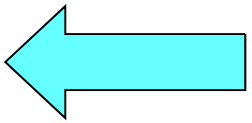
15. Array statici

Andrea Marongiu
(andrea.marongiu@unimore.it)
Paolo Valente

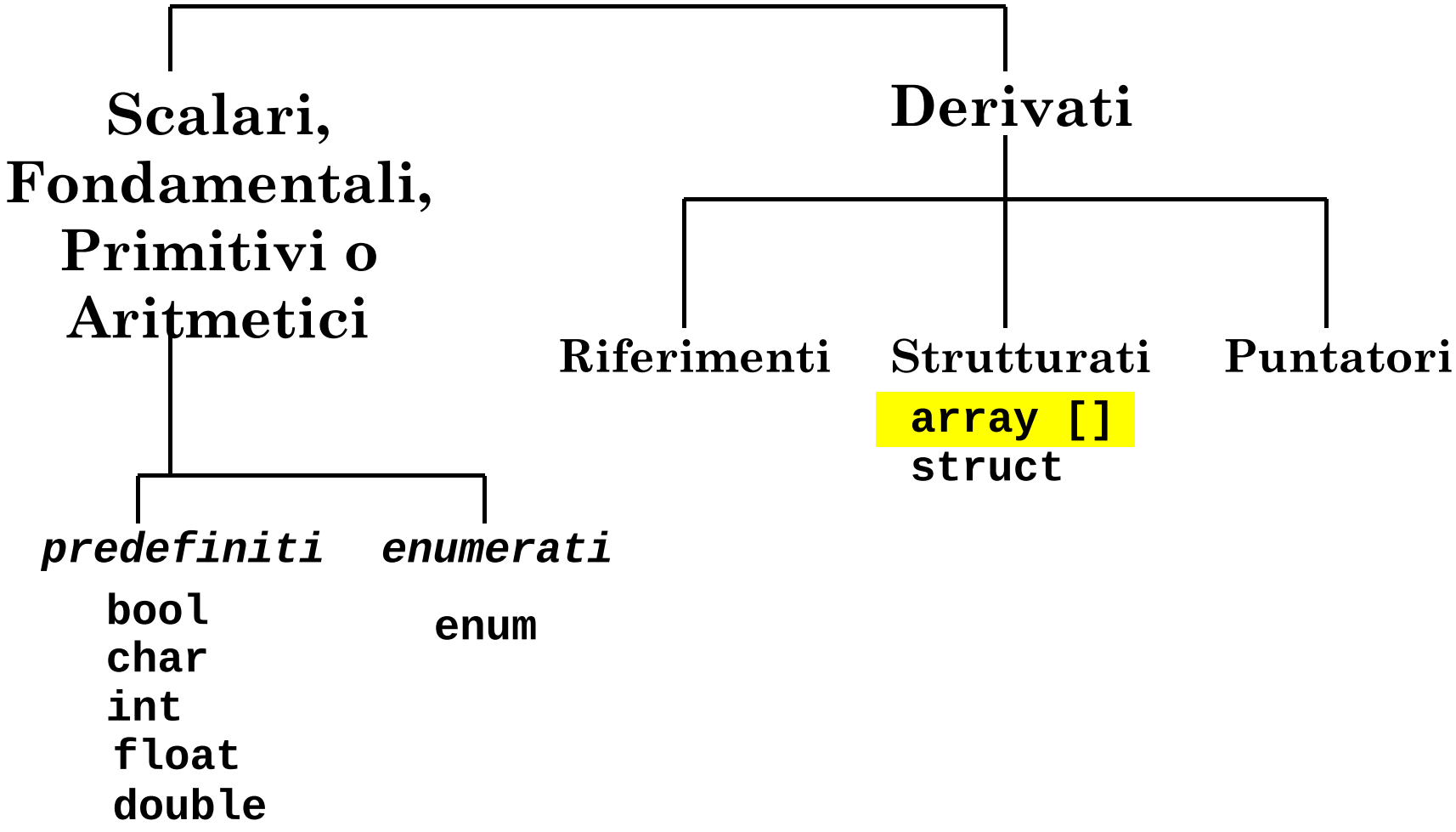
UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Contenuto lezione



- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array

Tipi di dato


Scalari, Fondamentali, Primitivi o Aritmetici
Derivati
predefiniti
enumerati
bool
char
int
float
double
enum
Riferimenti
Strutturati
Puntatori
array []
struct

Problema

- Scrivere un programma che
  - Legga da *stdin* 20 valori reali
  - Calcoli la media di tali valori
  - Ristampi solo i valori maggiori della media

Soluzione

- Una possibile soluzione basata sulle conoscenze acquisite finora (ossia le uniche conoscenze che possiamo utilizzare) è la seguente
  - Definire 20 variabili di tipo reale
  - Per ciascuna variabile
    - Leggere da stdin il valore della variabile
  - Calcolare la media dei valori
  - Per ciascuna variabile
    - Stamparne il valore solo se superiore alla media
- Il livello di replicazione del codice è intollerabile
- La qualità del codice è bassissima

Proposta

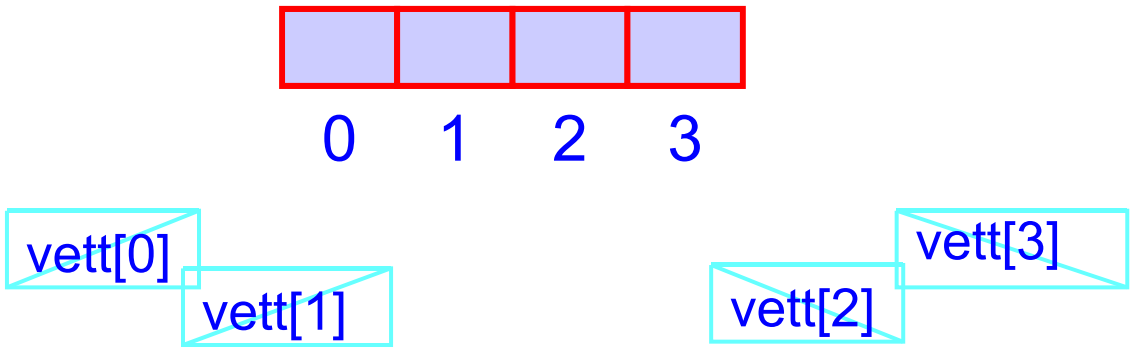
- Ci occorre una struttura dati che ci permetta di risolvere lo stesso problema con codice
  - Iterativo
  - Compatto
  - Senza replicazione o al più con un livello ragionevole di replicazione

Array

- Un *array* è una ennupla di *N* oggetti dello stesso tipo
  - Allocati in posizioni contigue in memoria
- Selezione elementi: mediante **indice**
  - Ciascun elemento dell'array è denotato mediante:
    - nome dell'array
    - seguito da un indice intero compreso fra 0 e
    - *N*-1, scritto tra parentesi quadre
- Esempio
  - Dato un array **`A`** di dimensione *N*
  - L'elemento *i-esimo* è denotato come **`A[i]`**, con
  - **0 <= i < *N***

Definizione

- SINTASSI della definizione di una variabile o di una costante con nome di tipo **array statico**:
- **[const]** *<tipo_elementi_array>* *<identificatore>* **[***<espr_costante>***] ;**
- SEMANTICA: alloca una sequenza contigua di elementi in memoria, tutti di tipo *<tipo_elementi_array>* ed in numero pari ad *<espr_costante>*
- Esempio: array statico di 4 elementi di tipo **int**
- **int vett[4] ;** // alloca spazio per 4 elementi **int** contigui



Dimensioni 1/2

- All'atto della definizione di un array le dimensioni devono essere stabilite mediante una espressione costante
  - Il valore deve essere quindi noto a tempo di scrittura del programma
  - Deve essere un numero naturale

Dimensioni 2/2

Corretto

```
const int
      NUM_ELEM = 500;

int vett[NUM_ELEM];
```

Scorretto

```
int num_elem ;
cin>>num_elem ;

int vett[num_elem];
```

- La seconda forma è **fuori dallo standard**
  - Alcuni compilatori la consentono
- Il programma risultante non sarebbe più portabile

Intervallo indice

- Contrariamente ad altri linguaggi, il C/C++ non consente di scegliere il valore iniziale dell'indice di un array
  - Parte sempre da 0
    - L'indice del primo elemento è quindi 0
  - Pertanto, un array di *N* elementi ha sempre, necessariamente, indici da 0 a *N*-1 (inclusi)

Esercizio 1

- Scrivere un programma che chieda all'utente di inserire un numero di valori interi prefissato (a tempo di scrittura del programma), li inserisca in un array e stampi l'array risultante.

