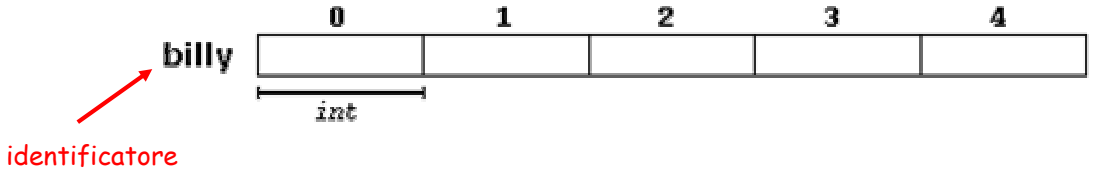
Strutture Dati

- Le strutture dati sono entità che permettono di memorizzare i dati in modo organizzato e funzionale a particolari compiti computazionali.
- Il loro utilizzo corretto spesso permette di realizzare algoritmi efficienti e veloci.
- Noi vedremo come alcune strutture dati possono essere definite ed utilizzate in C++, ed in particolare tratteremo i costrutti linguistici che riguardano:
  - Array;
  - Stringhe di caratteri;
  - Strutture;
  - Tipi definiti dall'utente.

Array

- Gli array sono sequenze di variabili dello stesso tipo che vengono situate consecutivamente nella memoria ed alle quali è possibile accedere usando uno stesso nome (identificatore) a cui viene aggiunto un indice.
- Ad esempio, possiamo memorizzare 5 valori di tipo **`int`** senza bisogno di dichiarare cinque diverse variabili con cinque diversi identificatori: è sufficiente dichiarare un array di cinque elementi dello stesso tipo **`int`** con un solo identificatore:



ogni cella rappresenta un elemento dell'array. Gli elementi sono numerati da **`0`** a **`4`** in quanto, in un array, l'indice del primo elemento è sempre **`0`** (e non **`1`**).

Array

- Come tutte le variabili anche gli array devono essere dichiarati prima di poterli usare.
- Un esempio di dichiarazione di un array in C++ è:

```
tipo nome [dimensione];
```

- dove **tipo** è il tipo degli elementi (**int, float** ...) detto anche tipo base dell'array , **nome** è un identificatore e **dimensione**, che deve essere racchiuso tra parentesi quadre **[]**, è la dimensione, ossia il numero di elementi, dell'array.
- La dichiarazione dell'array **billy** è:

```
int billy [5];
```

ATTENZIONE: Il campo **dimensione** deve essere un valore costante in quanto gli array sono blocchi di memoria di dimensione prefissata ed il compilatore deve conoscere esattamente quanta memoria serve per l'array prima che il programma venga eseguito.

Array: inizializzazione

- Come per le variabili semplici, anche per gli array è possibile specificare un valore iniziale. Ad esempio, con la dichiarazione:

```
int billy [5] = { 16, 2, 77, 40, 12071 };
```

l'array viene inizializzato come segue:

	0	1	2	3	4
billy	16	2	77	40	12071

- Il numero di valori usati per l'inizializzazione (quelli posti tra le parentesi grafe **{}**) deve essere esattamente uguale alla dimensione dell'array.
- In C++ è possibile anche usare la notazione:

```
int billy [] = { 16, 2, 77, 40, 12071 };
```

ed in questo caso viene assunto implicitamente come dimensione dell'array il numero di valori della lista di inizializzazione.

ATTENZIONE !

In un array senza inizializzazione il valore iniziale dei suoi elementi risulta indeterminato (i bit della memoria riservata per l'array conservano i valori lasciati dai programmi precedenti).

