



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI  
INFORMATICA - SCIENZA E INGEGNERIA

# ARRAY

**COSIMO LANEVE**

`cosimo.laneve@unibo.it`

**CORSO 00819 – PROGRAMMAZIONE**

# ARGOMENTI (CAPITOLO 7 SAVITCH)

1. array: dichiarazioni e esempi
2. array come argomento di funzioni
3. ricerca di elementi in un array
4. ordinamento di array
5. esempi/esercizi

# MOTIVAZIONI

**esempio:** scrivere una funzione che legge 20 interi e stampa il valore più vicino alla loro media

## **soluzione senza array:**

- \* definire 20 variabili di tipo `int` che memorizzano gli interi presi in input
- \* calcolare la media
- \* stampare la variabile che è più vicina alla media
- \* (utilizzare la funzione `abs` per calcolare la distanza in valore assoluto)

# MOTIVAZIONI

```
int average20(){
    int var0, . . . , var19 ;
    int r, dist;
    double m ;
    cin >> var0 ; . . . ; cin >> var19 ;
    m = (var0 + . . . + var19)/20.0 ;
    r = var0 ; dist = abs(m - var0) ;
    if (abs(m-var1) < dist) { dist = abs(m-var1) ; r = var1 ;}
    . . .
    if (abs(m-var19)< dist) { dist = abs(m-var19); r = var19;}
    return(r) ;
}
```

**osservazione:** sulle 20 variabili ripetiamo sempre le stesse operazioni

**si agisce in modo uniforme**

**nuovo problema:** risolvere il problema precedente quando le variabili sono 20.000

\* bisogna allungare il codice con le nuove variabili, i nuovi input, e i nuovi if

\* **per ovviare a questi problemi si usano gli array**

# ARRAY

sono ***collezioni di dati dello stesso tipo*** a cui viene associato un unico nome simbolico

- \* i dati appartenenti all'array sono detti **elementi** dell'array
- \* gli elementi dell'array vengono memorizzati in celle di memoria contigue

**motivazione:** avere una struttura che memorizza un numero finito di elementi dello stesso tipo su cui bisogna operare in  
**maniera uniforme**

# DICHIARAZIONI DI ARRAY

**sintassi**: *variable-declaration ::= ...*

*type id [int-const-no-sign] ;*

- \* *type* è il **tipo di base**
- \* *int-const-no-sign* è la **dimensione** dell'array

## **semantica**

- \* ***viene allocato spazio in memoria*** per contenere l'array il cui identificatore è *id*, costituito da *int-const-no-sign* elementi
- \* ogni elemento ***contiene*** un dato del tipo di base (esempi *int*, *double*, *char*, ma anche altri array...)
- \* la dimensione *int-const-no-sign* è **un'espressione costante** di tipo *int* ( $\geq 0$ )

# ARRAY/ESEMPIO

array per le variabili del programma del valore più vicino alla media

```
const int length = 20 ;  
int A [length] ;
```

per modificare la lunghezza dell'array è sufficiente cambiare la costante **length**

# ACCESSO AGLI ELEMENTI DI UN ARRAY

- \* ogni elemento dell'array **contiene** un dato del tipo di base (esempi `int`, `double`, `char`, ma anche altri array...)
- \* **si accede ai singoli elementi** mediante i termini  $id[0]$ ,  $id[1]$ , ...  
 $id[\text{int-const-no-sign} - 1]$
- \* il contenuto delle parentesi `[ . ]` è detto **indice**
- \* un indice è **valido** se assume un valore compreso tra 0 e  $\text{int-const-no-sign} - 1$  (dimensione dell'array meno 1)

**sintassi**:  $\text{expression} ::= \dots \mid \text{variable}[\text{int-expression}]$

**esempio**:  $A[i + 1]$

espressione di tipo `int`



**semantica**: al tempo di esecuzione, viene valutato l'**indice**, il valore ottenuto determina a quale elemento dell'array ci riferiamo



