Gli array

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna
http://www.moreno.marzolla.name/

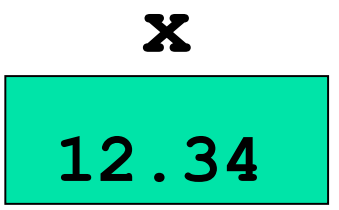
Ringraziamenti

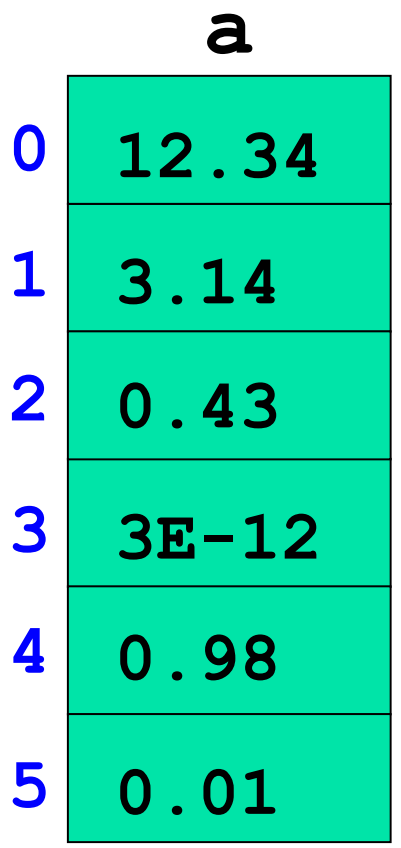
- prof. Stefano Mizzaro, Università di Udine
  - http://users.dimi.uniud.it/~stefano.mizzaro/

Gli array

- Scalare
  - Un “nome” a cui è associato un singolo valore di un determinato tipo
- Array
  - Un “nome” a cui sono associati più valori dello stesso tipo
    - Ogni valore ha una sua posizione all'interno dell'array
    - Un array di *N* elementi ha posizioni etichettate come 0, 1, … *N* - 1
    - Il primo elemento dell'array si trova in posizione 0.

