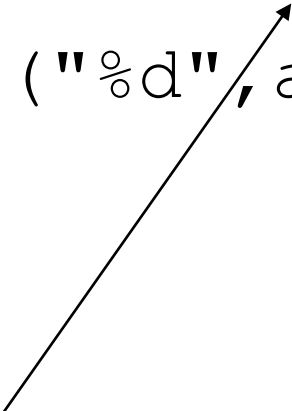
void stampaArray(int a[])

Passare gli array alle funzioni

- Se provassimo a scrivere la funzione *stampaArray* ci troveremmo di fronte ad un problema:

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

Quanto è lungo l'array?



Passare gli array alle funzioni

- Per questo motivo quando una funzione riceve un parametro di tipo array di solito riceve un secondo parametro che ne indica la dimensione

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

Passare gli array alle funzioni

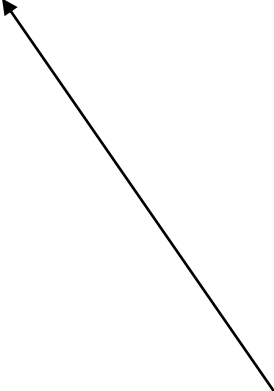
- Nota: nella definizione del parametro formale di tipo array si può anche indicare una dimensione, essa però viene ignorata

```
void stampaArray(int a[5], int size) {  
    for(int i=0; i<size; i++)  
        printf("%d", a[i]);  
}
```

Passare gli array alle funzioni

- L'invocazione della funzione stampaArray avviene nel seguente modo:

```
int a[10]={1,2,3,4,5,6,7,8,9,0};  
stampaArray(a,10);
```



Nota: nell'invocazione
l'array viene passato
senza parentesi quadre

Ancora sul passaggio di array

- Supponiamo di definire la seguente funzione:

```
void mistero(int a[], int dim) {  
    for(int i=0; i<dim; i++)  
        a[i]=0;  
}
```

- e scriviamo:

```
int b[5]={3, 6, 7, 8, -2};  
mistero(b, 5);  
for(int i=0; i<5; i++)  
    printf("%d\n", b[i]);
```

- che cosa verrà stampato?

Ancora sul passaggio di array

- Si potrebbe pensare che l'array *b* passato come parametro alla funzione *mistero* non venga da essa modificato
 - abbiamo visto che il passaggio dei parametri avviene sempre per valore
- In realtà il codice precedente stampa tutti 0, cioè l'array *b* viene modificato dalla funzione *mistero*
 - di fatto abbiamo un *passaggio per riferimento*
- Perché?

Ancora sul passaggio di array

- Ricordiamo che b memorizza semplicemente un indirizzo di memoria
 - l'indirizzo della prima cella di memoria dell'array
- Pertanto passando b come parametro viene copiato il suo valore (cioè l'indirizzo) nel parametro formale a
- Quando accediamo all'elemento $a[i]$ si calcola l'indirizzo cui accedere calcolando l'offset rispetto al valore memorizzato in a che è lo stesso memorizzato in b

Ancora sul passaggio di array

- Se si vuole impedire la modifica di un array passato per parametro ad una funzione si può usare la parola chiave *const*

- Esempio:

```
void prova(const int a[], int dim) {...}
```

- Qualunque tentativo di modificare l'array *a* all'interno della funzione *prova* darà un errore in fase di compilazione

Esempi sull'uso di array

- Scrivere le seguenti funzioni.
 - *int massimo(int a[], int size)*: riceve come parametri un array di interi e la sua dimensione e restituisce l'elemento di valore massimo nell'array
 - *int contiene(int a[], int size, int k)*: riceve come parametri un array di interi, la sua dimensione e un valore *k* e restituisce 1 (vero) se *k* è contenuto nell'array, 0 (falso) in caso contrario
 - *int uguali(int a[], int sizea, int b[], int sizeb)*: riceve come parametri due array di interi con le rispettive dimensioni e restituisce 1 (vero) se i due array sono uguali, 0 (falso) in caso contrario

