

Array

introduzione alla programmazione

tipi di dato

- tipo di dato semplice - può memorizzare un solo valore per volta
- tipo di dato strutturato - ogni elemento di quel tipo è in grado di memorizzare una raccolta di dati


gli array

- gli array sono un tipo di dato strutturato che ci permette di descrivere **sequenze di elementi *uniformi***
- gli elementi dell'array sono di un tipo di dato unico
 - array di interi
 - array di float
 - array di caratteri
 - array di ...

gli array: sintassi

- un array è formato da un numero prefissato di elementi dello stesso tipo
- In questa parte del corso ci concentreremo su **array unidimensionali** dove gli elementi saranno disposti in un elenco

tipodiDato nomeArray [espressioneIntera] ;



una volta valutata è un **intero positivo** che descrive la dimensione dell'array

esempio

```
int num[5];
```

num[0]	
num[1]	
num[2]	
num[3]	
num[4]	

rappresentazione
convenzionale

[0]	[1]	[2]	[3]	[4]

rappresentazione
abbreviata

accesso agli elementi dell'array

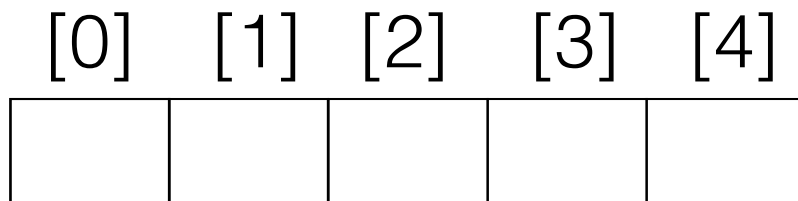
`nomeArray[espressioneIndice];`

specifica una posizione
nell'array a **partire da 0**

- `float angle[4];`
`angle[0]=4.93;`
`angle[1]=-1.2;`
`angle[2]=0.72;`
`angle[3]=1.67;`

[0]	[1]	[2]	[3]
4.93	-1.2	0.72	1.67

passo dopo passo



```
int list[5];
```

passo dopo passo

[0]	[1]	[2]	[3]	[4]
		45		

```
int list[5];  
list[2]=45;
```


passo dopo passo

[0]	[1]	[2]	[3]	[4]
		45	10	

```
int list[5];  
list[2]=45;  
list[1*2+1]=10;
```

passo dopo passo

[0]	[1]	[2]	[3]	[4]
		45	10	55


```
int list[5];  
list[2]=45;  
list[1*2+1]=10;  
list[4]=list[2]+list[3];
```

altri usi

- `cin >> angle[2];`
- `float y = sqrt(angle[0]);`
- `x=6.1*angle[2]+9;`

dimensione degli array

- gli array possono anche essere dichiarati in questo modo
- ```
const int ARRAY_SIZE = 10; //costante globale
int list[ARRAY_SIZE];
```
- Quando si dichiarano gli array la loro dimensione deve essere **nota al tempo di compilazione**
- Il seguente codice è sconsigliato (anche se ammesso da alcuni recenti compilatori):



- ```
int dimensione;  
cout << "inserisci dimensione" ;  
cin >> dimensione;  
int list[dimensione];
```

il compilatore non sa
quanta porzione
di memoria riservare
all'array

Indici fuori dai limiti (out of bound)

- `float a[10];`
`a[i]=4.5; //` ha senso per tutti i valori di `i` tra 0 e 9
- altrimenti ho un accesso ad una locazione di memoria non riservata al programma
- In C++, con gli array, non abbiamo una protezione nei confronti di questo fenomeno
 - non riceveremo un messaggio di errore specifico dal compilatore (ne' in fase di esecuzione) ne' avremo gli strumenti (la conoscenza) per poter sollevare eccezioni
- Il programma accederà all'elemento di memoria e l'effetto di questo accesso è imprevedibile

Indici fuori dai limiti (out of bound)

- Ricade sul programmatore l'onere di garantire il rispetto dei limiti!
- In seguito incontreremo altri strumenti che ci forniscono maggiori garanzie

ATTENZIONE!

char x[5]

è formato dai seguenti elementi:

x[0], x[1], x[2], x[3], x[4]

dichiarazione di array con inizializzazione

- INIZIALIZZAZIONE TOTALE

- `double sales[5]={1.4, 2.5, 14.5, 5.3, -1.4};`
- possiamo anche scrivere
`double sales[]={1.4, 2.5, 14.5, 5.3, -1.4};`
la dimensione è implicita nell'inizializzazione