x è una variabile scalare di tipo double il cui valore è 12.34

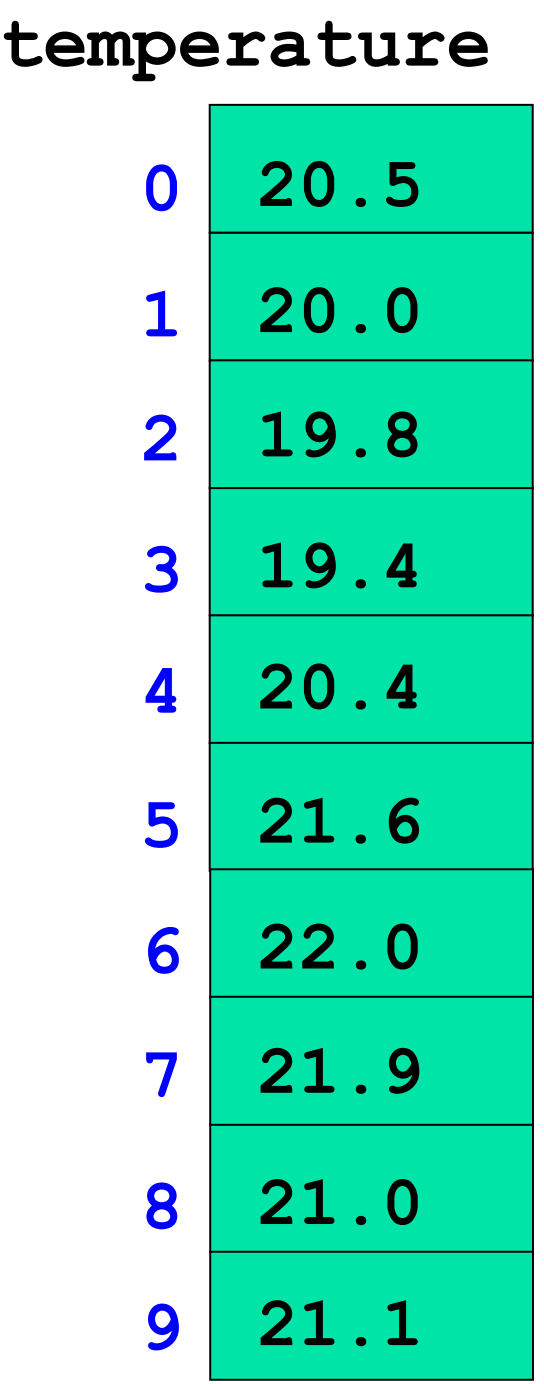




a è un array di 6 elementi di tipo double, le cui posizioni sono etichettate come 0, 1, … 5

