Corso “Programmazione 1”
Capitolo 06: Gli Array

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Giovanni De Toni - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: https://bit.ly/2VgfYwJ

Ultimo aggiornamento: 25 ottobre 2021

Terms of Use and Copyright

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Array multidimensionali

- In C++ è possibile dichiarare array i cui elementi siano a loro volta degli array, generando degli array multidimensionali (matrici)
- Sintassi:

```
tipo id[dim1][dim2]...[dimN];
tipo id[dim1][dim2]...[dimN]={lista_valori};
tipo id[][dim2]...[dimN]={lista_valori};
```

$\Longrightarrow$ un array multidimensionale $dim_1 \cdot ... \cdot dim_n$ può essere pensato come un array di dim_1 array multidimensionali $dim_2 \cdot ... \cdot dim_n$

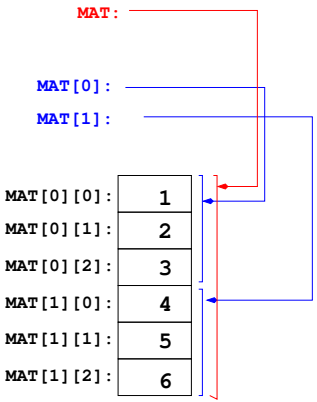
- Esempio:

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

- Le dimensioni devono essere valutabili durante la compilazione
- Elementi mancanti in inizializzazione (se presente) vengono sostituiti da zeri

Struttura di un array bidimensionale (statico)

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



Esempio matrice statica:

{ MATRICI/matrix_sta.cc }

Esempi: inizializzazione di array bidimensionali

- con inizializzazione:
  { `MATRICI/def_mat1.cc` }
- con inizializzazione parziale:
  { `MATRICI/def_mat2.cc` }
- con inizializzazione parziale (2):
  { `MATRICI/def_mat3.cc` }
- con inizializzazione, senza specifica prima dimensione:
  { `MATRICI/def_mat4.cc` }
- inizializzazione: errore:
  { `MATRICI/def_mat5.cc` }

Esempi: passaggio di array bidimensionali a funzioni

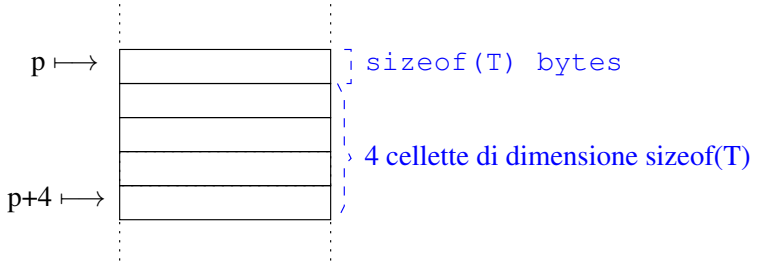
- solo una dimensione fissa, utilizzate in parte:
  { `MATRICI/matrix3.cc` }
- Err: (è necessario passare la dimensione):
  { `MATRICI/matrix3_err.cc` }

Aritmetica di Puntatori ed Indirizzi (richiamo)

Gli indirizzi e i puntatori hanno un'aritmetica:
se `p` è di tipo `*T` e `i` è un intero, allora:

- `p+i` è di tipo `*T` ed è l'indirizzo di un oggetto di tipo `T` che si trova in memoria dopo `i` posizioni di dimensione **`sizeof`**`(T)`
- analogo discorso vale per `p++`, `++p`, `p--`, `--p`, `p+=i`, ecc.

$\Longrightarrow$ `i` viene implicitamente moltiplicato per **`sizeof`**`(T)`

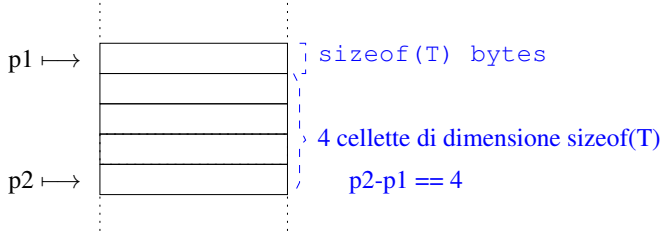


Aritmetica di Puntatori ed Indirizzi (richiamo) II

se `p1, p2` sono di tipo `*T`, allora:

- `p2-p1` è un intero ed è il numero di posizioni di dimensione **`sizeof`**`(T)` per cui `p1` precede `p2` (negativo se `p2` precede `p1`)
- si possono applicare operatori di confronto `p1<p2`, `p1>=p2`,ecc.