# ARRAY/ASSEGNAIMENTI

la sintassi dell'~~operazione di assegnamento~~ usata finora è

~~*variable = expression ;*~~

in realtà la sintassi di C++ è

*lhs-expression = expression ;*

dove *lhs* è un acronimo per ***left-hand-side***

- \* una *lhs-expression* è un termine che rappresenta un **indirizzo** di memoria
- \* una *variable* è una *lhs-expression*

# ARRAY/ASSEGNAMENTI

per assegnare un valore ad un elemento di un array: **usare l'operazione di assegnamento**

*id*[*int-expression*] è una ***lhs-expression***!

**esempio:**

```
int n = 2 ;  
A[n+2] = 35 ;
```

assegna all'elemento `A[4]` il valore 35

# ACCESSO SEQUENZIALE AGLI ARRAY

è spesso necessario elaborare gli elementi di un array in sequenza, partendo dal primo elemento

*di solito si utilizza un ciclo for, la cui variabile di controllo viene usata come indice dell'array*

## **esempi:**

\* inizializzazione del contenuto dell'array

```
int A[length], i;
```

```
for (i = 0; i < length; i = i+1) cin >> A[i] ;
```

il primo indice è  $i = 0$

\* somma degli elementi

```
int sum = 0 ;
```

```
for (i = 0; i < length;  $i_{i1} = i+1$ ) sum = sum + A[i] ;
```

l'ultimo indice è  $i = \text{length}-1$

# VALORE DI UN ARRAY PIÙ PROSSIMO ALLA MEDIA

```
const int length = 20 ;

int average() ;
// Postcondition: ritorna il valore memorizzato in var[length] più
//                vicino al valor medio

int average(){
    int var[length] ;
    int i, r, dist;
    double m = 0 ;
    for (i = 0; i < length ; i = i+1)      {   cin >> var[i] ;
                                                m = m + var[i] ; }

    m = m/(double)length ;
    r = var[0] ;
    dist = abs(m - var[0]) ;
    for (i = 1; i < length ; i = i+1)
        if (abs(m-var[i]) < dist){
            dist = abs(m-var[i]); r = var[i]; }
    return(r) ;
}
```

**osservazione:** modificare il programma per adattarlo a 200 variabili si riduce a cambiare la linea

```
const int length = 20 ;
```

# ARRAY/ERRORI

lunghezza dell'array: **NON PUÒ ESSERE** variabile

```
cout << "lunghezza dell'array?>" ;  
cin >> length ;  
int A [length] ;
```

**ERRORE!**  
**su molti compilatori**

**in C++: dynamic arrays**

accesso a elementi al di fuori della lunghezza (**out-of-range**)

```
const int length = 10 ;  
int i ;  
for (i = 0 ; i <= length ; i = i+1) A[i] = i*i ;
```

**ERRORE!**

**in C++ non viene segnalato alcun errore (!)**

- \* `A[length]` è l'indirizzo di una cella di memoria che potrebbe corrispondere ad un'altra variabile
- \* il risultato è **non predicibile**
- \* in molti linguaggi di programmazione (**e nei compiti**) ciò è proibito

# ARRAY PASSATI COME PARAMETRI


nella funzione `average` l'array `int var[length] ;`

è ragionevole che venga passato come argomento

\* l'inizializzazione viene fatta dal chiamante

- è **buona norma** passare sia l'array che la sua lunghezza
- la sua lunghezza è passata per **costante**

**esempio:**



```
int average(int vec[], const int n){
    int i, r, dist;
    double m = 0 ;
    for (i = 0; i < n ; i = i+1) m = m + vec[i] ;
    m = m/n ;
    r = vec[0] ; dist = abs(m - vec[0]) ;
    for (i = 1; i < n ; i = i+1)
        if (abs(m-vec[i]) < dist){
            dist = abs(m-vec[i]); r = vec[i] ; }
    return(r) ;
}
```