Array: accesso ai valori

- In ogni punto del programma in cui un array risulta visibile possiamo accedere individualmente ad uno degli elementi dell'array per leggerlo o modificarlo esattamente come esso fosse una normale variabile. Il formato è il seguente:

name[index]

- Proseguendo l'esempio dell'array **billy** di 5 elementi di tipo **int**, i nomi mediante i quali possiamo accedere a ciascun elemento dell'array sono quelli indicati sopra le singole celle nella figura seguente:

- Ad esempio, se vogliamo memorizzare il valore 75 nel terzo elemento di **billy** possiamo usare l'assegnamento:

```
billy[2] = 75;
```

oppure, per copiare il valore del terzo elemento nella variabile **a** possiamo usare:

```
a = billy[2];
```

Si comporta come una variabile di tipo **int**

Array: accesso ai valori

- Occorre notare i due diversi usi delle parentesi quadre [ ] con gli array:
  - nella dichiarazione di un array esse sono usate per indicare la dimensione dell'array ;
  - in tutti gli altri contesti esse vengono usate per specificare un indice per individuare un particolare elemento dell'array.

```
int billy[5];      // dichiarazione di un nuovo array di 5 elementi
billy[2] = 75;     // accesso ad un elemento particolare
                   // dell'array: quello di indice 2.
```

- Altre possibili operazioni con gli array sono:

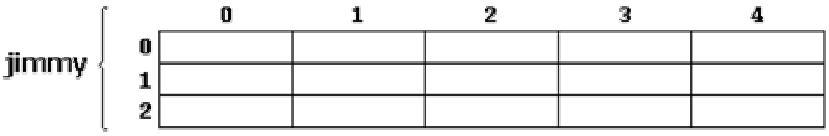
```
billy[0] = a;
billy[a] = 75;
b = billy [a+2];
billy[billy[a]] = billy[2] + 5;
```

```
// esempio con gli array
#include <iostream.h>

int billy [] = {16, 2, 77, 40, 12071};
int n, risultato=0;

int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    risultato += billy[n];
  }
  cout << risultato;
  return 0;
}
```

```
12206
```

Array multidimensionali

- Un array multidimensionale si può pensare come un array di array di array di ....
- Ad esempio, un array bidimensionale si può pensare come una tabella bidimensionale i cui elementi appartengono tutti allo stesso *tipo*.



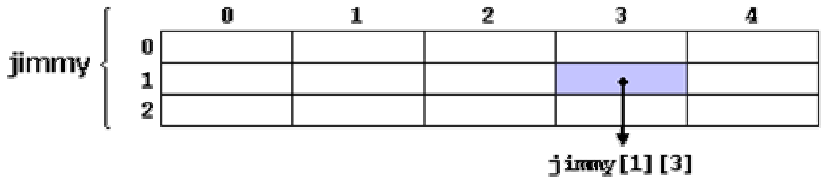
- **jimmy** è un array bidimensionale di 3 per 5 valori di tipo **int** .
- Esso si può pensare come un array di 5 elementi, ciscuno dei quali è a sua volta un array di 3 elementi (le colonne). Si dichiara come segue:

```
int jimmy [3][5];
```

e, per riferirsi all'elemento nella seconda riga e nella quarta colonna si usa la notazione:

```
jimmy[1][3]
```

ricordiamo che gli indici degli array iniziano sempre con 0.



Array multidimensionali

- Gli array multidimensionali possono avere più di due indici (due dimensioni). Ad esempio:

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

che però 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 gigabytes di memoria RAM.

- Gli elementi di un array multidimensionale sono memorizzati nella RAM uno di seguito all'altro come per gli array semplici:

```
int jimmy [3][5];    è equivalente a    int jimmy [15];
```

con l'unica differenza che il compilatore gestisce per noi la suddivisione in righe e colonne:

```
// array multidimensionale
#include <iostream.h>
#define COLONNE 5
#define RIGHE 3

int jimmy [RIGHE][COLONNE];
int n,m;
int main ()
{
  for (n=0;n<RIGHE;n++)
    for (m=0;m<COLONNE;m++)
    {
      jimmy[n][m]=(n+1)*(m+1);
    }
  return 0;
}
```

```
// array pseudo-multidimensionale
#include <iostream.h>
#define COLONNE 5
#define RIGHE 3

int jimmy [RIGHE * COLONNE];
int n,m;
int main ()
{
  for (n=0;n<RIGHE;n++)
    for (m=0;m<COLONNE;m++)
    {
      jimmy[n * COLONNE + m]=(n+1)*(m+1);
    }
  return 0;
}
```

