arrays

- un *array* (o *vettore*) è una sequenza di oggetti 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]`

a	25.1	34.2	5.25	7.45	6.09	7.54
	0	1	2	3	4	5

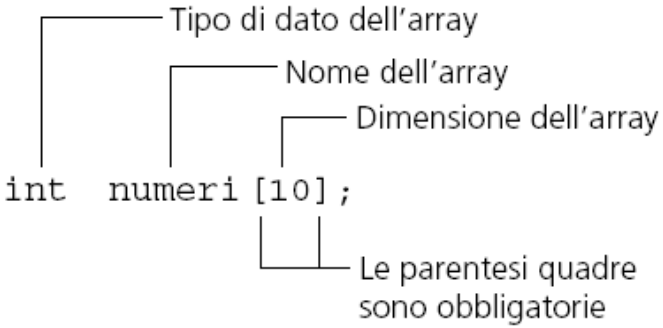
definizione di array

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 o un valore intero, o un'espressione costante; 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* scorrevole che ne rappresenta la posizione:

  nomeArray[n]

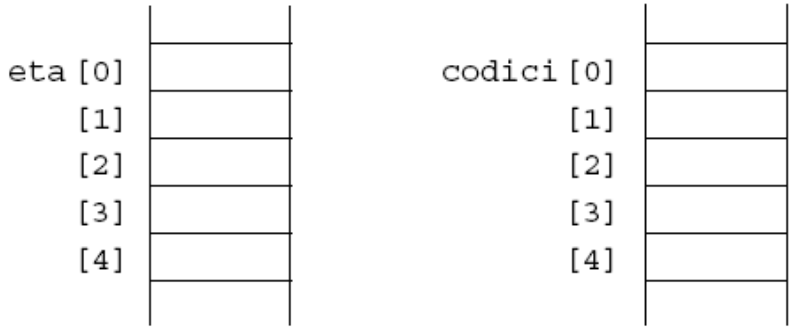
- C++ non verifica che gli indici dell'array stiano dentro la dimensione definita; così, ad esempio, se si accede a `numeri[12]` il compilatore non segnalerà alcun errore (*buffer overflow*)

```
int eta[5];          // contiene 5 elementi: il primo, eta[0] e l'ultimo, eta[4]
int pesi[25], lunghezze[100];  // definisce 2 arrays 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'emento 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 arrays si immagazzinano in blocchi contigui; così, ad esempio, gli arrays:

