

Fondamenti di Informatica
Ingegneria Clinica
Lezione 4/12/2009



Raffaele Nicolussi
FUB - Fondazione Ugo Bordoni
Via B. Castiglione 59 - 00142 Roma

Puntatori e array

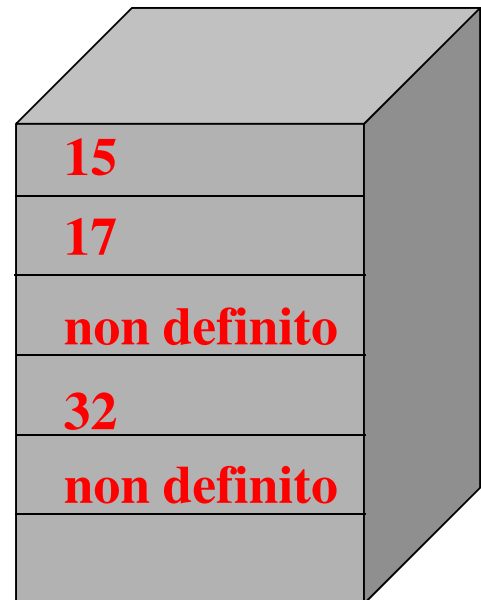
- Stretto legame tra puntatori e array
 - ⇒ **Il nome di un array è un puntatore**
 - Il nome dell'array è il puntatore al primo elemento dell'array
 - Il nome dell'array coincide con l'indirizzo del primo elemento
 - Se ar è un array di n elementi,
 ar coincide con $\&ar[0]$
- Quando viene inizializzato un array (per esempio di 10 elementi), sono allocati **10 blocchi di memoria CONSECUTIVI** dove memorizzare i valori degli elementi

⇒ il nome fa riferimento alla base dell'array, ossia alla prima cella di quell'array che è il suo primo elemento

```
int ar[5];  
ar[0] = 15;  
ar[1] = 17;  
ar[3] = ar[0] + ar[1];
```

ar[0]	1000
ar[1]	1004
ar[2]	1008
ar[3]	100C
ar[4]	1010
	1014

4 byte



◦ ar è &ar[0]

Il nome di un array che non è seguito da un indice né dalle parentesi viene interpretato come il puntatore all'elemento iniziale dell'array.

- Se il nome è un puntatore, allora possiamo estrarre il primo elemento usando l'**operatore di indirazione ***, ossia ***ar** dovrebbe tornare il valore di **ar[0]**.

```
int main(void) {
    int x[10] =
        {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    printf("Indirizzo di x[0]: %p\n",
           &x[0]);
    printf("Indirizzo x: %p\n", x);
    printf("Valore di x[0]: %d\n",
           x[0]);
    printf("Valore di x[0]: %d\n",
           *x);
    return 0;
}
```

val_ind.c

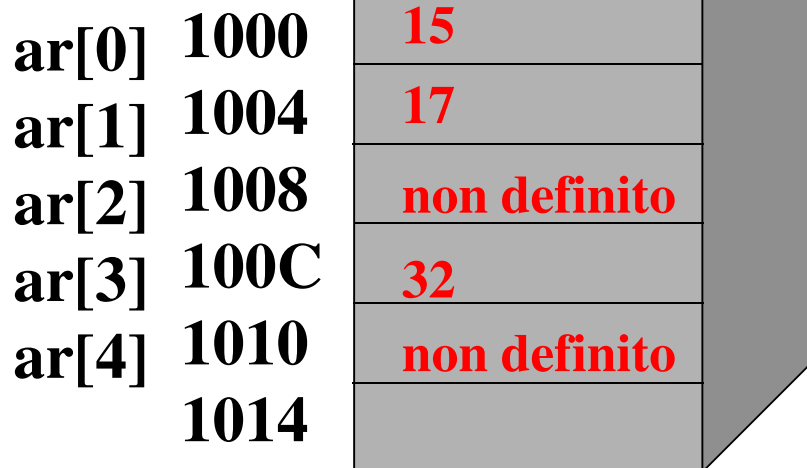
Indirizzo di x[0]: 0x0065FDD0

Indirizzo di x: 0x0065FDD0

Valore di x[0]: 0

Valore di x[0]: 0

4 byte



- è possibile accedere agli elementi di un array in due modi
 - nome dell'array con relativo indice
 - utilizzo del nome e puntatori

```
int *p;  
p = &ar[0]; (oppure p = ar;)
```