- Un possibile output è il seguente:

```
Inserisci un intero positivo (elemento 1 / 3): 2
Inserisci un intero positivo (elemento 2 / 3): 5
Inserisci un intero positivo (elemento 3 / 3): 1
Ecco l'array immesso:
Elemento 1: 2
Elemento 2: 5
Elemento 3: 1
```

Esercizio 1

- SOLUZIONE

```
#include <iostream>
using namespace std;

int main() {
  const int elems = 5;
  int array[elems];

  for (int i = 0; i < elems; i++) {
    cout<<"\nInserisci un intero positivo (elemento "<<i<<" / “
        <<elems<<"): ";
    cin>>array[i];
  }

  cout<<"Ecco l'array immesso:"<<endl;

  for (i = 0; i < elems; i++)
    cout<<"\nElemento"<<i<<": "<<array[i];
  cout<<endl;

  return 1;
}
```

Esercizio 2

- Scrivere un programma che definisca un array di interi di lunghezza prefissata (a tempo di scrittura del programma), lo inizializzi con valori casuali e lo stampi.

Esercizio 2

- la funzione di libreria rand() genera un numero casuale intero compreso tra 0 e RAND_MAX (prefissata, e quindi non modificabile).

- Essa permette di generare una sequenza di numeri pseudocasuali: fissato il primo valore della sequenza (chiamato seme), è fissata tutta la sequenza di valori che saranno generati nelle successive invocazioni della funzione rand().

- Infatti i numeri sono generati mediante una certa funzione f(x) che, dato l'ultimo valore x generato, genera il prossimo valore random.

Esercizio 2

- Eempio: supponiamo di aver dato come seme il valore 1. Allora il primo valore generato dalla funzione rand() sarà 41 (se la funzione rand() sulla vostra macchina e' basata sullo stesso algoritmo della funzione sulla mia macchina). La prossima volta che verra' invocata la funzione rand() si otterra' il valore f(41), che e' pari a 18647. Alla successiva invocazione si otterra' il valore di f(18647), che e' pari a 6334, e cosi' via.

Esercizio 2

- Per ottenere sequenze diverse, bisogna cambiare il valore del seme con la funzione srand(n), ove n e' il nuovo valore che si vuol dare al seme.

- In definitiva, la funzione srand va invocata una sola volta, per decidere il seme, mentre la funzione rand va invocata ogni volta che si vuole ottenere il prossimo valore.

- Per utilizzare le funzioni rand() ed srand(n) (e la costante RAND_MAX), bisogna includere il file di intestazione "cstdlib".

Esercizio 2

- Per ottenere sequenze 'quasi' completamente casuali, si può tentare di dare al seme un valore casuale.
- A questo scopo si può ad esempio sfruttare il valore di ritorno della funzione time (per usare tale funzione bisogna includere ctime), che è uguale al numero di secondi trascorsi dal 1 gennaio, 1970, GMT.
- La funzione time prende in ingresso un valore numerico, che poniamo uguale a 0.
- Si rimanda chi fosse interessato ad ulteriori dettagli alla documentazione su questa funzione di libreria.