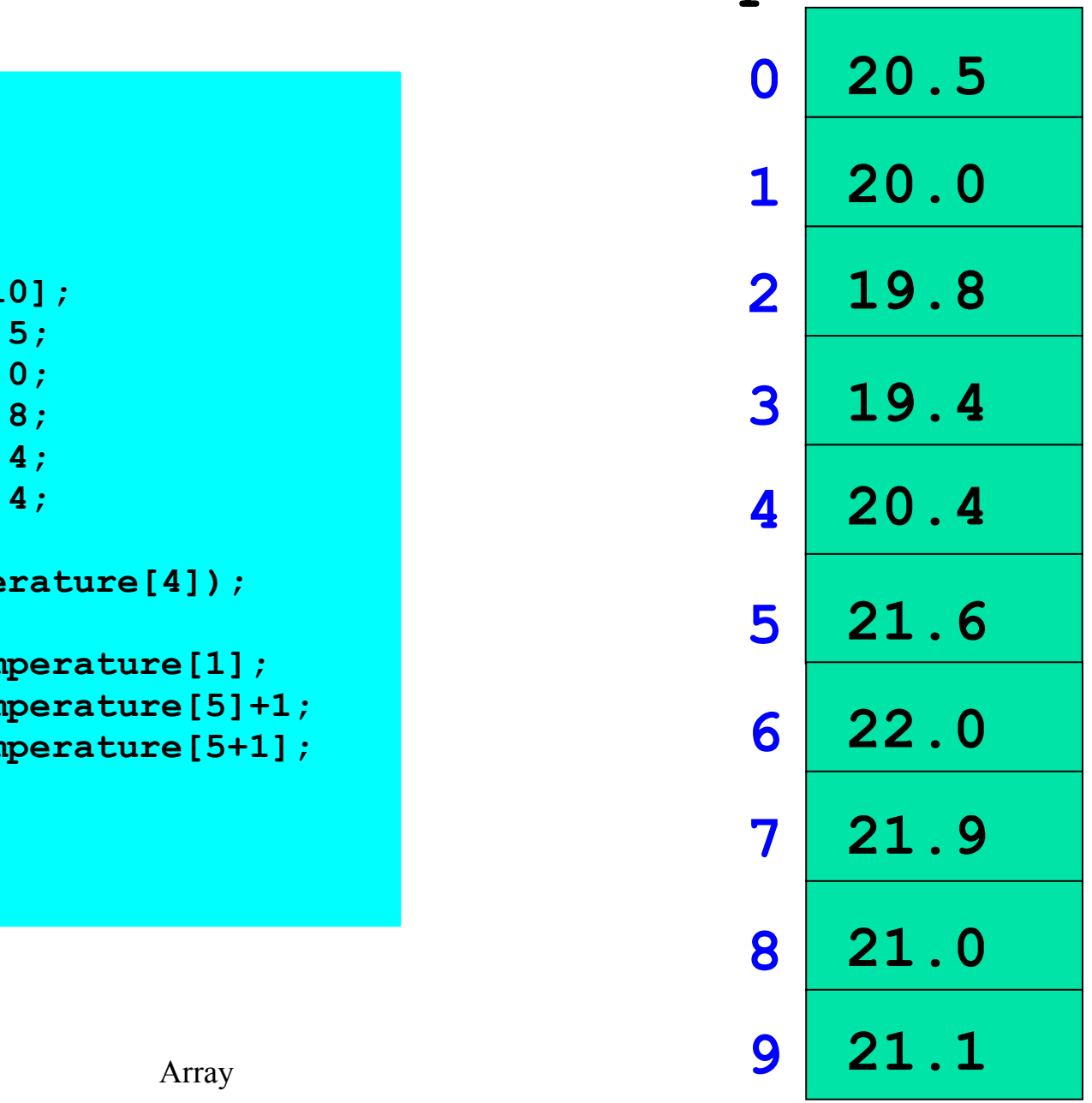
Esempio

- Programma che deve memorizzare 10 valori di temperatura
  - Uso 10 variabili **`double temperatura1, temperatura2, ..., temperatura10`**??
  - No, uso un array **`temperature`** di 10 elementi di tipo **`double`**
- Posso accedere ad ogni singolo valore tramite la posizione



In C

```
#include <stdio.h>

int main( void )
{
    double temperature[10];
    temperature[0] = 20.5;
    temperature[1] = 20.0;
    temperature[2] = 19.8;
    temperature[3] = 19.4;
    temperature[4] = 20.4;
    /* … */
    printf("%f\n", temperature[4]);
    /* … */
    temperature[5] = temperature[1];
    temperature[6] = temperature[5]+1;
    temperature[7] = temperature[5+1];
    /* … */
    return 0;
}
```



Vantaggio degli array

- Gestione uniforme
  - Ad es., per azzerare tutte le temperature…

```
temperature[0] = 0.0;
temperature[1] = 0.0;
temperature[2] = 0.0;
...
temperature[9] = 0.0;
```

Anziché fare questo...

```
int i;
for (i=0; i<10; i++) {
  temperature[i] = 0.0;
}
```

...possiamo fare questo

Gestione uniforme

- Calcolare la media delle temperature

```
double temperature[10];
double somma = 0.0;
int i;
...
for (i = 0; i < 10; i++) {
  somma = somma + temperature[i];
}
printf("media=%f\n", somma/10);
```

- Fatelo con le 10 variabili…
- … e poi fatelo per 100…
  - NB: la dimensione dell'array compare più volte (il letterale 10...); vedremo più avanti come migliorare

Variable-length arrays

- Lo standard C99 consente di dichiarare array la cui dimensione è contenuta in una variabile
  - La variabile deve essere almeno dichiarata (quindi visibile) prima della dichiarazione dell'array

```
/* OK sia in C99 che in ANSI C: a e' un array di 20 int, b e' un array di 10 int; il contenuto iniziale degli array e' indefinito */
int a[20], b[10];

/* OK in C99 (non ammesso in ANSI C): c e' un array di N=20 elementi; il suo contenuto iniziale e' indefinito */
int N = 20;
int c[N];

/* OK in C99 (non ammesso in ANSI C): meglio, d e' un array di K=30 elementi; il suo contenuto iniziale e' indefinito */
const int K = 30;
int d[K];
```

Dichiarazione

- A differenza di altri linguaggi (es., Java), la lunghezza di un array in C non è associata all'array
  - Non c'è un modo generale per ricavare la lunghezza di un array sapendo il suo nome
- Per tale motivo può essere utile usare un simbolo per tenere traccia della dimensione dell'array

Esempio

- Avete scritto un programma per gestire un array di 10 elementi

```
#include <stdio.h>

int main( void )
{
    int v[10], i;
    for (i=0; i<10; i++) {
        v[i] = i*i;
    }
    /* … */
    for (i=0; i<10; i++) {
        printf("v[%d]=%d\n", i, v[i]);
    }
    return 0;
}
```