Esempi sull'uso di array

- Scrivere poi un programma che
 - fa inserire all'utente un array di interi a
 - dice all'utente qual è l'elemento massimo in a
 - fa inserire all'utente un intero k
 - dice all'utente se l'intero k è presente in a
 - fa inserire all'utente un secondo array b
 - dice all'utente se a e b sono uguali

Funzione massimo(...)

```
int massimo(int a[], int size) {  
    int massimo=a[0];  
    for (int i = 1; i<size; i++)  
        if (a[i]>massimo)  
            massimo=a[i];  
    return massimo;  
}
```

Funzione contiene(...)

```
int contiene(int a[], int size, int k) {  
    int presente=0;  
    int i=0;  
    while(i<size && !presente) {  
        if(a[i]==k)  
            presente=1;  
        i++;  
    }  
    return presente;  
}
```


Funzione uguali(...)

```
int uguali(int a[], int sizea, int b[], int sizeb){
    int uguali;
    if(sizea!=sizeb)
        uguali=0;
    else{
        uguali=1;
        int i=0;
        while(i<sizea && uguali){
            if(a[i]!=b[i])
                uguali=0;
            i++;
        }
    }
    return uguali;
}
```

Funzione uguali(...)

```
int main() {
    int sizea;
    printf("Quanti elementi nel primo array?\n");
    scanf("%d", &sizea);

    int a[sizea];

    for(int i=0; i<sizea; i++) {
        printf("Inserisci l'elemento in posizione %d\n", i);
        scanf("%d", &a[i]);
    }

    int max=massimo(a, sizea);
    printf("Il massimo elemento dell'array è: %d\n", max);
    ...
}
```

Funzione uguali(...)

```
int main() {
    ...
    int k;
    printf("Inserisci un elemento da cercare nell'array\n");
    scanf("%d", &k);

    if(contiene(a, sizea, k))
        printf("Il numero %d è presente nell'array\n", k);
    else
        printf("Il numero %d non è presente
nell'array\n", k);

    int sizeb;
    printf("Quanti elementi nel secondo array?\n");
    scanf("%d", &sizeb);
    ...
}
```

Funzione uguali(...)

```
int main() {  
    ...  
    int b[sizeb];  
  
    for(int i=0;i<sizeb;i++){  
        printf("Inserisci l'elemento in posizione %d\n",i);  
        scanf("%d",&b[i]);  
    }  
  
    if(uguali(a,sizea,b,sizeb))  
        printf("I due array sono uguali\n");  
    else  
        printf("I due array non sono uguali\n");  
}
```

Array multidimensionali

- Gli array che abbiamo visto finora sono caratterizzati dall'utilizzo di un solo indice per far riferimento agli elementi dell'array
- Questo tipo di array si chiamano anche [array unidimensionali](#)
- È possibile definire array i cui elementi sono riferiti per mezzo di più indici
- Si parla in questo caso di [array multidimensionali](#)

Array bidimensionali

- In particolare un array i cui elementi sono accessibili tramite due indici sono detti array bidimensionali
- Gli array bidimensionali vengono usati prevalentemente per modellare matrici
- La sintassi per la creazione di un array bidimensionale di tipo *T* è la seguente

T <nome> [<num. righe>][<num. colonne>]

- Esempio

int m[2][3];

	0	1	2
0			
1			

Accesso agli elementi

- Per accedere agli elementi di un array bidimensionale sono necessari due indici detti rispettivamente indice di riga e indice di colonna
- Ad esempio con $m[i][j]$ si denota l'elemento di riga i e colonna j dell'array bidimensionale m

Esempio

```
int m[2][3];  
a[0][0] = 10;  
a[0][1] = 71;  
a[0][2] = -3;  
a[1][0] = 1;  
a[1][1] = -7;  
a[1][2] = 21;
```

	0	1	2
0	10	71	-3
1	1	-7	21

Letterali array bidimensionale

- Anche gli array bidimensionali possono essere inizializzati attraverso letterali array