4 byte

```
int *p;  
p = &ar[0];  
p = ar;
```

p	ar[0]	1000
p+1	ar[1]	1004
p+2	ar[2]	1008
p+3	ar[3]	100C
p+4	ar[4]	1010
		1014



***p** da' il valore di **ar[0]**, ossia 15

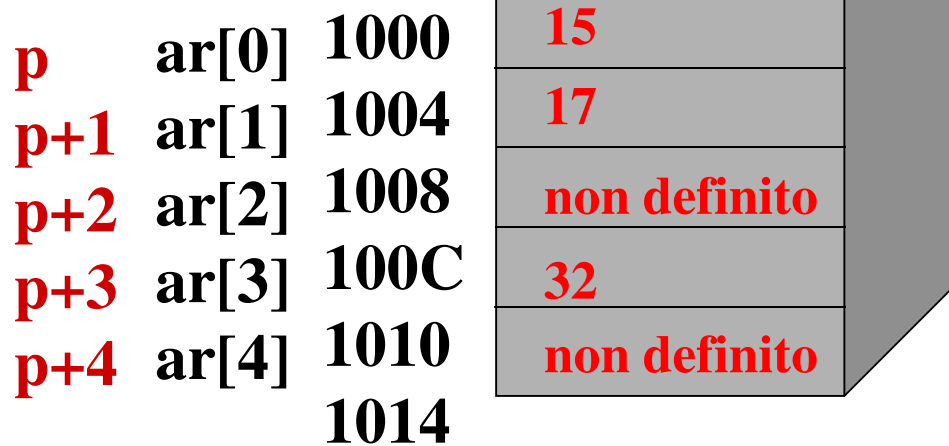
***(p +3)** corrisponde a **ar[3]**

° identificano la stessa locazione di memoria

***(p +e)** coincide con **ar[e]**

Fino a quando **p** non viene modificato **ar**
e **p** identificano la stessa locazione di
memoria

4 byte



***p e ar[0] identificano lo stesso valore**

***p** **→** **ar[0]** **→** **15**

***(p+ 1)** **→** **ar[1]** **→** **17**

*** p da' il valore di ar[0], ossia 15
(operatore di indirezione)**

- **(p+n)** è il puntatore all'ennesimo elemento dell'array
- **n** è l'offset rispetto al puntatore iniziale
 - ar => &ar[0]**
 - p = &ar[0];**
 - p => ar;**
 - *(p + e) = ar[e]**
 - ar[n] equivale a *(ar + n)**
- Un **nome di array** è, per il compilatore C, il **puntatore all'elemento iniziale dell'array** e quindi gli indici vengono interpretati come spostamenti dalla posizione dell'indirizzo base (offset)

***(p + n)** notazione con **puntatore** e **offset**

- Le **variabili puntatori** e i **nomi degli array** possono essere utilizzati indifferentemente per **accedere agli elementi di un array**.

***(p + n)** *notazione con puntatore e offset*

p[1] *notazione con puntatore e indice*

ar[2] *notazione con nome e indice*

- I **nomi degli array** non possono essere modificati
 - **non sono variabili**, ma **riferimenti a indirizzi** delle variabili di array
- Un array non può apparire a sinistra di un assegnamento a meno che sia associato ad un **indice** o a *****

float ar[5], *p;

p = ar; **Corretta.**

Equivale a **p = &ar[0]**

ar = p; **Errata.** Non è possibile fare
assegnamenti ad un indirizzo di
array *

&p = ar; **Errata.** Non è possibile fare
assegnamenti ad un indirizzo di
puntatore

ar++; **Errata.** Non è possibile
incrementare un indirizzo di
array

ar[1] = *(p+e);
Corretta.

ar[1] è una variabile

p++; **Corretta.** È possibile
incrementare una
variabile di puntatore

Quando viene dichiarato un array, viene riservato **spazio contiguo in memoria** per contenere tutti gli elementi dell'array

```
int a[100], i , *p, sum =0;
```

```
&a[0]--> 300, &a[1]--> 304, ...
```

(indirizzi degli elementi)

Sommare in **sum** tutti gli elementi dell'array

```
for (p = a; p < &a[100]; p++)  
    sum += *p;
```

```
for (i = 0; i < 100; i++)  
    sum += *(a + i);
```

```
p =a;  
for (i = 0; i < 100; i++)  
    sum += p[i];
```

```
for (i = 0; i < 100; i++)  
    sum += ar[i];
```

```
#include <stdio.h>

int main(void) {
    int *ptr;
    int arrayInts[10] =
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    ptr = arrayInts;
        /* ptr= &arrayInts[0]; */
    printf("Il puntatore sta puntando
           al primo");
    printf(" elemento dell'array, che
           e' %d.\n", *ptr);
    printf("Incrementiamolo ...\n");
    ptr++;
    printf("Ora dovrebbe puntare
           all'elemento successivo,");
    printf("che e' %d.\n", *ptr);
}
```

punt_array.c

```

printf("Ora supponiamo di
       puntare al terzo e quarto:
       %d%d.\n",
       *(ptr+1), *(ptr+2));
ptr+=2;
printf("Ora saltiamo i
       prossimi 4 e puntare
       all'ottavo: %d.\n",
       *(ptr+=4));

ptr--;
printf("Ho mancato il numero
       %d?!\n", *(ptr++));
printf("Prima dell'ottavo
       e poi ..... %d.\n",
       *ptr);
return 0;
}

```

Output :

Il puntatore sta puntando al primo elemento dell'array, che e' **1**.