Esempio

- ...Poi, un bel giorno, qualcuno vi chiede di modificare il programma per funzionare con un array di 20 elementi

```
#include <stdio.h>

int main( void )
{
    int v[20], i;
    for (i=0; i<20; i++) {
        v[i] = i*i;
    }
    /* … */
    for (i=0; i<10; i++) {
        printf("v[%d]=%d\n", i, v[i]);
    }
    return 0;
}
```

...così dovete andare a caccia di tutti i punti nel codice in cui usate il vecchio valore, per sostituirlo con il nuovo (che succede se ve ne dimenticate qualcuna?)

Soluzione

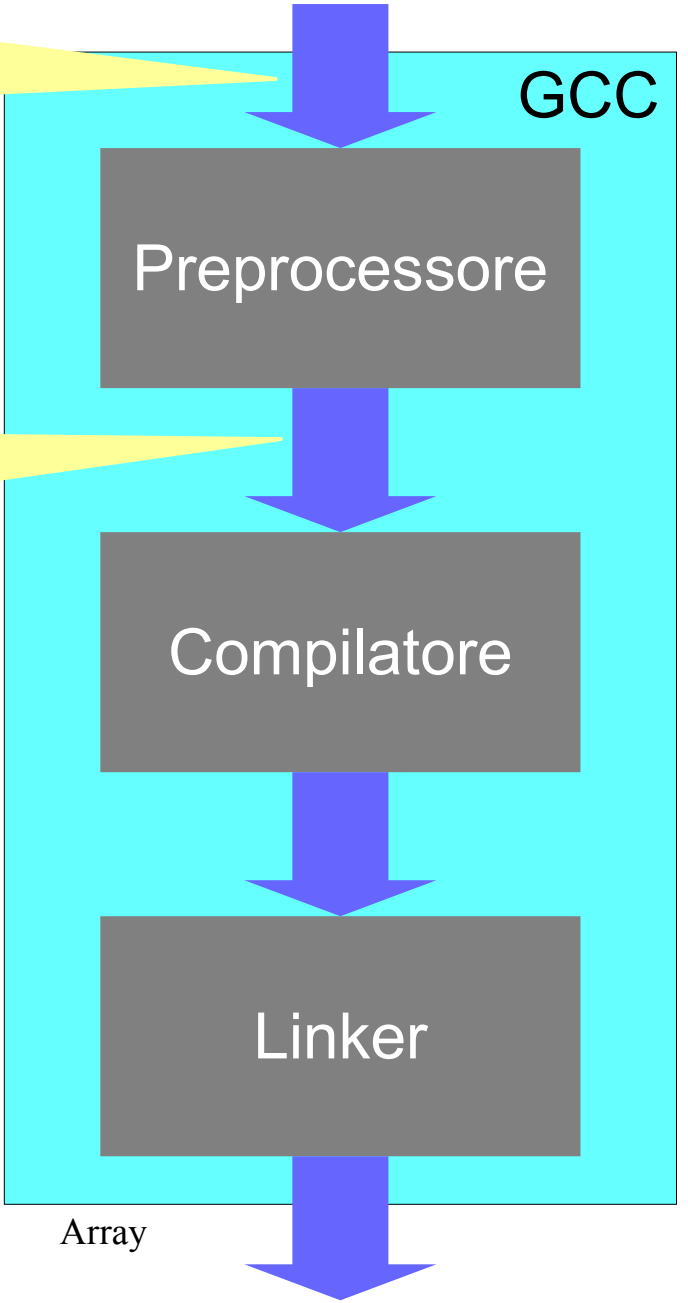
- Memorizzo la lunghezza di un array in un simbolo **N**
- **N** non è né una costante né una variabile
  - Il compilatore vero e proprio non vede **N**: il simbolo viene sostituito prima della compilazione con la sua definizione
- Questa soluzione funziona sia con ANSI C che con C99

```
#include <stdio.h>

#define N 10

int main( void )
{
    int v[N], i;
    for (i=0; i<N; i++) {
        v[i] = i*i;
    }
    /* … */
    for (i=0; i<N; i++) {
        printf("v[%d]=%d\n",
                    i, v[i]);
    }
    return 0;
}
```