[0]	[1]	[2]	[3]	[4]
1.4	2.5	14.5	5.3	-1.4

- INIZIALIZZAZIONE PARZIALE

- `double sales[5]={1.4};`

[0]	[1]	[2]	[3]	[4]
1.4				

Esercizio

- Scrivere un programma che legge da input una sequenza di numeri interi, li memorizza in un array e
 - Calcola la somma degli elementi dell'array
 - Calcola la media degli elementi dell'array
 - Calcola il massimo degli elementi dell'array (e stampa sia il valore massimo che la posizione occupata dal massimo)
- ... cosa cambia se modifichiamo il tipo base (da intero a char, string, float, ...?)

gli array non hanno operatori aggregati!

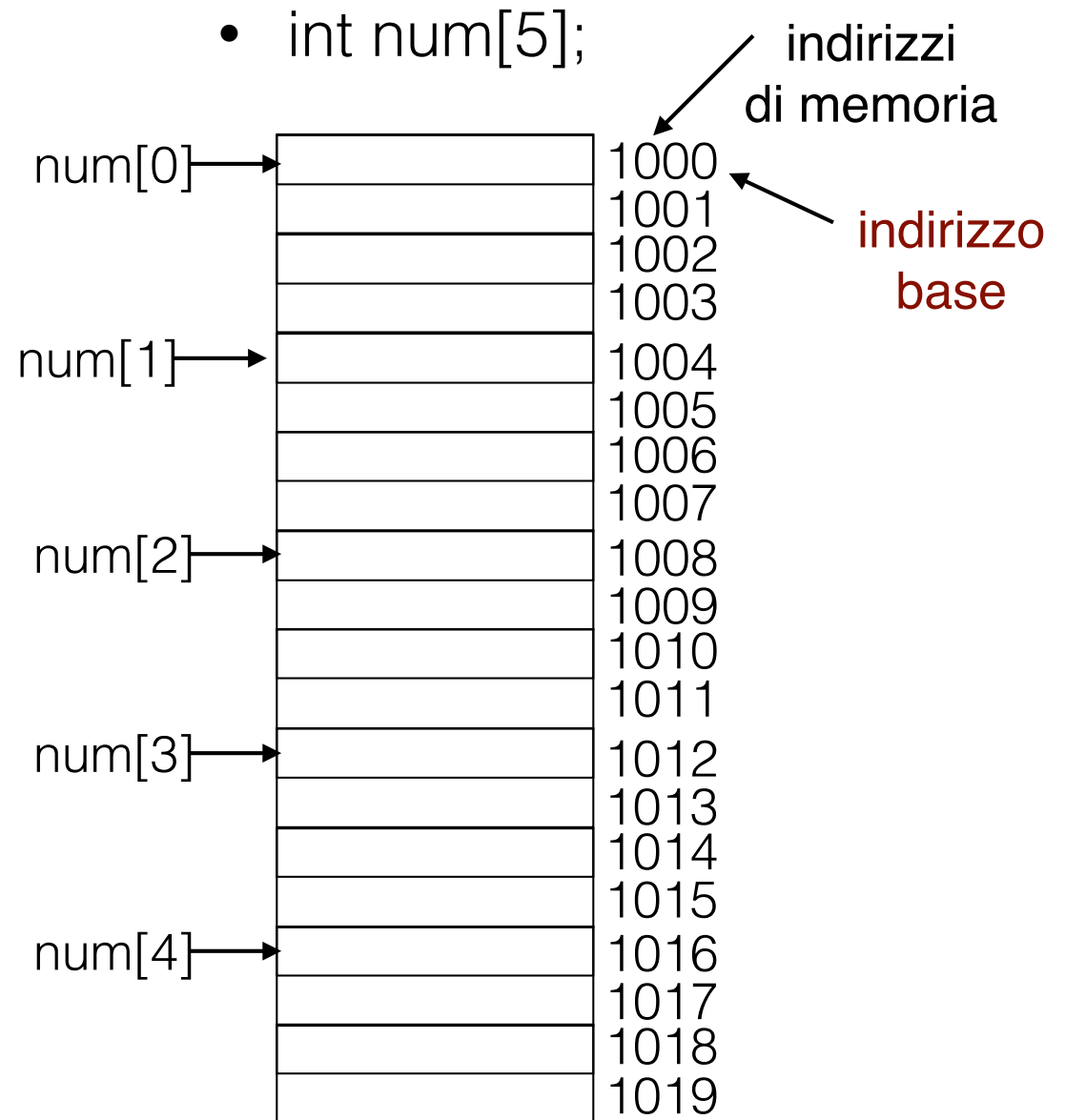
- `int list1[4]={1,2,3,4};`
`int list2[4];`
- `list2=list1;` **// non è corretto (come fare?)**
- `cin >> list2;` **// non è corretto (come sopra)**
- `if (list1 == list2)`
 **// è corretto ma non fa quello che vogliamo**
 // (confronta indirizzi di memoria)