$\Longrightarrow$ `p2-p1` viene implicitamente diviso per **`sizeof`**`(T)`

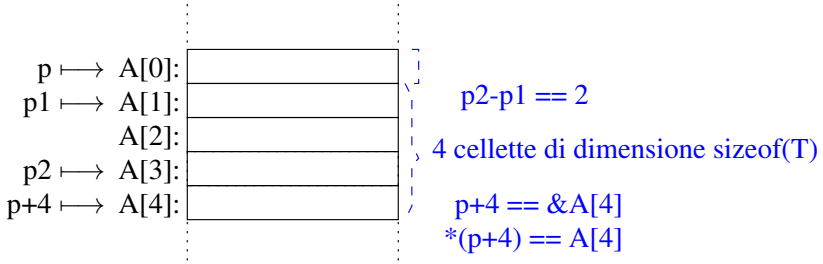


Esempio di operazioni aritmetiche su puntatori (richiamo):

{ `ARRAY_PUNT/aritmetica_punt.cc` }

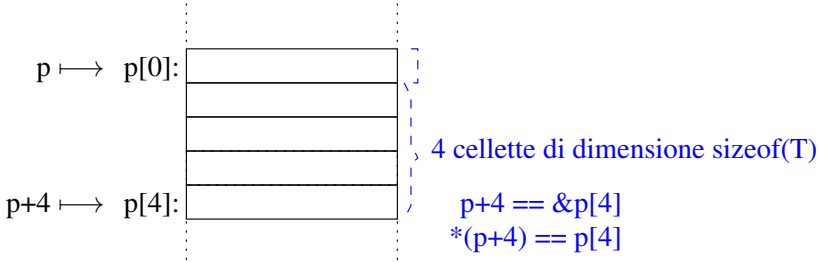
Puntatori ad elementi di un array

- Se un puntatore `p` punta al primo elemento di un array `A`, l'espressione `p+i` è l'indirizzo dell'i-esimo elemento dell'array `A`
  $\Longrightarrow$ valgono `p+i == &(A[i])` e `*(p+i) == A[i]`
- Se due puntatori dello stesso tipo `p1,p2` puntano ad elementi di uno stesso array, `p2-p1` denota il numero di elementi compresi tra `p1` e `p2` (`p2-p1` negativo se `p2` precede `p1`)



Array e puntatori

- Il nome di un array è (implicitamente equivalente a) una costante puntatore al primo elemento dell'array stesso
- se `p` è il nome di puntatore o di un array, le espressioni `&p[i]` e `p+i` sono equivalenti, come lo sono `p[i]` e `*(p+i)`



Esempi

- relazione tra array e puntatori:
  { `ARRAY_PUNT/punt_vett.cc` }
- idem, con doppio avanzamento:
  { `ARRAY_PUNT/punt_vett1.cc` }
- scansione array con puntatori:
  { `ARRAY_PUNT/scansione_array.cc` }
- ..., variante:
  { `ARRAY_PUNT/scansione_array2.cc` }

Passaggio di Array tramite Puntatore

- Nelle chiamate di funzioni viene passato solo l'indirizzo alla prima locazione dell'array,

$\Longrightarrow$ un parametro formale array può essere specificato usando indifferentemente la notazione degli array o dei puntatori

- Le seguenti scritture sono equivalenti (array semplici):

```
int f(int arr[dim]);
int f(int arr[]);
int f(const int *arr);
```

- Le seguenti scritture sono equivalenti (array multi-dimensionali):

```
int f(int arr[dim1][dim2]...[dimN]);
int f(int arr[][dim2]...[dimN]);
int f(const int *arr[dim2]...[dimN]);
```

Esempi

- array passati come tali (richiamo):
  { `ARRAY_PUNT/concatena_array.cc` }
- ... passati come puntatori:
  { `ARRAY_PUNT/concatena_array1.cc` }
- ... passati e manipolati come puntatori:
  { `ARRAY_PUNT/concatena_array2.cc` }
- passaggio con par. attuale puntatore, par. formale array:
  { `ARRAY_PUNT/concatena_array3.cc` }

Restituzione di un Array

Problema importante

Una funzione può restituire un array (allocato staticamente)?

- Sì, ma solo se è allocato staticamente esternamente alla funzione!
  (es. parametro formale, array globale)
- No, se è allocato staticamente internamente alla funzione!

- corretta, restituisce array parametro formale:
  { `ARRAY_PUNT/restituzione_array.cc` }
- corretta, restituisce array globale:
  { `ARRAY_PUNT/restituzione_array1.cc` }
- compila, ma fa disastri:
  { `ARRAY_PUNT/err_restituzione_array2.cc` }

Esercizi proposti

Vedere file `ESERCIZI_PROPOSTI.txt`