Fasi di compilazione

```
#define N 10
int main( void )
{
        int a = N - 1;
        int v[N];
        return 0;
}
```

```
int main( void )
{
        int a = 10 - 1;
        int v[10];
        return 0;
}
```



Inizializzazione

```
/* OK: pero' il contenuto iniziale dell'array e' indefinito */
int giorniMese[12];
```

```
/* OK: calcola la lunghezza automaticamente */
int giorniMese[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

```
/* OK: lunghezza data esplicitamente */
int giorniMese[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

```
/* OK: pero' giorniMese[2], … giorniMese[11] sono inizializzati a 0 */
int giorniMese[12] = {31,28};
```

```
/* NON OK: warning: excess elements in array initializer */
int prova[3] = {10, 30, 20, 20};
```

Uso

- Per accedere all'elemento si usano le `[]` e l'indice (posizione) dell'elemento
  - L'indice deve essere una espressione di tipo **`int`**
- Ricordare che l'indice del primo elemento è 0 (zero)

```
/* Cosa fa questo frammento di codice? */
int a[] = {3, 1, 7, 2, 2};
int i;
for (i=1; i<5; i++) {
  a[i] = a[i-1] + a[i];
}
```

Gli esseri umani contano partendo da uno

I computer contano partendo da zero

Esempi

- Dato un array **`a[N]`** di **`int`**:
  - Azzerare **`a[]`** (assegnare zero a tutti gli elementi di **`a`**)

```
for (i = 0; i < N; i++) {
    a[i] = 0;
}
```

  - Assegnare 0, 1, 2, … agli elementi di **`a[]`**

```
for (i = 0; i < N; i++) {
    a[i] = i;
}
```

  - Incrementare di 1 ogni elemento di **`a[]`**

```
for (i = 0; i < N; i++) {
    a[i] = a[i] + 1;
}
```

Esempi

- Assegnare 1 agli elementi di **`a[]`** di indice dispari e 2 a quelli di indice pari

```
for (i = 0; i < N; i++) [
    if (i % 2 != 0) {
        a[i] = 1;
    } else {
        a[i] = 2;
    }
}
```

- ...oppure...

```
for (i = 0; i < N; i++) {
  a[i] = ( (i % 2 != 0) ? 1 : 2 );
}
```

Attenzione...

- Attenzione alle soluzioni troppo "astute"

```
for (i = 0; i < N; i = i+2) {
  a[i] = 2;
  a[i+1] = 1;
}
```

