# Laboratorio di Programmazione 1



**Docenti:** Vincenzo Bonnici (A - L)
Maurizio Boscaini (M - Z)

**Lezione 8 – a.a. 2016/2017**

# Gli Array: Cosa Sono

- Un *array* è un tipo di dato che rappresenta un insieme ordinato di elementi dello *stesso tipo*.

- Ciascun elemento dell'insieme è identificato da un *indice* univoco.
  - `x[i]`

- In un array di *n* elementi, gli indici vanno da *0* a *n-1*.

# Gli Array: Dichiarazione ed Utilizzo

- La *dichiarazione* di una variabile di tipo array richiede di specificare:
  - Il tipo degli elementi contenuti nell'array (es. tutti i tipi primitivi).
  - Il nome dell'array.
  - Il numero massimo di elementi che saranno contenuti nell'array (tra parentesi quadrate).
    - `int voti[10];` // array di interi di dimensione massima 10

- Ciascun elemento dell'array può essere recuperato indicando l'indice corrispondente tra parentesi quadrate.
  - `Voti[3]; // quarto elemento dell'array voti.`

- Ogni elemento di un array può essere utilizzato come una qualsiasi variabile del corrispondente tipo primitivo.
  - Assegnare ad una variabile il valore di un elemento dell'array:
    - `int v = voti[3];`
  - Assegnamento di un valore ad un elemento dell'array.
    - `voti[3] = 26;`

# Gli Array: Gli Indici

- Gli indici usati per recuperare il valore di un elemento dell'array possono essere:
  - Costanti intere: `voti[3]`.
  - Espressioni con valori interi: `voti[(i+5)/2]`.
    - dove `i` è una variabile di tipo `int`.

- Il compilatore C non controlla se gli indici specificati sono validi (cioè appartengono ai limiti consentiti).
  - L'accesso ad un elemento con indice non valido può non generare un errore, ma semplicemente un comportamento inaspettato.

# Gli Array: Inizializzazione

- Un array può essere inizializzato direttamente durante la dichiarazione:
  - `int counters[5] = {1,2,3,4,5}`
    - `counters[0] == 1;`
    - `counters[1] == 2;`
    - `counters[2] == 3;`
    - `counters[3] == 4;`
    - `counters[4] == 5;`

- Non è necessario inizializzare tutti gli elementi: gli elementi non specificati saranno posti a zero:
  - `int counters[5] = {1,2}`
    - `counters[0] == 1;`
    - `counters[1] == 2;`
    - `counters[2], counters[3], counters[4] == 0;`

# Gli Array: Inizializzazione

- Se un array viene inizializzato nella dichiarazione non è necessario specificare il massimo numero di elementi che può contenere:
  - `char word[] = {'H', 'e', 'l', 'l', 'o'};`

- La dimensione viene determinata direttamente dal numero di elementi inizializzati.
  - L'array `word` avrà dimensione massima 5.

# Gli Array Multidimensionali

- Un array multidimensionale è un array i cui elementi hanno a loro volta tipo array.
- Gli array multidimensionali più utilizzati sono quelli bidimensionali: matrici.
- Dichiarazione di un array bidimensionale:
  - `int matrix[10][5];`
- Accesso ad un elemento:
  - `int value = matrix[5][3];`
  - `matrix[3][2] = 10;`

# Gli Array Multidimensionali

- Anche gli array multidimensionali possono essere inizializzati direttamente durante la dichiarazione.
  - Non è necessario specificare tutti gli elementi.

- Gli elementi vengono specificati per riga:

```
int matrix[3][4] =
  {{1,2,3,4},
   {5,6,7,8},
   {9,10,11,12}};
```

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

# Esercizio 1

Scrivere un programma C che richiede all'utente 10 valori e li memorizza all'interno di un array, quindi stampa tutti quelli maggiori o uguali dell'ultimo valore inserito.

