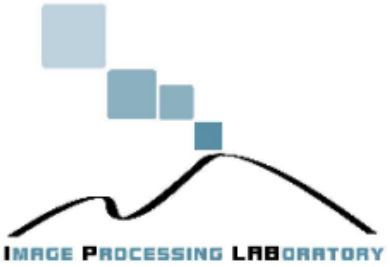
Gli Array in C++

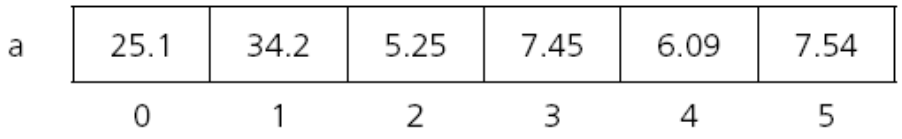
Alessandro Ortis
Image Processing Lab - iplab.dmi.unict.it

ortis@dmi.unict.it
www.dmi.unict.it/ortis/



Gli Array

- un *array* (o *vettore*) è una sequenza di locazioni di memoria riservata ad una serie di dati dello stesso tipo.
- gli oggetti si chiamano *elementi* dell'array e si numerano consecutivamente `0`, `1`, `2`, `3`.. ; questi numeri si dicono *indici* dell'array, ed il loro ruolo è quello di localizzare la posizione di ogni elemento dentro l'array, fornendo *acceso diretto* ad esso.
- il tipo di elementi immagazzinati nell'*array* può essere qualsiasi tipo di dato predefinito del C++, ma anche tipi di dato definiti dall'utente.
- se il nome del vettore è `a`, allora `a[0]` è il nome del primo elemento, `a[1]` è il nome del secondo elemento, ecc; l'elemento *i-esimo* si trova quindi nella posizione *i-1*, e se l'array ha *n* elementi, i loro nomi sono `a[0]`, `a[1]`, `...` , `a[n-1]`



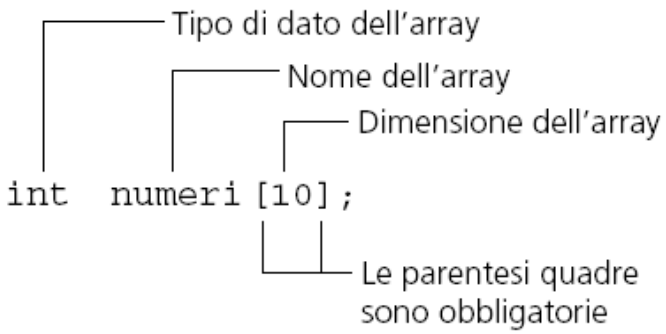
Definizione di array

un array si definisce specificando il tipo dei suoi elementi e, tra parentesi quadre, la sua dimensione (o *lunghezza*):

```
tipo_elementi nome_array[numero_elementi];
```

dove *numero_elementi* deve essere un valore intero; ad esempio, per creare un array (lista) di dieci variabili intere, si scrive:

```
int numeri[10]; //Crea un array di 10 elementi int
```



Accesso agli elementi di un array

- si può accedere ad un elemento dell'array mediante il suo nome ed un *indice* che ne rappresenta la posizione:

  nomeArray[n]

- C++ non verifica che gli indici dell'array stiano dentro la dimensione definita; così, ad esempio, se si accede a `numeri[12]` il compilatore non segnalerà alcun errore (*buffer overflow*)

```
int eta[5];          // contiene 5 elementi: il primo, eta[0] e l'ultimo, eta[4]
int pesi[25], lunghezze[100];      // definisce 2 array di interi
float salari[25];          // definisce un array di 25 elementi float
double temperature[50];        // definisce un array di 50 elementi double
char lettere[15];          // definisce un array di caratteri
eta[4]                          // referenzia il quinto elemento del vettore eta
vendite[totale + 5]        // accede all'elemento di indice pari al valore della
                              // variabile o costante totale aumentato di cinque
giorni[mese]     // accede all'elemento del vettore giorni dato dal valore della
                  // variabile o costante mese
salario[mese[i] * 5] // accede all'elemento del vettore salario il cui
          // indice è dato dal valore dell'i-esimo elemento del vettore mese
          //  moltiplicato per cinque
```

Allocamento in memoria di un array

- gli elementi degli array si immagazzinano in blocchi contigui;

```
int eta[5];
char codici[5];
```

- si può utilizzare l'operatore `sizeof` per conoscere il numero di bytes occupati dall'array; ad esempio, supponiamo che si definisca un array di `100` numeri interi denominato `eta`:

```
n = sizeof(eta);
```