- Esempio:

```
int a [3][2]={10, 3}, {2, 71}, {-3, 1};
```

- Gli elementi all'interno della lista sono a loro volta letterali array, uno per riga
- È anche possibile scrivere:

```
int a [3][2]={10, 3, 2, 71, -3, 1};
```

Letterali array bidimensionale

- Come nel caso unidimensionale se gli elementi nella lista sono meno del necessario quelli mancanti vengono impostati a 0

- Esempi:

int a [3][2]={10, 2, 71, -3, 1};

	0	1
0	10	0
1	2	71
2	-3	1

int a [3][2]={10, 2, 71, -3, 1};

	0	1
0	10	2
1	71	-3
2	1	0

Passare gli array alle funzioni

- Immaginiamo di voler scrivere una funzione che stampi gli elementi di un array bidimensionali
- Tale funzione deve ricevere come parametro un array bidimensionale

Passare gli array alle funzioni

- Si potrebbe pensare che il parametro possa essere specificato così

void stampaArray(int a[][], int rows, int cols)

- In realtà è necessario indicare il numero di colonne

void stampaArray(int a[][5], int rows, int cols)

- chiaramente questo codice funzionerà soltanto con matrici con 5 colonne
- In generale con un array multidimensionale bisogna specificare tutti le dimensioni tranne al più la prima

Passare gli array alle funzioni

- È possibile specificare il numero di righe e colonne tramite un espressione valutata a tempo di esecuzione nel seguente modo

void stampaArray(int rows, int cols, int a[][cols])

- questo codice funzionerà con qualunque numero di colonne
- nota: il parametro che indica il numero delle colonne deve venire prima del parametro array

Visitare gli elem. di un array bidim.

- Gli elementi di un array monodimensionale vengono tipicamente scanditi uno dopo l'altro tramite un ciclo
- Come possiamo scandire gli elementi di un array bidimensionale?
- Tipicamente essi vengono scandite per righe (dalla prima all'ultima) e all'interno di ogni riga per colonne (dalla prima all'ultima)
 - ciò può essere fatto con due cicli annidati

Esempio

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

Es. di uso di array bidimensionali

- Vediamo ora un esempio d'uso di array bidimensionali
- Realizziamo un programma che:
 - fa inserire all'utente una matrice di numeri interi; l'utente deve scegliere sia le dimensioni (righe colonne) che i singoli elementi della matrice;
 - per ogni riga della matrice inserita, visualizza all'utente la somma degli elementi di tale riga.

Es. di uso di array bidimensionali

- Se la matrice inserita dall'utente fosse:

$$\begin{bmatrix} 4 & 2 & 11 & -2 \\ 10 & 5 & -3 & 0 \\ -1 & -3 & 6 & 1 \end{bmatrix}$$

- il programma dovrebbe stampare:

Somma elementi di riga 0 = 15

Somma elementi di riga 1 = 12

Somma elementi di riga 2 = 3

Codice

```
#include<stdio.h>

int main(){

    // definisce una matrice di dimensioni specificate
    int m, n;
    printf("Inserisci le dimensioni della matrice");
    scanf("%d%d",&n,&m);
    int matrice[m][n];

    // fa inserire all'utente gli elementi della matrice
    for (int i = 0; i<m; i++)
        for (int j = 0; n; j++){
            printf("Elemento (%d,%d)?",i,j);
            scanf("%d",&matrice[i][j]);
        }
    ...
}
```

Codice

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

```
int main(){
```

```
...
```

```
// visualizza le somme degli elementi di ogni riga
```

```
int somma;
```

```
for (int i = 0; i<m; i++){
```

```
    // calcola la somma sulla riga i
```

```
    somma = 0;
```

```
    for (int j = 0; n; j++)
```

```
        somma += matrice[i][j];
```

```
    // visualizza la somma sulla riga i
```

```
    printf("Somma riga %d = %d", i, somma);
```

```
}
```

```
}
```