# ARRAY PASSATI COME PARAMETRI/CONT.

1. un parametro formale **può essere un array**
2. l'**array è passato per riferimento** -- la funzione ha accesso all'indirizzo del primo elemento dell'array (conoscendo quello e conoscendo il tipo di base, è possibile accedere a qualunque elemento dell'array)
3. un parametro formale di tipo array è specificato utilizzando le parentesi quadre "[ ]" **senza alcun indice**

**sintassi:**     *formal-parameter ::= ... | type id[ ]*

**esempio:**     `int average(int vec[ ], int n){ ... }`

4. il parametro attuale di tipo array, in fase di chiamata, è specificato **solamente dall'identificatore** (senza usare le [ ] )

**esempio:**     `average(A, length)`

# ARRAY PASSATI COME PARAMETRI/DETTAGLI

## cosa conosce il calcolatore di un'array?

- \* il tipo di base
- \* l'indirizzo del primo elemento dell'array
- \* la lunghezza dell'array

## cosa conosce una funzione di un'array?

- \* il tipo di base
- \* l'indirizzo del primo elemento dell'array

***poiché le funzioni non conoscono la lunghezza dell'array***

- \* quando si definisce la funzione occorre includere un parametro formale che specifica tale lunghezza

**conseguenza:** la funzione può operare su array di differenti dimensioni

in C++ gli array **non possono essere ritornati** come valore di funzioni



# ESERCIZI

1. scrivere una funzione che dato un numero binario memorizzato in un array lo converte in decimale
2. palindrome: un array di caratteri è palindromo se leggendolo da destra verso sinistra o da sinistra verso destra si ottiene lo stesso array. Scrivere una funzione che verifica se un array è palindromo o meno
3. scrivere una funzione che prende in input un array di interi **a** e un intero **n** e ritorna **true** o **false** a seconda che **n** si trova in **a** oppure no
4. scrivere una funzione che prende in input un array di interi e ne stampa gli elementi senza stampare i duplicati

# ARRAY PASSATI COME PARAMETRI/CONST

quando una funzione non modifica un array (accede all' array in lettura) conviene passare l'array in modalità **const**

**esempio:** calcolo del valore più prossimo alla media

```
int average(const int vec[], int n){ ... }
```

- \* il **compilatore darà errore** se l'array compare come *lhs-expression* nel corpo della funzione
- \* il **compilatore darà errore** se l'array viene passato come argomento a una seconda funzione il cui parametro **non** è **const**

# ARRAY PASSATI COME PARAMETRI/ESEMPIO

memorizzare la somma di due array in un terzo array:

```
void add_arrays(const int A[], const int B[], int SUM[], int n){
    int i;
    for (i = 0; i < n; i = i+1) SUM[i] = A[i] + B[i] ;
}

int main(){
    const int length = 100 ;
    int vec1[length], vec2[length], vecsum[length], i ;
    for (i=0 ; i < length; i=i+1){
        vec1[i] = 2*i ; vec2[i] = 2*i+1 ;
    }
    add_arrays(vec1,vec2,vecsum,length) ;
    return(0) ;
}
```

cosa succede se si rimpiazza il testo in rosso con  
`add_arrays(vec1,vec2,vec1,length) ;` ?  
in molti compilatori non viene segnalato!

# ARRAY RIEMPITI PARZIALMENTE

alcuni programmi manipolano sequenze di dati di lunghezza diversa

- \* si può utilizzare lo stesso tipo di array per contenere sequenze diverse
- \* occorre dichiarare array di dimensioni sufficienti a contenere la sequenza più lunga
- \* occorre tener traccia del numero di elementi contenuti nell'array — di solito conviene restituirlo come valore di ritorno dell'eventuale funzione
- \* in questi casi **conviene** utilizzare un ciclo `while`

# ARRAY RIEMPITI PARZIALMENTE/ESERCIZIO

prendere in input numeri interi fino a un massimo di 100 oppure finchè l'utente non inserisce uno 0, quindi stampare la sequenza inserita al contrario