Esercizio 2

- `#include <iostream>`
- `#include <cstdlib>`
- `#include <ctime>`

- `srand (n)` *`// n è il nuovo valore del seme`*
- `rand ()` *`// ritorna un numero casuale`*
- `time (0)` *`// ritorna i secondi dal 1/1/1970`*

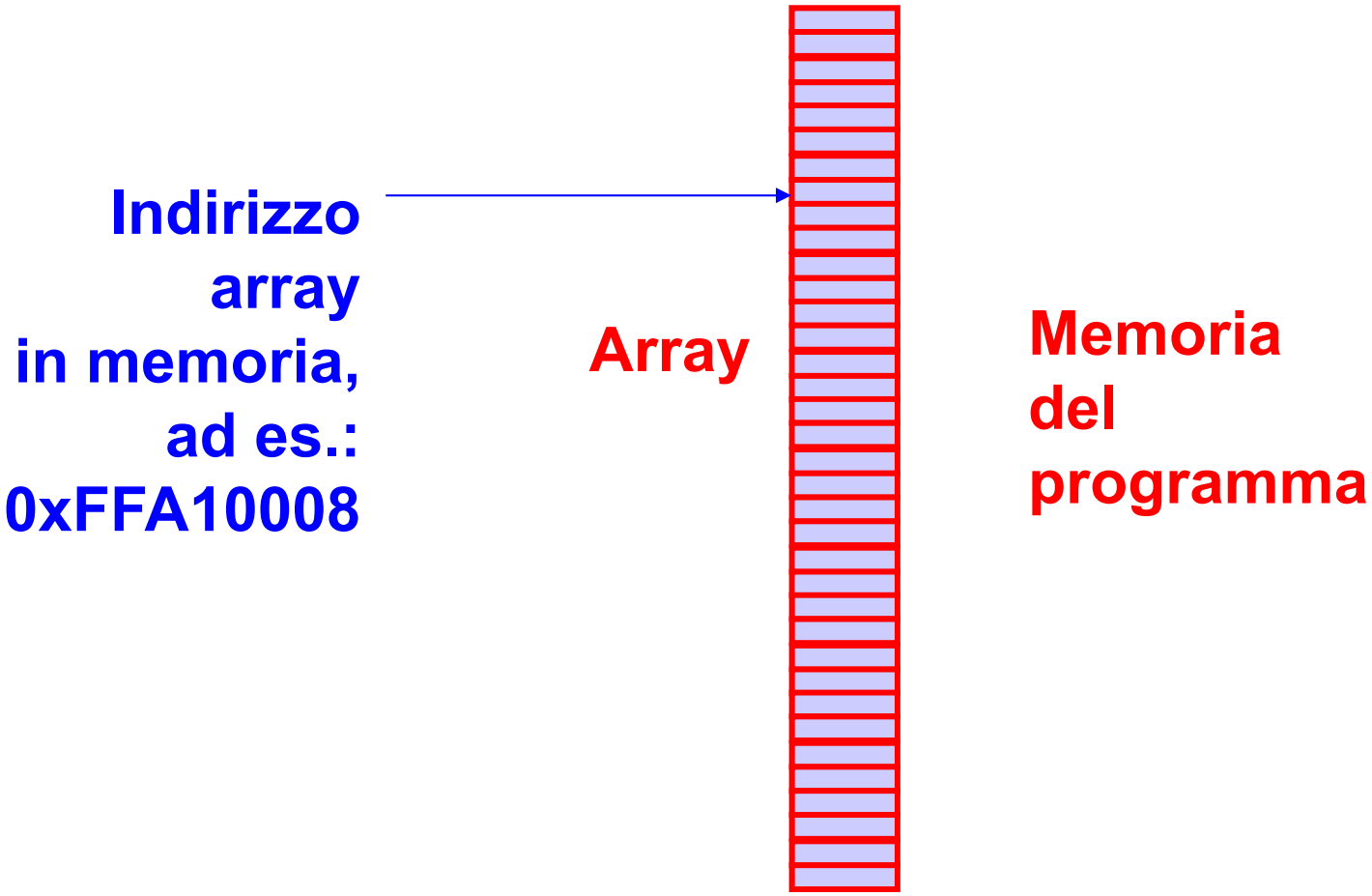
Esercizio 2

- SOLUZIONE

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  const int elems = 5;
  int array[elems];

  srand (time (0));

  for (int i = 0; i < elems; i++)
    array[i] = rand ();

  cout<<"Ecco l'array immesso:"<<endl;

  for (i = 0; i < elems; i++)
    cout<<"\nElemento"<<i<<": "<<array[i];
  cout<<endl;