SBAGLIATO

```
for (i = 0; i < N-1; i = i+2) {
  a[i] = 2;
  a[i+1] = 1;
}
```

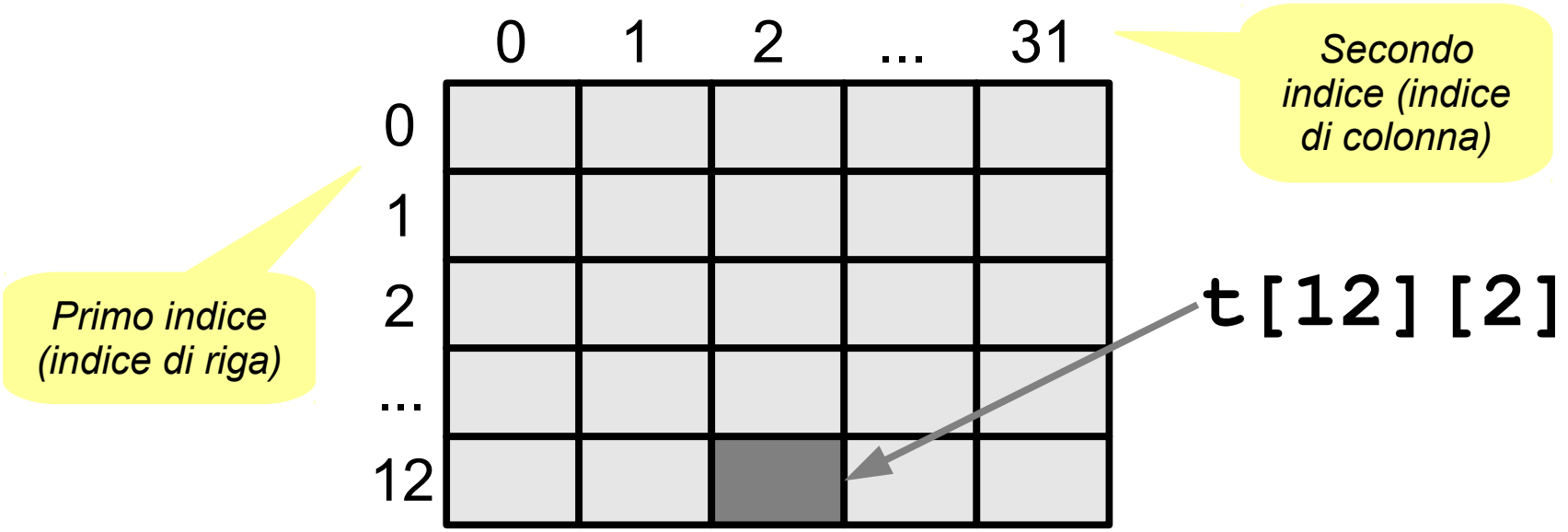
SBAGLIATO

```
for (i = 0; i < N; i = i+2) {
  a[i] = 2;
  if ( i+1<N ) { a[i+1] = 1; }
}
```

CORRETTO
ma contorto

Array bidimensionali (matrici)

- Doppie parentesi quadre
  - `double t[13][32]`
  - Il primo è il numero di righe, il secondo il numero di colonne
- Doppio indice
  - `t[12][2]`
  - Il primo è indice di riga, il secondo di colonna



Lavorare su matrici

- Data una matrice **`m[10][30]`** di **`int`**, assegnare 1 a tutti gli elementi

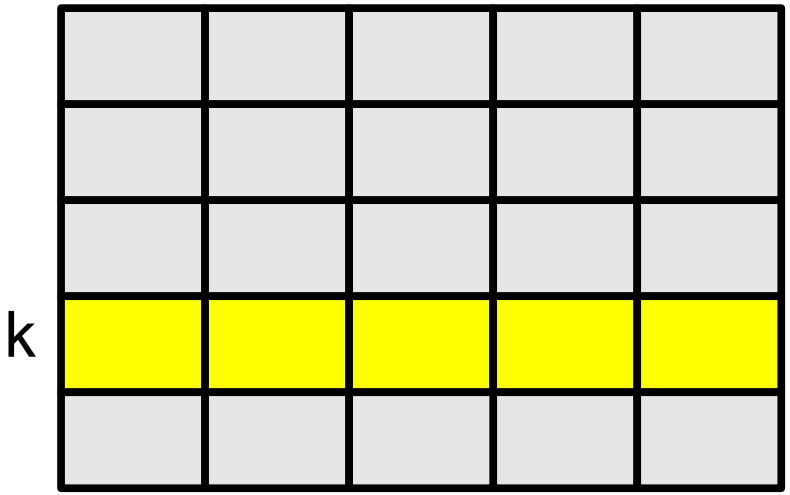
```
int m[10][30];
int i, j;

for (i = 0; i < 10; i++) {
    for (j = 0; j < 30; j++) {
        m[i][j] = 1;
    }
}
```