## algoritmo:

1. si utilizza un array di interi di cento elementi (`length = 100`)
2. si utilizza una variabile booleana sentinella `got_0`
3. iterare finchè `i < length` e `got_0` è falso (`i` è l'indice):
  - ogni volta prendere in input un intero e se diverso da 0 memorizzarlo nell'array, altrimenti uscire dal ciclo ponendo `got_0 = true`

# ARRAY RIEMPITI PARZIALMENTE/ESERCIZIO

```
void fill_to_sentinel(int A[], int length){
    bool get_0 = false ;
    int x ;
    int i = 0 ;
    while ((i < length) && !get_0) {
        cin >> x ;
        if (x == 0) get_0 = true ;
        else {
            A[i] = x ;
            i = i+1 ;
        }
    }
    i = i-1 ;
    for (int j = i ; j >= 0 ; j = j-1) cout << A[j] ;
}
```

# RICERCA DI UN ELEMENTO IN UN ARRAY

verificare se un certo elemento **k** è **presente o meno** in un array

## algoritmo:

1. si utilizza un ciclo per esaminare gli elementi dell'array uno alla volta, confrontandoli con **k**
2. quando si trova un valore uguale a **k**, si esce dal ciclo
3. si utilizza un variabile bool per indicare che il valore è stato trovato e si può uscire dal ciclo (uso del comando **while**)
4. la funzione ritorna l'indice dell'elemento
5. costo computazionale caso pessimo (quando l'elemento non è presente): **lunghezza\_dell'\_array**

# RICERCA DI UN ELEMENTO IN UN ARRAY/IMPLEMEN.

- \* definiamo una funzione `search` che prende un array, la sua lunghezza `length` e il valore `k` da trovare
- \* la funzione ritorna la posizione di `k` nell'array
- \* se l'elemento non si trova, `search` ritorna `length`

```
int search(int A[], int length, int k) ;  
  
// Precondition: length > 0  
// Postcondition: ritorna i per cui, se i<length allora  
// A[i]=k  
  
int search(int A[], int length, int k){  
    bool found = false ;  
    int i = 0 ;  
    while (!found && (i < length))  
        if (A[i] == k) found = true ; else i = i+1 ;  
    return (i);  
}
```



# ALGORITMI DI ORDINAMENTO DI ARRAY (SORTING)

**ordinare** una lista di valori è una operazione molto comune

- \* creare una ordinamento di studenti in ordine alfabetico
- \* ordinare in maniera crescente
- \* ordinare in maniera decrescente

ci sono molti algoritmi di ordinamento

- \* alcuni sono semplici da comprendere
- \* altri sono molto efficienti computazionalmente
- \* **esempi:** selection-sort, bubble-sort, quicksort, mergesort

# ALGORITMI DI ORDINAMENTO DI ARRAY (SORTING)

quando l'ordinamento dell'array **A** è completo si avrà un array in cui

$$A[0] \leq A[1] \leq \dots \leq A[\text{length}-1]$$

ciò porta a un algoritmo molto semplice:

```
for (int i = 0 ; i < length ; i = i+1)
    metti in A[i] l' i-esimo valore più piccolo
    memorizzato in A
```

