Ripasso di Vettori e Matrici in C

1. Vettori (Array Monodimensionali)

Un vettore è una collezione di variabili dello **stesso tipo**, memorizzate in locazioni contigue di memoria. Ogni elemento è accessibile tramite un **indice** intero, che parte sempre da 0.

Dichiarazione e Accesso

La sintassi è:

```c
int numeri[5]; // vettore di 5 interi
numeri[0] = 10; // primo elemento
numeri[4] = 50; // ultimo elemento (indice 4)
```

Inizializzazione

Si può inizializzare il vettore contestualmente alla dichiarazione:

```c
int v[5] = {1, 2, 3, 4, 5}; // tutti e 5 i valori
int w[] = {6, 7, 8, 9, 10}; // dimensione 5 dedotta automaticamente
```

Se si forniscono meno valori della dimensione, i rimanenti sono inizializzati a 0.

Scorrere un Vettore con un Ciclo

```c
#include <stdio.h>

int main() {
int v[5] = {10, 20, 30, 40, 50};
int somma = 0;

      for (int i = 0; i < 5; i++) {
      printf("Elemento %d: %d\n", i, v[i]);
            somma += v[i];
      }

      printf("Somma: %d\n", somma);
return 0;
}
```

2. Matrici (Array Bidimensionali)

Una matrice è un array di array: un insieme di righe e colonne. In C si dichiara specificando il numero di righe e di colonne.

Dichiarazione e Accesso

```c
int matrice[2][3]; // 2 righe, 3 colonne (6 elementi totali)
matrice[0][1] = 7; // elemento a riga 0, colonna 1
```

L'indice di riga e l'indice di colonna partono entrambi da 0.

Inizializzazione

Si possono usare parentesi graffe annidate:

```c
int m[2][3] = {
      {1, 2, 3}, // prima riga
      {4, 5, 6} // seconda riga
};
```

Oppure in forma lineare (il compilatore riempie per righe):

```
int m[2][3] = {1, 2, 3, 4, 5, 6};
```

È possibile omettere la prima dimensione (il numero di righe), ma **non** la seconda (le colonne):

```
int m[][3] = {1, 2, 3, 4, 5, 6}; // OK, il compilatore deduce 2 righe
```

Scorrere una Matrice con Cicli Annidati

```
#include <stdio.h>

int main() {
      int m[2][3] = {
            {1, 2, 3},
            {4, 5, 6}
      };
      int totale = 0;

      for (int r = 0; r < 2; r++) { // per ogni riga
            for (int c = 0; c < 3; c++) { // per ogni colonna
                  printf("%d ", m[r][c]);
                  totale += m[r][c];
            }
            printf("\n"); // a capo dopo ogni riga
      }

      printf("Somma totale: %d\n", totale);
      return 0;
}
```

Output:

```
1 2 3
4 5 6
Somma totale: 21
```

Operazioni tra Matrici

Esempio di somma di due matrici delle stesse dimensioni:

```
#define R 2
#define C 3

int main() {
      int A[R][C] = {
      {1, 2, 3},
      {4, 5, 6}
      };
      int B[R][C] = {
      {6, 5, 4},
      {3, 2, 1}
      };
      int somma[R][C];

      for (int i = 0; i < R; i++)
            for (int j = 0; j < C; j++)
                  somma[i][j] = A[i][j] + B[i][j];

// stampa matrice somma
      for (int i = 0; i < R; i++) {
            for (int j = 0; j < C; j++)
                  printf("%d ", somma[i][j]);
                  printf("\n");
            }
      return 0;
}
```

3. Passare Vettori e Matrici a Funzioni

In C, quando si passa un vettore a una funzione, viene passato l'indirizzo del primo elemento (il nome decade a puntatore). È quindi buona norma passare anche la dimensione come ulteriore parametro.

Funzione che Calcola la Media di un Vettore

```
float media(int arr[], int n) {
      int somma = 0;
      for (int i = 0; i < n; i++)
            somma += arr[i];
      return (float)somma / n;
}
```

Chiamata:

```
int v[] = {10, 20, 30};
printf("Media: %.2f", media(v, 3));
```

Funzione che Stampa una Matrice

Per le matrici è **obbligatorio** specificare la seconda dimensione (il numero di colonne) nel parametro, perché il compilatore deve sapere come calcolare gli spostamenti in memoria.

```
void stampaMatrice(int m[][3], int righe) { // colonne == 3 fissato
      for (int r = 0; r < righe; r++) {
            for (int c = 0; c < 3; c++)
                  printf("%d ", m[r][c]);
            printf("\n");
      }
}
```

Chiamata:

```
int mat[2][3] = {{1, 2, 3}, {4, 5, 6}};
stampaMatrice(mat, 2);
```

4. Note Importanti

- Gli indici dei vettori e delle matrici partono **sempre da 0**.
- Il C **non controlla** se l'indice è fuori dai limiti: è responsabilità del programmatore evitare buffer overflow.
- La dimensione di un vettore deve essere un'espressione costante a tempo di compilazione
- Le matrici sono memorizzate per righe: gli elementi della prima riga occupano locazioni consecutive, seguiti da quelli della seconda riga, e così via.
- Quando si passa un vettore a una funzione, si passa di fatto un puntatore al primo elemento; eventuali modifiche nella funzione influenzano l'array originale.

Con questi strumenti puoi gestire facilmente collezioni di dati omogenei in C.

Per un uso più avanzato, dovrai abbinarli ai puntatori e all'allocazione dinamica.