# Esercizio 2

Scrivere un programma C che:

- Richiede all'utente di inserire i vettori a e b di lunghezza 10:
  - `scanf( "%i,%i,%i,...", &a[0],&a[1],&a[2],...)`
- Richiede all'utente di scegliere un'operazione da eseguire su tali vettori:
  - +: somma tra due vettori
    - Il risultato è un vettore in cui ciascun elemento è la somma dei corrispondenti elementi nei vettori di input.
  - -: differenza tra due vettori
    - Il risultato è un vettore in cui ciascun elemento è la differenza dei corrispondenti elementi nei vettori di input.
  - *: prodotto elemento per elemento
    - Il risultato è un vettore in cui ciascun elemento è il prodotto dei corrispondenti elementi nei vettori di input.
  - x: prodotto scalare
    - Il risultato è un numero ottenuto facendo la somma dei prodotti a[i]*b[i].
  - Esempi
    - (2, 3, 4) + (4, 5, 6) = (6, 8, 10)
    - (2, 3, 4) – (4, 5, 6) = (-2, -2, -2)
    - (2, 3, 4) * (4, 5, 6) = (8, 15, 24)
    - (2, 3, 4) x (4, 5, 6) = 2*4 + 3*5 + 4*6 = 47

# Esercizio 3

Scrivere un programma C che richiede all'utente 20 voti (compresi tra 18 e 30). Per ogni voto da 18 a 30 stampa il numero delle sue occorrenze e individua i voti che hanno più occorrenze.

*Esempio:* Se i voti immessi sono:

18 23 21 25 29 30 19 21 23 24

23 30 27 21 29 24 21 23 24 22

21 e 23 hanno 4 occorrenze e sono anche i voti col maggior numero di occorrenze. Il programma li deve individuare entrambi.

# Esercizio 4

Scrivere un programma C che salva in ogni elemento di un array bidimensionale il valore ($i$+$j$) dove $i$ è il numero di riga e $j$ il numero di colonna.

- Scegliete a piacere il numero di righe e colonne.
- Finché l'utente vuole continuare, il programma richiede all'utente i valori $i$ e $j$ e stampa il corrispondente elemento a video, oppure un messaggio di errore se gli indici non sono validi.

# Esercizio 5

Scrivere un programma C che:

- Richiede all'utente di inserire due matrici quadrate 3x3.
  - ```
    for( i = 0; i < 3; i++ ){
      scanf( "%i,%i,%i",
             &a[i][0], &a[i][1], &a[i][2] );
    }
    ```
- Esegue la moltiplicazione tra le due matrici.
  - Date due matrici quadrate a e b, l'elemento c[i][j] della matrice c risultato è dato dal prodotto scalare della riga i di a per la colonna j di b.
  - Per il prodotto scalare si veda l'esercizio 2.
- Stampa a video il risultato.
    ```

# Esercizio 5 - Esempio

Matrice a:

| | | |
|---|---|---|
| 3 | 2 | 5 |
| 1 | 0 | 4 |
| 4 | 1 | 3 |

Matrice b:

| | | |
|---|---|---|
| 1 | 0 | 3 |
| 4 | 7 | 2 |
| 5 | 1 | 3 |

Matrice c:

| | | |
|---|---|---|
| 36 | 19 | 28 |
| 21 | 4 | 15 |
| 23 | 10 | 23 |

c[ 1 ][ 1 ] = a[ 1 ][ 1 ]*b[ 1 ][ 1 ] + a[ 1 ][ 2 ]*b[ 2 ][ 1 ] + a[ 1 ][ 3 ]*b[ 3 ][ 1 ]
c[ 1 ][ 2 ] = a[ 1 ][ 1 ]*b[ 1 ][ 2 ] + a[ 1 ][ 2 ]*b[ 2 ][ 2 ] + a[ 1 ][ 3 ]*b[ 3 ][ 2 ]
c[ 1 ][ 3 ] = a[ 1 ][ 1 ]*b[ 1 ][ 3 ] + a[ 1 ][ 2 ]*b[ 2 ][ 3 ] + a[ 1 ][ 3 ]*b[ 3 ][ 3 ]
c[ 2 ][ 1 ] = a[ 2 ][ 1 ]*b[ 1 ][ 1 ] + a[ 2 ][ 2 ]*b[ 2 ][ 1 ] + a[ 2 ][ 3 ]*b[ 3 ][ 1 ]
c[ 2 ][ 2 ] = a[ 2 ][ 1 ]*b[ 1 ][ 2 ] + a[ 2 ][ 2 ]*b[ 2 ][ 2 ] + a[ 2 ][ 3 ]*b[ 3 ][ 2 ]
….
c[ i ] [ j ] = a[ i ][ 1 ]*b[ 1 ][ j ] + a[ i ][ 2 ]*b[ 2 ][ j ] + a[ i ][ 3 ]*b[ 3 ][ j ]