Array multidimensionali

- Array di array di array di …
- Si usa una coppia di `[]` per ogni dimensione
- Es.:
  - `double a[10][100][1000];`
  - `a[0][0][0] … a[9][99][999]`
- Utilità
  - Temperature di tutte le ore
  - `double t[13][32][24];`
    `... t[12][25][0] ...`

Esempi con matrici

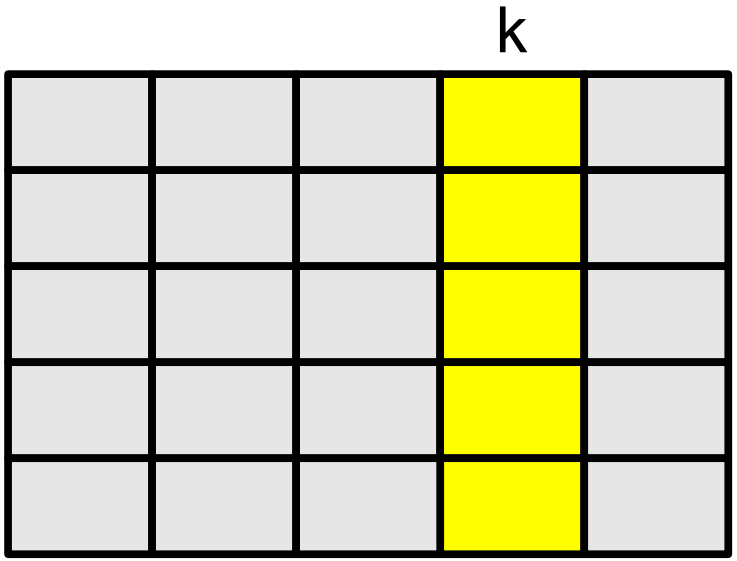
- Azzerare la riga *k*-esima di una matrice **`m[R][C]`**




```
#define R 20
#define C 30

int main( void )
{
    int m[R][C];
    int j;

    for (j = 0; j < C; j++) {
        m[k][j] = 0;
    }
    return 0;
}
```

Esempi con matrici

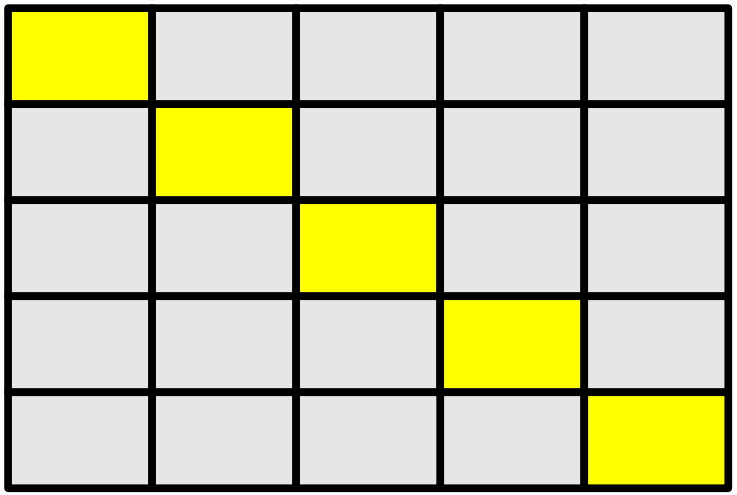
- Azzerare la colonna *k*-esima di una matrice **`m[R][C]`**




```
#define R 20
#define C 30

int main( void )
{
    int m[R][C];
    int i;

    for (i = 0; i < R; i++) {
        m[i][k] = 0;
    }
    return 0;
}
```

Esempi con matrici (3/3)