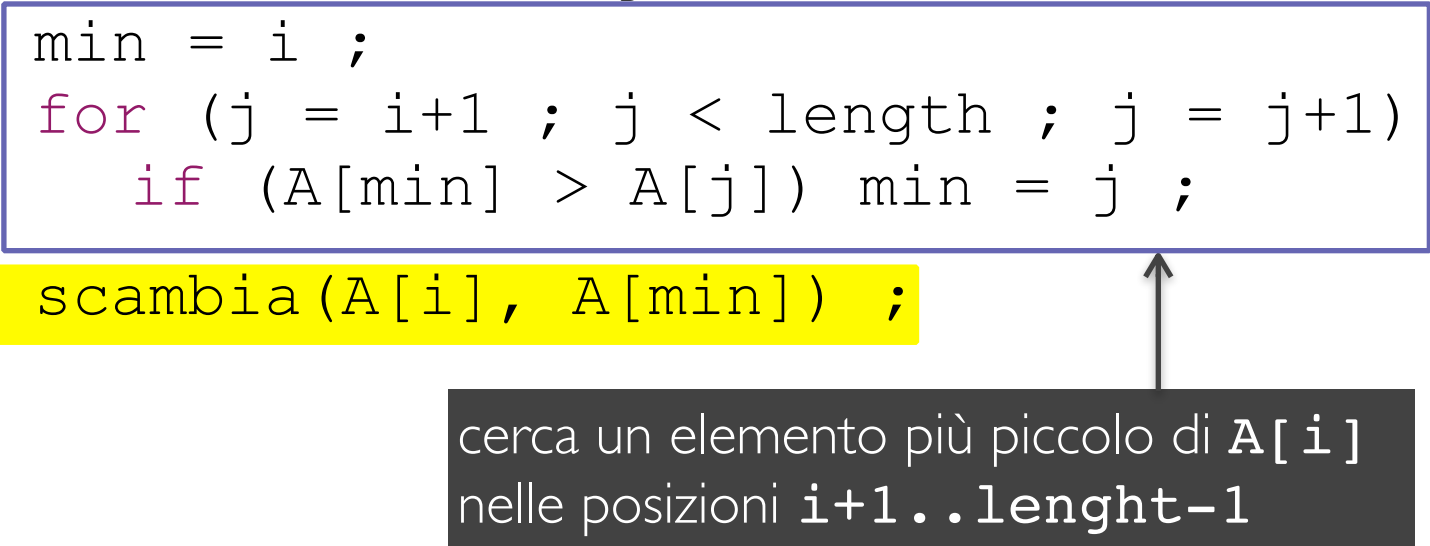
# ORDINAMENTO DI ARRAY/SELECTION-SORT

un algoritmo per ordinare un array è:

1. ricerca il più piccolo valore nell'array  $A$  e sia  $i$  la sua posizione
2. sostituisci  $A[0]$  con  $A[i]$
3. iniziando da  $A[1]$  ricerca il più piccolo valore nell'array e sostituiscilo con  $A[1]$
4. iniziando da  $A[2]$  ricerca il più piccolo valore nell'array e sostituiscilo con  $A[2]$
5. . . .

# ORDINAMENTO DI ARRAY/SELECTION SORT

```
void scambia (int& x, int& y) {  
    int tmp ; tmp = x ; x = y ; y = tmp ;  
}  
  
void selection_sort(int A[], int length) {  
    int i, j, min ;  
    for (i = 0 ; i < length-1 ; i = i+1) {  
        min = i ;  
        for (j = i+1 ; j < length ; j = j+1)  
            if (A[min] > A[j]) min = j ;  
        scambia(A[i], A[min]) ;  
    }  
}
```



# ORDINAMENTO DI ARRAY/BUBBLE SORT

**bubble-sort** : fa "galleggiare" verso la parte destra il valore più grande scambiando via via gli elementi consecutivi

**esempio:**    3        10       9       2       5  
                 ^

non c'è scambio perché  $3 \leq 10$

             3        10       9       2       5  
                 ^

c'è scambio perché  $10 > 9$

             3        9        10       2       5  
                                 ^

# ORDINAMENTO DI ARRAY/BUBBLE SORT

dopo un po di scambi:

3      9      2      5      10  
                                 ^

galleggia il valore  
più grande!

si ricomincia considerando l'array meno l'ultimo elemento . . .

3      9      2      5      10  
                                 ^

non c'è scambio perché  $3 \leq 9$

3      9      2      5      10  
                                 ^

c'è scambio perché  $9 > 2$


3      2      9      5      10  
                                 ^

c'è scambio perché  $9 > 5$  . . .