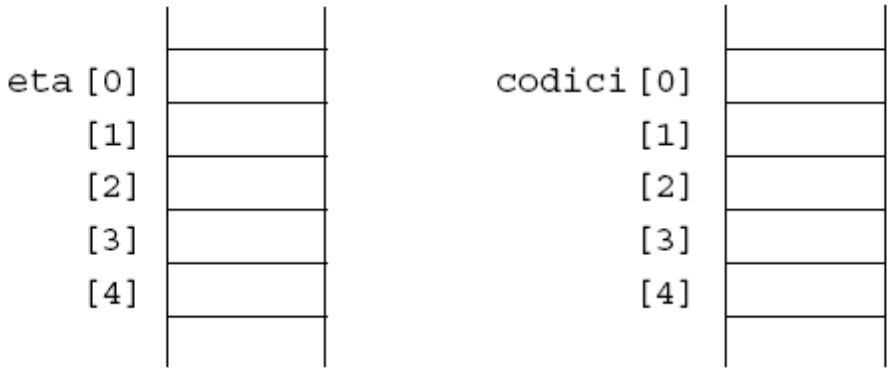
assegna `400` ad `n`; se si vuole conoscere la dimensione di un elemento individuale dell'array si può scrivere:

```
n = sizeof(eta[0]);
```

si può utilizzare l'operatore `sizeof` anche per conoscere il numero di bytes occupati da un tipo di dato: `n = sizeof(int);`

Verifica dell'indice di un array

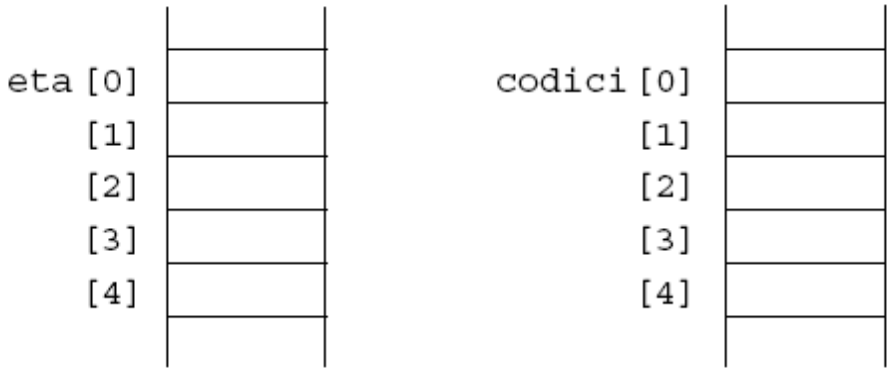
Come possiamo proteggere un codice (es. una funzione) da errori di buffer overflow?




```
double somma(double a[], int n){


      double s = 0.0;
      for (int i=0;i<n;i++)
            s+=a[i];
      retunrn s;
}
```

Verifica dell'indice di un array

Come possiamo proteggere un codice (es. una funzione) da errori di buffer overflow?



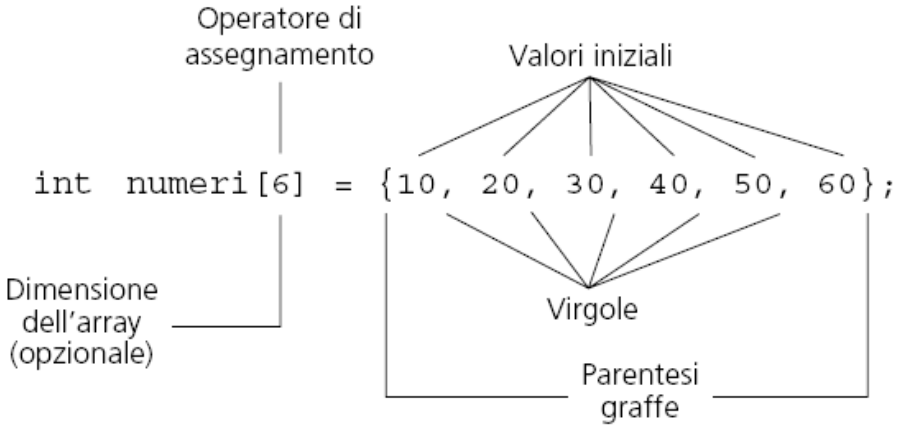

```
double somma(double a[], int n){

      if (n* sizeof(double)>sizeof(a)) return 0;
      double s = 0.0;
      for (int i=0;i<n;i++)
            s+=a[i];
      retunrn s;
}
```

Inizializzazione di un array

Per assegnare un valore ad un elemento di un array si può usare ovviamente l'operatore di assegnamento:

```
prezzi[0] = 10;
```

Possiamo inizializzare più elementi solo in fase di definizione.




```
int numeri[6] = {10, 20, 30, 40, 50, 60};
int numeri[6] = {10, 20, 30};

// in questo caso possiamo omettere '6', perché?
int numeri[] = {10, 20, 30, 40, 50, 60};
```

Inizializzazione di un array

Come posso inizializzare un array di 100 elementi tutti pari a zero ?

