

Corso “Programmazione 1”

Capitolo 06: Gli Array

Docente: **Marco Roveri** - `marco.roveri@unitn.it`
Esercitori: **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: 13 ottobre 2021

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2021-2022.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

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.

1 Definizione ed Utilizzo di Array

2 Array e Funzioni

3 Array Ordinati

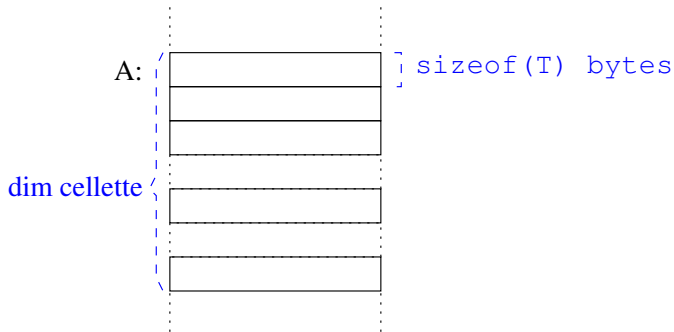
4 Array Multi-Dimensionali

5 Array e Puntatori

6 Array, Puntatori e Funzioni

Tipi e Variabili Array

- **Array**: sequenza finita di elementi consecutivi dello stesso tipo.
- Il numero di elementi di un array (**dimensione**) è fissata a priori
- Per un array di tipo `T` e dimensione `dim`, il compilatore alloca `dim` cellette consecutive di `sizeof(T)` bytes (**allocazione statica**)
- Un array rappresenta l'indirizzo del primo elemento della sequenza



Definizione ed Inizializzazione di Array

- Sintassi definizione:

- `tipo id[dim];`
- `tipo id[dim]={lista_valori};`
- `tipo id[]={lista_valori};`

- Esempi:

```
double a[25]; //array di 25 double
```

```
const int c=2;
```

```
char b[2*c]={'a','e','i','o'}; // dimensione 4
```

```
char d[]={ 'a','e','i','o','u'}; // dimensione 5
```

- La **dimensione** dell'array deve essere valutabile **al momento della compilazione**:

- **esplicitamente**, tramite l'espressione costante `dim`
- **implicitamente**, tramite la dimensione della lista di inizializzazione

- Se mancano elementi nella lista di inizializzazione, il corrispondente valore viene inizializzato allo **zero del tipo T**

Operazioni non lecite sugli array

- Sugli array **non** sono definite operazioni **aritmetiche**, **di assegnamento**, **di input**
- Le operazioni **di confronto** e **di output** sono definite, ma danno risultati imprevedibili

⇒ per tutte queste operazioni è necessario scrivere funzioni ad-hoc

```
int a[4] = {1,2,3,4};
int b[4] = {1,2,3,4};
// a++;           // ERRORE IN COMPILAZIONE
// a=b;           // ERRORE IN COMPILAZIONE
// cin >> a;      // ERRORE IN COMPILAZIONE
cout << (a==b) << endl; // COMPILA, MA DA' FALSE
cout << (a<=b) << endl; // COMPILA, MA IMPREVEDIBILE
cout << a << endl;      // COMPILA, MA IMPREVEDIBILE
```

esempio di cui sopra, espanso:

```
{ ARRAY/array.cc }
```

Operazioni sugli array: selezione con indice

L'unica operazione definita sugli array è la **selezione con indice** (**subscripting**), ottenuta nella forma:

`identifier[expression]`

- `identifier` è il nome dell'array
- il valore di `expression`, è l'**indice** dell'elemento
 - è di tipo discreto (convertibile in intero)

- **è un'espressione variabile!!**

Es: `v[100]` diversissimo da `v0, . . . , v99`, posso usare `v[i]`

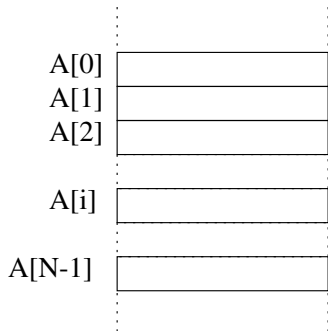
- `identifier[expression]` **è un'espressione dotata di indirizzo**
⇒ può ricevere un input, essere assegnata, passata per riferimento, ecc.

```
cin >> a[i];
```

```
a[n+3]=3*2;
```

```
scambia(a[i], a[i+1]);
```

Range degli array



- Gli elementi di un array di dimensione N sono numerati da 0 a $N-1$
- Esempio: se i vale 3, $a[i]=7$ assegna il quarto elemento di a a 7

Esempi: uso di array e range

- lettura di un valore, stampa in ordine inverso:
`{ ARRAY/array2_mia.cc }`
- ... con errore sul range (problema tipico):
`{ ARRAY/array2_errata.cc }`
- `g++ -fsanitize=bounds -fsanitize=bounds-strict` può aiutare ad identificare a **run-time** la maggior parte dei casi di errore di **array subscripting out of bounds**!

Esempi: inizializzazione di array e range

- **inizializzazione:**
{ ARRAY/array3.cc }
- **... (errore: il range di un array inizia da 0):**
{ ARRAY/array3_err.cc }
- **... senza esplicitazione della dimensione:**
{ ARRAY/array3_bis.cc }
- **inizializzazione a zero di default:**
{ ARRAY/array4.cc }
- **inizializzazione a tutti zero di default:**
{ ARRAY/array4_bis.cc }
- **vettore non inizializzato:**
{ ARRAY/array5.cc }

Attenzione: Uscita dal range di un array

In C++, l'operatore “[]” permette di “uscire” dal range $0 \dots \text{dim}-1$ di un'array!

```
int A[dim]; int i=0;
```

```
A[i-1];    // cella sizeof(int) bytes prima di A[0]
```

```
A[i+dim];  // cella sizeof(int) bytes dopo A[dim-1]
```

⇒ Effetti potenzialmente catastrofici in caso di errore

⇒ È responsabilità del programmatore garantire che un operazione di subscripting non esca mai dal range di un'array!

- Esempio:

errore, fuori range: catastrofico: va a sovrascrivere una variabile ma non rivelato!

(usare `-fno-stack-protector`):

```
{ ARRAY/array3_errato.cc }
```