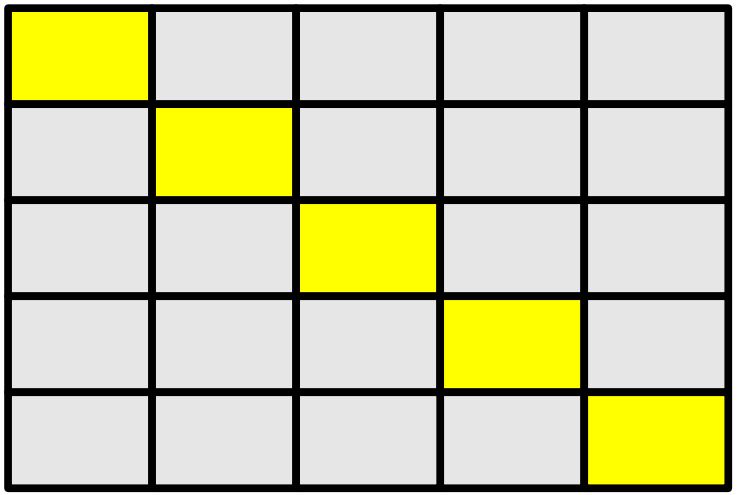
- Porre a 1 gli elementi della diagonale principale di una matrice **`m[N][N]`**, e 0 tutti gli altri



```
#define N 30

int main( void )
{
    int m[N][N];
    int i, j;

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            if (i==j) {
                m[i][j] = 1;
            } else {
                m[i][j] = 0;
            }
        }
    }
    return 0;
}
```

Esempi con matrici (3/3)

- Porre a 1 gli elementi della diagonale principale di una matrice **`m[N][N]`**, e 0 tutti gli altri



```
/* soluzione alternativa, usando l'operatore
ternario ? (forse meno chiara, ma comunque
corretta) */

#define N 30

int main( void )
{
    int m[N][N];
    int i, j;

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            m[i][j] = (i==j ? 1 : 0);
        }
    }
    return 0;
}
```

Funzioni e parametri di tipo array

- In C si possono passare parametri di tipo array alle funzioni senza specificarne la dimensione
- Se si modifica il contenuto di un parametro di tipo array viene modificato anche il parametro attuale!
  - Vedremo il motivo nelle successive lezioni
- In C non è consentito che una funzione restituisca un risultato di tipo array
  - Soluzione: si passa alla funzione un ulteriore parametro di tipo array, che viene “riempito” con il risultato

Esempio

```c
/* iniz-stampa-array.c: inizializza e stampa il contenuto di un array */
#include <stdio.h>
#define N 10

void inizializza(int a[], int n)
{
    int i;
    for (i=0; i<n; i++) {
        a[i] = i;
    }
}

void stampa(int a[], int n)
{
    int i;
    for (i=0; i<n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}

int main( void )
{
    int v[N];
    inizializza(v, N);
    stampa(v, N);
    return 0;
}
```

E i parametri di tipo matrice?

- Passare parametri di tipo matrice è un po' più complicato
- Nell'intestazione della funzione occorre tassativamente indicare il numero di colonne
  - Il numero di righe può restare non specificato
  - Se la funzione deve conoscere il numero di righe, è necessario passare tale informazione mediante un ulteriore parametro intero, o in qualche altro modo

```c
/* parametro-matrice.c: dimostrazione dell'uso di una matrice come parametro di una funzione */
#include <stdio.h>
#define R 3
#define C 4

void inizializza(int m[][C], int r)
{   int i, j, k = 0;
    for (i = 0; i < r; i++) {
        for (j = 0; j < C; j++) {
            m[i][j] = k++;
        }
    }
}

void stampa(int m[][C], int r)
{   int i, j;
    for (i = 0; i < r; i++) {
        for (j = 0; j < C; j++) {
            printf("%d ", m[i][j]);
        }
        printf("\n");
    }
}

int main( )
{
    int matrice[R][C];
    inizializza(matrice, R);
    stampa(matrice, R);
    return 0;
}
```

Quindi, riassumendo

- I parametri di tipo predefinito (**`int, float, double, char`**) vengono passati per valore
  - Le modifiche non si riflettono sul parametro attuale
- I parametri di tipo array e matrice sono passati "per riferimento"
  - Modifiche su parametro formale si riflettono sul parametro attuale