Inizializzazione di un array

Come posso inizializzare un array di 100 elementi tutti pari a zero ?

```
int numeri[100] = {0};
```

L'inizializzazione parziale può infatti essere sfruttata per inizializzare un array con elementi tutti pari a zero. Un altro modo per inizializzare gli elementi di un array è usare un ciclo.

```
int numeri[100];

for(int i=0;i<100;i++)
       numeri[i]=i*2;
```

Inizializzazione di un array

Quale sarà l'output di questo codice?

```
int n = 100;
int a[n] = {0};


n = 200;

cout << sizeof(a)/sizeof(int) << endl;
```

Inizializzazione di un array

Quale sarà l'output di questo codice?

```
int n = 100;
int a[n] = {0};


n = 200;

cout << sizeof(a)/sizeof(int) << endl;
```

>>> 100

La dimensione dell'array viene fissata all'atto della sua dichiarazione dal compilatore mediante il ***valore attuale*** della variabile *n*. Le eventuali successive variazioni del valore di *n* (o la sua rimozione dalla memoria) non hanno alcun effetto sull'array.

Array di caratteri e stringhe

Qual è l'output del seguente codice ?

```
char caratteri[] = {'A','l','d','o'};
char stringa1[] = ''Aldo'';

cout << sizeof(caratteri) << endl;
cout << sizeof(stringa1) << endl;

Output:
```

Array di caratteri e stringhe

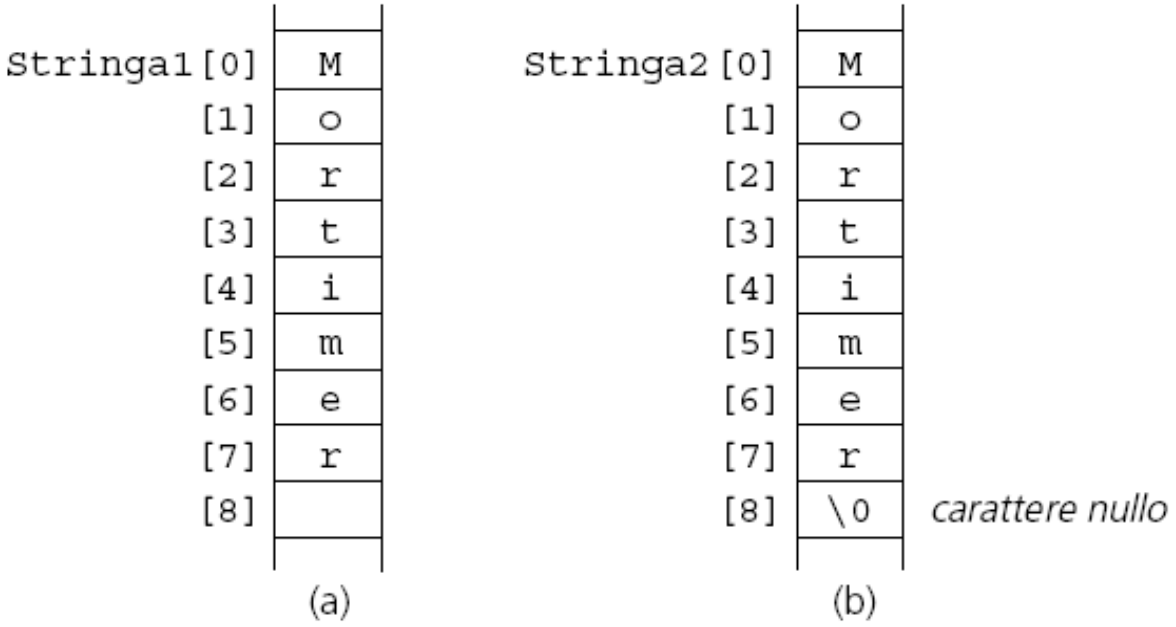
Qual è l'output del seguente codice ?