Array multidimensionali

```
// array multidimensionale
#include <iostream.h>
#define COLONNE 5
#define RIGHE 3

int jimmy [RIGHE][COLONNE];
int n,m;
int main ()
{
  for (n=0;n<RIGHE;n++)
    for (m=0;m<COLONNE;m++)
    {
      jimmy[n][m]=(n+1)*(m+1);
    }
  return 0;
}
```

```
// array pseudo-multidimensionale
#include <iostream.h>
#define COLONNE 5
#define RIGHE 3

int jimmy [RIGHE * COLONNE];
int n,m;
int main ()
{
  for (n=0;n<RIGHE;n++)
    for (m=0;m<COLONNE;m++)
    {
      jimmy[n * COLONNE + m]=(n+1)*(m+1);
    }
  return 0;
}
```

- Entrambi i programmi assegnano i seguenti valori al blocco di memoria riservato per **jimmy**:

jimmy

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

- Notate l'uso della direttiva **#define** per facilitare ridimensionamenti dell'array.

Array come parametri

- Potremmo voler passare un array come parametro ad una funzione.
- In C++ non è possibile passare come parametro ad una funzione il valore di un array (ossia l'insieme dei valori di tutti i suoi elementi).
- Però possiamo passare come parametro l'indirizzo dell'array.
- Per indicare che un argomento di una funzione rappresenta un parametro di tipo array basta scrivere il tipo degli elementi dell'array (il tipo base dell'array) seguito da una coppia di parentesi quadre [].
- Ad esempio la funzione:

```
void procedura (int arg[])
```

aspetta un argomento **arg** di tipo "Array di **int** ". Per passare alla funzione l'array:

```
int mioarray [40];
```

è sufficiente scrivere una chiamata della funzione del tipo:

```
procedura (mioarray);
```

Array come parametri

Esempio di passaggio di un array come parametro

```
// array come parametri
#include <iostream.h>

void stampaarray (int arg[], int lunghezza)
{
 int n;
  for (n=0; n<lunghezza; n++)
    cout << arg[n] << " ";
  cout << "\n";
}

int main ()
{
  int primoarray[] = {5, 10, 15};
  int secondoarray[] = {2, 4, 6, 8, 10};
  stampaarray (primoarray,3);
  stampaarray (secondoarray,5);
  return 0;
}
```

```
5 10 15
2 4 6 8 10
```

Come si vede il primo argomento (**int arg[]**) ammette qualsiasi array con tipo base **int**, indipendentemente dalla sua dimensione.
Il secondo argomento segnala alla funzione quale sia la dimensione dell'array passato come primo argomento. La dimensione dell'array è necessaria perché il ciclo **for** possa stampare il giusto numero di elementi.

Array come parametri

- In una dichiarazione di funzione si possono anche usare array multidimensionali come argomenti.
- Ad esempio, la forma dell'argomento per un array tri-dimensionale è:

  `tipo_base [][d2][d3]`

  in cui devono essere specificate tutte le dimensioni ad esclusione della prima.
- Tale argomento ammette un qualsiasi array tridimensionale avente seconda e terza dimensione **d2** e **d3** prefissate e prima dimensione qualsiasi.
- Ad esempio:

  `void procedura (int mioarray[][3][4])`
- La ragione per cui **d2** e **d3** devono essere fissate è che il compilatore, per determinare la posizione in memoria dell'elemento **`mioarray[i][j][k]`** usa la formula **`i*3*4+j*4+k`** .

Esercizi su Array

- Esercizio 1:

  Scrivere un programma che riceve da tastiera il punteggio raggiunto da sei studenti e ne determina il maggiore, il minore e la media.

- Esercizio 2:

  Riscrivere il programma precedente dove il codice di ricezione dei dati in ingresso è raccolto in una funzione con il seguente prototipo:

```
void leggidatiingresso (int dati[], int lunghezza);
```