Incrementiamolo ...

Ora dovrebbe puntare all'elemento successivo, che e' **2**.

Ora supponiamo di puntare al terzo e quarto: **3 4**.

Ora saltiamo i prossimi 4 e puntare all'ottavo: **8**.

Ho mancato il numero **7**?!

Prima dell'ottavo e poi
8.

Passare i vettori alle funzioni

- Piuttosto che passare valori singoli ad una funzione, può essere comodo memorizzarli in un vettore e poi passarli alla funzione stessa
- Il vettore può essere usato come parametro d'ingresso per le funzioni
- Nella chiamata si usa il nome dell'array
 - **chiamo_funzione(<nome_array>);**
- Nella definizione e nel prototipo vanno specificati il tipo, il nome e le parentesi quadre
 - <tipo> chiama_funzione**
 - (< tipo> <nome_array>[])**
 - **NON IL NUMERO DI ELEMENTI**

- **dichiarazione array:**
int temp_oraria [24];
- **prototipo/definizione:**
int cambia_array(int []);
int cambia_array(int b[]) { ... }
- **chiamata:**
int ris = cambia_array (temp_oraria);

Il numero di elementi dell'array non è noto alle funzioni chiamate:

- o usiamo un identificatore di costante
- o passiamo la dimensione

- **dichiarazione array:**
int temp_oraria [24];
- **prototipo/definizione:**
int cambia_array(int [], int);
int cambia_array(int b[], int n) {...}
- **chiamata:**
cambia_array (temp_oraria, 24);

- La lista dei parametri di una funzione deve specificare **in modo esplicito che l'argomento è un array**

intestazione : `cambia_array(int b[], int size)`

- La funzione **cambia_array** ha come argomenti un **vettore di interi b** e una variabile **intera size**
 - Non si mettono le dimensioni del vettore all'interno delle parentesi quadrate nell'intestazione (tanto verrebbero ignorate)
 - Le parentesi sono obbligatorie
- Quando la funzione sarà chiamata lavorerà direttamente sul vettore corrispondente nella funzione chiamante

prototipo:

`cambia_array (int [], int)`

chiamata:

`cambia_array(b, 24)`

```

#include <stdio.h>
int addNumbers(int []);
int main(void) {
    int array[5];
    int i;
    printf("Enter 5 integers
separated by spaces: ");
    for(i=0 ; i<5 ; i++)
        scanf("%d", &array[i]);
    printf("\nTheir sum is: %d\n",
        addNumbers(array));
    return 0;
}
int addNumbers(int fiveNumbers[])
{
    int sum = 0;
    int i;
    for(i=0 ; i<5 ; i++)
        sum+=fiveNumbers[i];
    return sum;
}

```

somma_array.c

```
int addNumbers (int fiveNumbers []);  
int main(void) {  
    ...  
        addNumbers (array) );  
...  
}  
int addNumbers (int  
    fiveNumbers []) { .. }
```

- La dimensione dell'array è lasciata in bianco sia nel prototipo che nella definizione

Passare i vettori alle funzioni

- Un vettore viene passato ad una funzione **attraverso il suo nome** (senza parentesi quadrate)
 - Il nome è un puntatore
 - Quindi viene **passato un indirizzo!**
- **Stiamo implementando una chiamata per riferimento in cui le modifiche sui parametri formali si ripercuotono sui parametri attuali perché lavoriamo sulle stesse locazioni di memoria**
 - In questo modo si evita di **fare una copia dell'array (costoso in termini di spazio e tempo)** (cosa che accadrebbe con la chiamata per valore)
- **Attenzione:** La funzione chiamata può modificare i valori degli elementi

- I vettori sono passati tramite una chiamata per riferimento simulata, ma i singoli elementi sono passati per valore
 - Per passare un singolo elemento si usa **il suo nome con l'indice relativo**

```
dichiarazione: int temp_oraria [24];  
chiamata: cambia_valore (temp_oraria[3]);
```