```
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]);
```

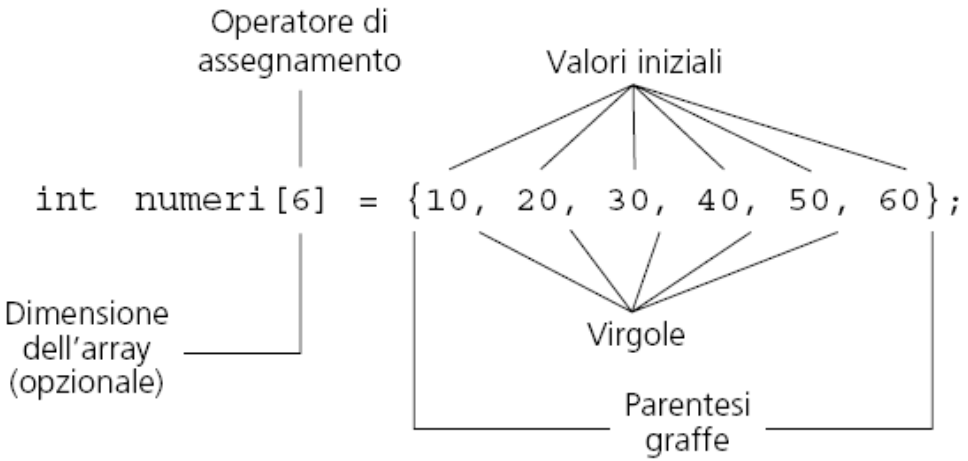

inizializzazione di un array

- per assegnare un valore ad un elemento di un array si può usare ovviamentre l'operatore di assegnamento:

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

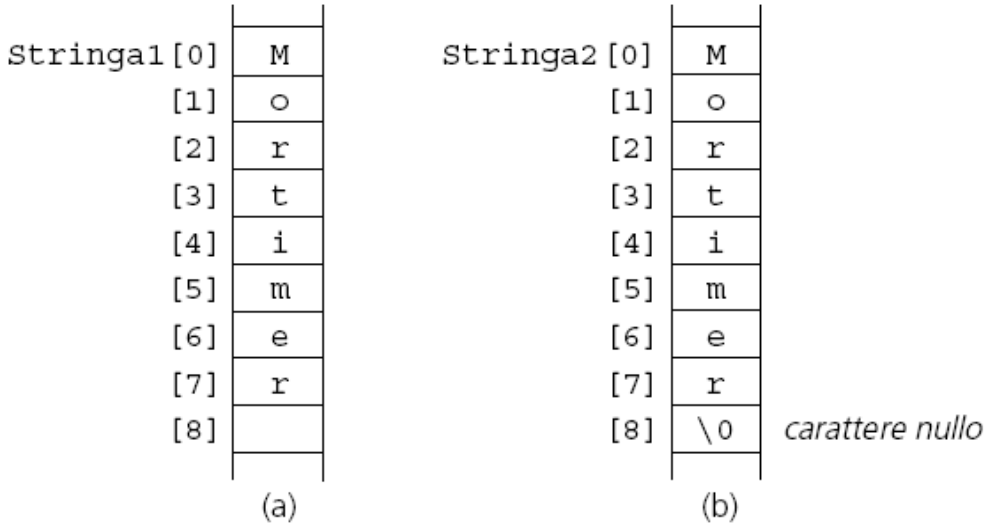
ma assegnare un intero array si può fare solo nella sua definizione:

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



array di caratteri e stringhe

- C++ rappresenta le stringhe di testo inserendole in arrays di caratteri terminandole 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 d un array al di fuori della definizione

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

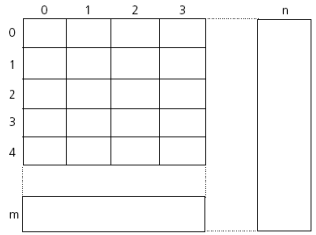
- Si utilizza la funzione di libreria strcpy (che aggiunge il carattere \0 alla fine della stringa)

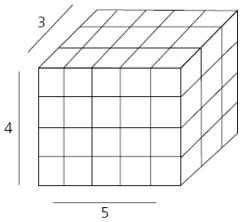
```
strcpy(nome, "Ciao");
```


array multidimensionali

- gli arrays di arrays 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 arrays di arrays di arrays 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] = {51};
```

inizializza a zero gli altri elementi

passaggio di array come argomenti

- ***gli arrays si passano per riferimento***;
  - quando si passa un array come parametro ad una funzione C++ tratta la chiamata come se vi fosse l'operatore di indirizzo & davanti al nome del vettore

```
main()
{
   char parola[4]= "ABC"
   cambiare(parola);
   cout << parola << endl;
   return;
}
```

parola →

```
cambiare(char c[4])
{
    cout << c << endl;
    strcpy(c, "AMA");
    return;
}
```

- il `4` nel `char[4]` della figura precedente può essere omesso.
- Passare un vettore significa passare *solo* il suo indirizzo.
- Se la funzione deve conoscere la dimensione del vettore, questa può essere passata come (secondo) parametro

Esempio: Segmenti di Somma Massima

- Segmento: sequenza di elementi consecutivi in un array a
  - a array di n interi
  - a[i,j] segmento se $0 \leq i \leq j \leq n-1$
- Determinare il segmento di somma massima
  - Banale se gli elementi sono tutti positivi (o tutti negativi)
  - A parità di somma si predilige il segmento più corto