# ORDINAMENTO DI ARRAY/IMPLEMENT. BUBBLE

richiede un annidamento di cicli . . .

```
void bubble_sort(int A[], int length){  
    int i, j ;  
    for (i = 0 ; i < length ; i = i+1){  
        for (j = 0 ; j < length-1-i ; j = j+1)  
            if (A[j] > A[j+1]) scambia(A[j], A[j+1]) ;  
        }  
    }
```



piccola ottimizzazione!  
length-1-i invece che  
length-1

# ALGORITMO DI RICERCA SU ARRAY ORDINATI

per cercare se un array  $A$  i cui **elementi sono ordinati** contiene o meno un elemento  $k$  si utilizza la **ricerca binaria** (simile al metodo utilizzato per cercare un numero nell'elenco telefonico)

- \* si accede all'elemento memorizzato a metà di  $A$  e si verifica se esso è uguale a  $k$
- \* se è uguale a  $k$  l'algoritmo termina e ritorna l'indice
- \* altrimenti si sceglie la metà appropriata di  $A$  e si ripete il passo 1
- \* l'algoritmo termina con  $-1$  se  $k$  non è presente

**osservazione:** il costo computazionale è  $\log_2 \text{length}$



# ALGORITMO DI RICERCA SU ARRAY ORDINATI

## implementazione:

1. si utilizzano due indici  $l$  (sta per left) e  $r$  (sta per right) che puntano alla porzione di array su cui si deve cercare la presenza di  $k$  ( $l \leq r$ )
2. all'inizio  $l=0$  e  $r = \text{length}-1$
3. si accede all'elemento a metà della porzione di array tra  $l$  e  $r$ 
  - \* dove si trova questo elemento? ad indice  $(l+r)/2$
4. se  $A[(l+r)/2] == k$  allora abbiamo trovato l'elemento
5. se  $A[(l+r)/2] > k$  allora bisogna cercare nella parte sinistra
  - \* si itera (si ritorna al passo 1) con  $r == (l+r)/2 - 1$
6. se  $A[(l+r)/2] < k$  allora bisogna cercare nella parte destra
  - \* si itera (si ritorna al passo 1) con  $l == (l+r)/2 + 1$

# ALGORITMO DI RICERCA SU ARRAY ORDINATI

```
int bin_search(int A[], int length, int k) {  
    bool found = false ;  
    int l = 0 ;  
    int r = length ;  
    int m ;  
    while (!found && (l < r)) {  
        m = (r + l) / 2 ;  
        if (A[m] == k) found = true ;  
        else if (A[m] > k) r = m ;  
        else l = m + 1 ;  
    }  
    if (found) return (m) ; else return (-1) ;  
}
```

# ESERCIZI

1. Definire una funzione che dato un array restituisce la posizione della seconda occorrenza del primo carattere che occorre almeno due volte, restituisce `-1` se nessun carattere occorre almeno due volte. (esame del 29/5/2013)
2. Scrivere un programma che implementa una pila utilizzando un array (definire le funzioni `is_empty`, `push` e `pop`; la funzione `push` ritorna `overflow` se l'array è pieno)
3. Scrivere un programma che implementa due pile utilizzando un solo array. In particolare, dato un array di lunghezza `L` (`L` è una costante del programma) definire le funzioni `push1`, `pop1`, `is_empty1` e le funzioni `push2`, `pop2`, `is_empty2` che implementano sull'array le note funzioni sulle pile. Le funzioni `push1` e `push2` restituiscono un errore (overflow) solamente se l'array è pieno. (esame del 15/6/2012)
4. Definire una funzione `void parola(char str[], int n, char dest[])` che prende come parametri un array `str` e un intero `n` e restituisce nel parametro `dest` la parola corrispondente all'ennesima parola dentro `str`. Si assuma che una parola sia una qualunque sequenza di caratteri diversi da spazio e che le parole siano separate tra loro da uno o più spazi. In caso di errore la funzione restituisce la parola vuota. (esame del 17/2/2012)

domani **CATCH-UP LECTURE!**  
solo per gli **UNSKILLED PROGRAMMERS**