Array multidimensionali

Programmazione – Canale M-Z

LT in Informatica
16 Gennaio 2017

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Avviso

- Il primo compitino del corso di programmazione avrà luogo Giovedì 26 Gennaio a partire dalle 9:30, nei laboratori di Informatica del Paolotti
- La lista di iscrizione è aperta su uniweb. Appare come una prova parziale. Si chiude 2 giorni prima del 26/1.
- Su uniweb è aperta anche un'altra lista per un appello che fa riferimento al corso dell'anno scorso (2015-16).
- Gli studenti iscritti quest'anno devono iscriversi alla lista del 26/1 e non a quella del 19/1. Chi si fosse iscritto erroneamente alla lista del 19/1 è pregato di disiscriversi.

Array a più dimensioni

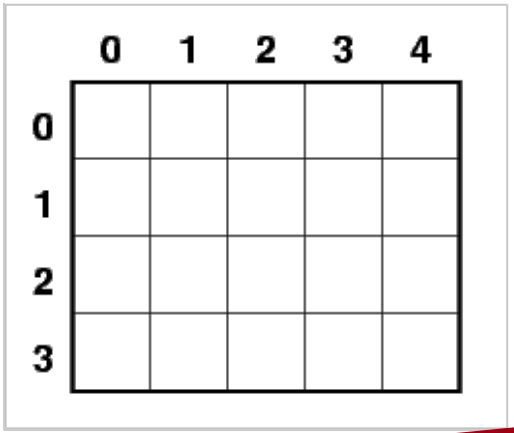
- Un array multidimensionale è un array i cui elementi sono indicizzati da tuple (n-uple) di interi invece di indici singoli.
- Il caso più semplice è quello in cui gli elementi sono indicizzati da coppie.
  - In questo caso si ottengono delle strutture chiamate matrici, di dimensione 2.
  - Le matrici sono il caso più comune di array multidimensionale che utilizzeremo.
- Possiamo anche avere array a 3, 4, 5, . . . , dimensioni:
  - è molto raro che servano più di 3 dimensioni

Dichiarare array multidimensionali

La dichiarazione è del tutto analoga a quella di un array monodimensionale, con la differenza che bisogna riportare il numero di elementi per ogni dimensione.

```
tipo nome [dim_1] [dim_2] .... [dim_n]
```

Esempio: `int matrix[4][5]` dichiara una matrice di interi di dimensioni 4×5 (quattro righe e cinque colonne):



Memoria necessaria

- Lo spazio necessario per memorizzare un array dipende dal numero totale di elementi
- Nel caso di dimensioni maggiori di 2 o 3, la memoria necessaria per un array multidimensionale può essere eccessiva!

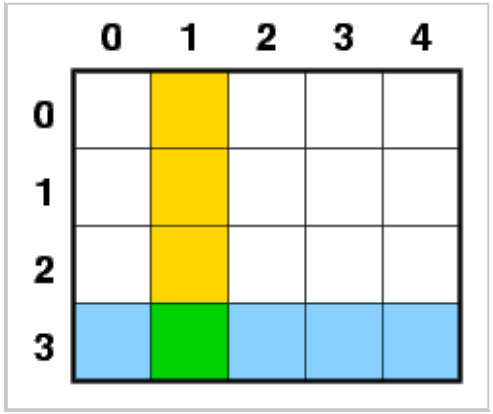
Esempio:

```
char secolo [100][365][24][60][60];
```

- richiede memoria per un valore `char` per ogni secondo contenuto in un secolo
- più di 3 miliardi di `char`!
- il che richiede più di 3 gigabyte di memoria RAM.

Accesso agli elementi

- Si può accedere ad un singolo elemento di un array bidimensionale usando doppi indici (per righe e colonne).
  - `matrix[3][1]` seleziona il secondo elemento della quarta riga:



  - Ricordiamoci che gli indici degli array iniziano sempre con 0!
- Nel caso di array di dimensioni maggiori si devono specificare tutti gli indici necessari:
  - a 3 dimensioni: `cartesio[x][y][z]`
  - a quattro dimensioni servono quattro indici
  - ...

Rappresentazione in memoria

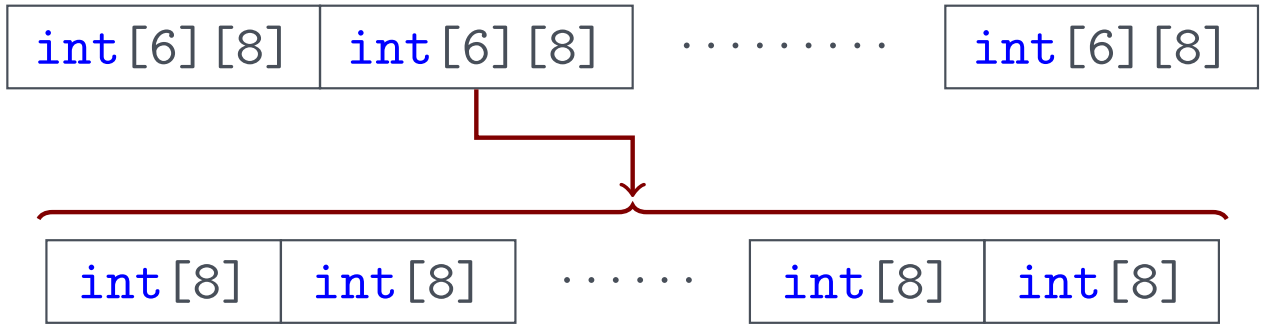
- Gli elementi di un array multidimensionale sono memorizzati nella RAM uno di seguito all'altro come per gli array semplici.
- Nel caso bidimensionale, la memorizzazione è per riga:
  - all'inizio tutti gli elementi della prima riga,
  - seguiti da tutta la seconda riga,
  - e così via fino all'ultima riga della matrice.
- Nel caso di dimensione $n > 2$, si ottiene una sequenza di array di dimensione $n - 1$.

Rappresentazione in memoria: esempi

1. `char A[5][10];` è rappresentato come 5 array `char[10]`:

   sono tutti appiccicati uno dopo l'altro

2. e `int B[5][6][8];` ? sono 5 array `int[6][8]`:



Operare su matrici: un esempio

- L'uso di cicli annidati consente di scrivere in maniera naturale gli algoritmi che operano su matrici
  - e più in generale, che operano su array multidimensionali

Esempio

Scrivere un programma che calcola la somma di tutti gli elementi di una matrice `4 x 5` e stampa il risultato.

- Il programma userà due cicli annidati
- Il ciclo più esterno scorre le righe
  - Invariante: `somma` contiene la somma dei valori nelle prime `i-1` righe
- Il ciclo più esterno scorre le colonne
  - Invariante: `somma` contiene la somma dei valori nelle prime `i-1` righe + la somma dei primi `j-1` valori della riga `i`

Esempio – codice della soluzione


```
#include <iostream>

using namespace std;

// PRE = cin contiene una matrice di interi 4 x 5 rappresentata per righe
int main() {
    int matrix[4][5];

    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 5; j++) {
            cin >> matrix[i][j];
        }
    }

    int somma = 0;
    for(int i = 0; i < 4; i++) {
        // R = somma contiene la somma dei valori nelle prime i-1 righe
        for(int j = 0; j < 5; j++) {
            // R = somma contiene la somma dei valori nelle prime i-1 righe
            //     + la somma dei primi j-1 valori della riga i
            somma = somma + matrix[i][j];
        }
    }

    cout << "La somma dei valori e': " << somma << endl;
}
// POST = somma contiene la somma dei valori della matrice
```