- I valori in questo caso non sono modificati perché passati per valore
 - Per cambiarli devono essere passati per indirizzo

```
cambia_valore (&temp_oraria[3]);
```

Esercizio :: DNA

- L'informazione genetica del DNA, é codificata nella sequenza di basi (adenina, guanina, citosina e timina) che lo formano.
- Per convenzione, sequenze di DNA sono rappresentate come liste di lettere 'A', 'G', 'C', 'T'.
- Vogliamo analizza sequenze di questo tipo, di lunghezza fissata DIM, che rappresentiamo come array di caratteri 'A', 'G', 'C', 'T'

Inizializzare l'array nel main

A[]={'A', 'G', 'T', 'A', 'C', 'A', 'T', 'G', 'T', 'A'}

int DIM = 10

DNA

1. Scrivere un programma che stampa quante volte ciascun carattere è presente
2. Scrivere un programma che, dato un array di caratteri 'A', 'G', 'C', 'T', elimina dall'array *la prima occorrenza di 'A'*, e stampa l'array risultante.
3. Scrivere un programma che, dato un array di caratteri 'A', 'G', 'C', 'T', elimina dall'array *tutte le occorrenze di 'A'*, e stampa l'array risultante.
4. Trasformare i programmi in funzioni

DNA

- **Da pensare:** cosa significa eliminare dall'array?
- **Attenzione:** non vogliamo lasciare “vuoti”...
- **Sugg:** la dimensione dell'array é fissa, ma possiamo tener conto, in una variabile dedicata, del numero di elementi significativi (la lunghezza che ci interessa), oppure del livello di riempimento (ultimo indice significativo).

- **Algoritmo semplice:**
 1. *creo un secondo array di appoggio, B*
 2. *scorro il primo e copio nel secondo solo gli elementi che mi interessano*
 - *Attenzione: avrò bisogno di un `indice_array1` e `indice_array2`*
 - *`indice_array1` :: scorre A*
 - *`indice_array2` :: scorre B*
 3. *Stampo l'array ottenuto*

DNA

◦ Algoritmo in versione **tosta**:

1. Fino a che non è finito A (ciclo su indice **i**)

2. A[**i**] va eliminato? (if)

1. **SI**

a. **sposta** tutto il resto dell'array **A** (da **i+1** fino a **DIM-1**) copiandolo nelle posizioni da **i** a **DIM-2**

b. **DIM--;**

2. **NO**

1. non fare nulla (**if** senza **else**)

3. sposta ::

1. for (k=i; k<DIM-1; k++)

A[K] = A[K+1]

Chiamata per indirizzo

- È possibile **simulare la chiamata per indirizzo** in cui i **valori dei parametri attuali seguano le (eventuali) modifiche dei parametri formali**
 - **L'indirizzo dei parametri formali e quello dei parametri attuali è lo stesso**
 - **per effettuare una chiamata per indirizzo in una funzione si usano**
 - **i puntatori**
 - **l'operatore di indirectione (*)**

Chiamata per indirizzo: funz. chiamante

- Quando la funzione viene chiamata, devono essere passati come parametri gli **indirizzi delle variabili**
 - il passaggio degli indirizzi potrà essere ottenuto applicando **l'operatore di indirizzo & alla variabile che deve essere modificata**
 - sono passate le “celle” fisiche dove i valori sono memorizzati

&a

a = 2

&b

b = 3

```
int a; int b;  
a = 2; b = 3;  
ordina (&a, &b)
```

Chiamata per indirizzo

- Nella definizione della funzione chiamata deve essere usato l'operatore di deferimento per le variabili passate per indirizzo
 - Operatore di deferimento nel corpo della funzione

Pa

***Pa = 6**

Pb

***Pb = 7**

```
void ordina
(int *Pa, int *Pb)
{
    .....
    *Pa = 6 ;
    *Pb= 7; .... }

```

```
int a; int b;  
a = 2; b = 3;  
ordina (&a, &b)
```

&a

6

&b

7

```
void ordina(int *Pa, int *Pb)  
{ .....  
  *Pa=6;  
  *Pb=7;  
.....}
```

```
#include <stdio.h>
```

```
void ordina (int *, int *);
```

```
int main (void)
```

```
{ int a, b;
```

```
printf("Introduci due interi da  
ordinare \n");
```

```
scanf("%d %d", &a, &b);
```

```
ordina (&a, &b);
```

```
printf("Numeri ordinati %d  
%d\n", a, b);
```

```
return 0; }
```

```
void ordina (int *p, int *q)
```

```
{ int aux;
```

```
if (*p > *q)
```

```
{ aux = *p;
```

```
*p = *q;
```

```
*q = aux; } }
```

ordina_ind.c