`caratteri[]` viene visto come array di char, mentre `stringa1[]` come stringa di caratteri, la quale è un un vettore di char con l'aggiunta di un carattere conclusivo in più, cioè il carattere ASCII NULL (otto bit pari a zero).

```
char caratteri[] = {'A','l','d','o'};
char stringa1[] = ''Aldo'';

cout << sizeof(caratteri) << endl;
cout << sizeof(stringa1) << endl;

Output:
4
5
```

Array di caratteri e stringhe

C++ rappresenta le stringhe di testo inserendole in array di caratteri che terminano con il carattere nullo '\0' (caso b in figura); senza di esso la stringa non è tale ma è un semplice array di caratteri (caso a)




```
char Stringa2[9] = "Mortimer"
```

Array di caratteri e stringhe: note

Non si può assegnare una stringa ad un array al di fuori della definizione:

```
stringa="Ciao";// Darebbe luogo ad errore
```

Si utilizza la funzione di libreria strcpy (che automatizza l'aggiunta del carattere \0 alla fine della stringa)

```
strcpy(nome, "Ciao");
```

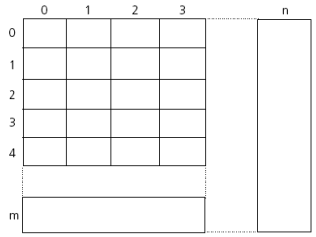
Il compilatore inserisce automaticamente il carattere nullo alla fine della stringa, tuttavia il carattere nullo non deve essere necessariamente l'ultimo elemento del vettore…

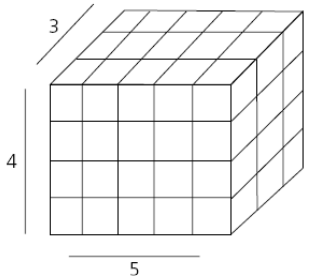
```
char str[] = "HelloWorld";
str[5] = '\0';
cout << str << endl;
```

Array multidimensionali

Gli array di array si dicono *bidimensionali*; hanno due dimensioni e, pertanto, due indici; sono noti anche con il nome di *tabelle* o *matrici*: la sintassi per la dichiarazione è:

tipo_elemento nome_array **[***NumRighe***][***NumColonne***]**





- gli array di array di array sono tridimensionali e così via
- gli array multidimensionali si inizializzano normalmente; es:

```
int tabella[2][3] = {{51, 52, 53}, {54, 55, 56}};
```

oppure:

```
int tabella[2][3] = {51, 52, 53, 54, 55, 56};
```

- anche gli assegnamenti sono intuitivi:

```
int x = tabella[1][0] // assegna ad x 54
tabella[1][2] = 58; // sostituisce 58 a 55
```

Array multidimensionali

Non si inizializzano per default (a meno che non siano globali).
Se si inizializzano solo alcuni elementi (tipicamente) gli altri vengono inizializzati a 0.

```
int tabella[2][3] = {0};
```

inizializza a zero tutti gli elementi della matrice con 2 righe e 3 colonne.

Attenzione:

```
// OK
int tabella[][5] = {1,2,3,4,5,6,7,8,9,10};


// ERRORE!!!
int tabella[][] = {1,2,3,4,5,6,7,8,9,10};
```

Passaggio di array come parametro

Nel passaggio di un array come parametro di una funzione bisogna tenere a mente che ***il nome dell'array è un puntatore al primo elemento dell'array.***

```
int num[10];
int *p;
p = num;
```

Possiamo dire che:

- Sia *p* che *num* sono puntatori di *num[0]*.
- *num* e *&num[0]* sono la stessa cosa.
- *num[0]* e **num* sono la stessa cosa.
- *num[1]* e **(num + 1)* sono la stessa cosa.
- *x[n]* e **(num + n)* sono la stessa cosa (n è un intero).

Per questo motivo il passaggio di un array nelle funzioni avviene sempre per indirizzo.

Esercizi

1. Scrivere un programma che sfrutta un metodo ‘leggiElementi()’ per inizializzare un array di double di dimensione non nota, chiedendo all’utente di inserire un valore per volta fino a quando non viene inserito il valore zero. Successivamente, stampare gli elementi mediante un altro metodo ‘stampaElementi()’

2. Definire più metodi per stampare i numeri primi <=n, e valutare il loro tempo di esecuzione per valori di n crescenti.

3. Scrivere un programma che legga una frase, sostituisca tutte le sequenze di uno o due spazi con uno spazio singolo e visualizzi la frase risultante.

4. Scrivere un programma che inizializza solo i primi due elementi di un vettore di 4 e visualizzi il valore degli elementi non assegnati.

5. Scrivere una funzione di conversione che riceva come parametro una stringa rappresentante una data in formato gg/mm/aa e la restituisca nel formato testuale, es: 17/11/91 --> 17 novembre 1991

6. Definire un programma che visualizzi il triangolo di Pascal