indirizzo base di un array

- l'indirizzo base di un array è l'indirizzo di memoria del suo primo elemento

- `cout << num << endl;`
stampa l'indirizzo base

Indirizzo base `0x7fff57741ae0`
(in esadecimale)



caratteristiche di una variabile di tipo “array di...” in C++

```
float numeri[10];
```

- `numeri` NON è una variabile di tipo float;
`numeri[i]` (con `i` intero) lo è
- `numeri` NON È NEMMENO VARIABILE:
contiene un indirizzo di memoria prefissato
- ~~`numeri[10]`~~ NON è un elemento dell'array!!!
L'indice deve essere fra 0 e 9!!!
- `numeri` contiene una sola informazione: un indirizzo.
Non contiene la dimensione!

- se **A** è l'indirizzo base dell'array **x**
 - e **D** è la dimensione degli elementi di **x**
D = sizeof x[0];
 - allora l'indirizzo di **x[i]** si calcola così:
A + D*i
 - **A** è contenuta in **x**
e **D** è la dimensione del tipo degli elementi di **x**.
 - Il limite superiore per **i** invece (numero elementi)
NON È SCRITTO DA NESSUNA PARTE!!!!
-

repetita iuvant:

una variabile di tipo array

non contiene i valori dell'array

ma l'indirizzo in memoria

a partire dal quale

i valori sono memorizzati

Approfondimento: array di caratteri

- **Array di caratteri**- un array i cui elementi sono di tipo char
- Ha modalità e usi particolari - per questo lo trattiamo a parte
- negli insiemi di caratteri più comuni (ASCII e EBCDIC) il primo carattere è il *carattere null* '\0'
- quindi il carattere '\0' è inferiore a qualunque carattere

Array di caratteri e C-stringhe

- Attenzione!
 - una C-stringa è un array di caratteri che termina con un carattere null \0
 - Inoltre '\0' può comparire solo in fondo all'array
- Esempi

```
char name[16];  
char name[16]={'J','o','h','n'};  
char name[16]="John";  
char name []="John";
```
- ```
char name[16];
name="John";
```

 // non è corretto

# funzioni predefinite per C-stringhe

```
#include<cstring>
```

- strcpy(s1,s2) // copia la stringa s2 nella variabile s1
- strcmp(s1,s2) // confronta s1 e s2
- strlen(s) // restituisce la lunghezza di s
- char name[16];  
strcpy(name,"John"); // è corretto



# C-stringhe e I/O

- in questo ambito il C++ tratta in modo diverso le C-stringhe da qualunque altro array
- `char name[31];`  
`cin >> name;` // consente l'immissione da tastiera di  
                  //stringhe lunghe al massimo **30** caratteri  
                  // **senza spazi**           **attenzione! se inseriamo più caratteri**  
                                              **essi vengono mantenuti nello stream**
- `cout << name;` // visualizza il contenuto fino a `'\0'`  
  
                                              **quando inseriamo la stringa**  
                                              **da tastiera non servono le virgolette!**

# C-stringhe e file di I/O

- ifstream infile;  
char filename[50];

```
cout << "enter the input file: " << endl;
cin >> filename;
infile.open(filename);
```

# C-stringhe e string

- il tipo di dato **class** string non sono la stessa cosa delle C-stringhe
- in particolare non terminano con il '\0'
- in alcuni casi possono essere usate indifferentemente, in altri no
- ifstream infile;  
string filename;

```
cout << "enter the input file: " << endl;
cin >> filename;
infile.open(filename.c_str());
```