  return 1;
}
```

Array in memoria 1/3

- All'atto delle definizione di un array, viene allocato spazio nella memoria del programma per una sequenza di celle

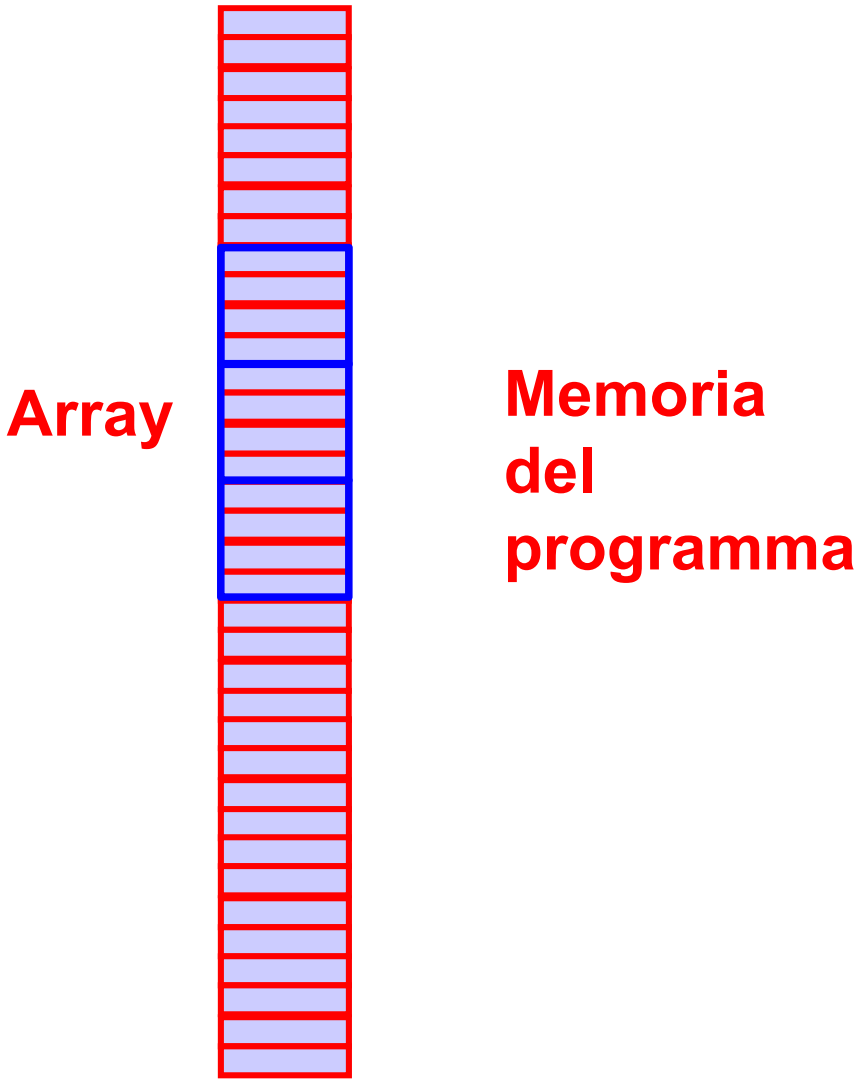


Array in memoria 2/3

- Nel caso più semplice gli elementi dell'array sono di un tipo di dato che può essere memorizzato su una sola cella di memoria
  - Accade tipicamente per il tipo **char**
- Se le cose non stanno così, l'array è di fatto una sequenza di sottosequenze di celle
  - Ogni sottosequenza è utilizzata per memorizzare uno degli elementi dell'array
- Ad esempio, se il tipo degli elementi è **int** e tale tipo occupa 4 byte, l'array è una sequenza di sottosequenze da 4 byte ciascuna

Array in memoria 3/3

```
int a[3] ;
```



Assegnamento tra array

- La sintassi del C/C++ non permette di utilizzare l'operatore di assegnamento per copiare il contenuto di un intero array all'interno di un altro array

Esempio:

```
int a[10], b[10] ;
…
a = b ; // VIETATO !!!!!!!!!!!!!!
```

- L'unica soluzione è assegnare (copiare) gli elementi uno alla volta
- Vedremo più avanti il motivo esatto per cui l'assegnamento tra array statici è vietato

Array e vettori

- Un array è un oggetto informatico che permette di implementare l'oggetto matematico vettore
  - Di fatto un array statico di *N* elementi corrisponde per definizione proprio ad un **vettore** statico **di *N* elementi**
  - Come vedremo a breve, con un array statico si può però anche implementare un **vettore dinamico**, ossia un vettore con un numero di elementi che può variare durante l'esecuzione del programma ...

Esercizio 3

- Dato un vettore di *N* interi, inizializzati da *stdin* o casualmente, si determini il valore massimo tra quelli memorizzati nel vettore e lo si stampi
- Cogliamo anche quest'occasione per effettuare i passi fondamentali assieme
- Iniziamo dall'idea ...

Idea

- Assumi, come tentativo, che il “massimo momentaneo” sia il primo elemento del vettore
- Poi, confronta via via il “massimo momentaneo” con ciascuno dei successivi elementi del vettore
  - Ogni volta che trovi un elemento del vettore maggiore del “massimo momentaneo” sostituisci il “massimo momentaneo” con quell’elemento del vettore
- Dopo aver controllato tutti gli elementi del vettore, il “massimo momentaneo” corrisponderà al massimo del vettore

Verso un algoritmo

- Per trasformare la precedente idea in un algoritmo bisogna definire in modo preciso la struttura dati ed i passi da effettuare
- Come è fatta la struttura dati?

Struttura dati

- Array che realizza il vettore
- Costante **`N`** contenente la dimensione dell'array
- Variabile ausiliaria **`massimo`** destinata a contenere il massimo alla fine dell'algoritmo
- Variabile contatore ausiliaria per scandire l'array

Algoritmo

- Assegno alla variabile **`massimo`** il valore del primo elemento del vettore (quello di indice 0)
- Poi, scandisco il vettore da 1 a **`N`**-1 confrontando **`massimo`** con ciascun elemento
- Se trovo un elemento del vettore maggiore di **`massimo`** sostituisco il valore di **`massimo`** con il valore di quell'elemento del vettore
- Dopo aver controllato tutti gli elementi del vettore, il massimo del vettore sarà contenuto nella variabile ausiliaria **`massimo`**

Esercizio 3

- SOLUZIONE

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  int massimo;
  const int N = 10 ;
  int vettore[N];

  for (int i=0; i<N; i++)
    cin>>vettore[i];

  massimo=vettore[0];
  for (int i=1; i<N; i++)
    if (vettore[i]>massimo)
      massimo=vettore[i];

  cout<<"Il massimo del vettore e' "<<massimo<<endl;
  return 1;
}
```

Esercizio 4

- Dato un vettore di *N* interi, inizializzati da *stdin* o casualmente, si determini il valore massimo e si stampi sia il massimo sia la posizione del vettore in cui tale massimo compare

- Servono due variabili **`massimo`** e **`pos_massimo`**, o ne basta una?

Esercizio 3

- SOLUZIONE

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  int posizione;
  const int N = 10 ;
  int vettore[N];

  for (int i=0; i<N; i++)
    cin>>vettore[i];

  posizione = 0;
  for (int i=1; i<N; i++)
    if (vettore[i]>massimo)
      posizione = i;

  cout<< "Il massimo è in posizione « << posizione
      << " e vale " << vettore[posizione] <<endl;
  return 1;
}
```

Domanda

- Dove è memorizzata implicitamente la dimensione di un array?

Lettura dimensione array

- In una zona nascosta della memoria non controllabile dal programmatore
- Si può poi risalire alle dimensioni dell'array mediante l'operatore sizeof
- Restituisce il numero di byte occupati dall'array
- Esempio:
- `int a[10] ;`
- `cout<<sizeof(a) ; // stampa 40 se gli int sono`
- `// memorizzati su 4 byte`

Mancanza controlli ed errori

- In ogni caso, nel linguaggio C/C++ **non è previsto nessun controllo della correttezza degli indici** (inferiore e superiore) nell'accesso agli elementi di un array
  - Per esempio, per un array così definito,
  - `int vettore[100];`
  - istruzioni del tipo
  - `vettore[105]=54;` `vettore[100]=32;`
  - verrebbero accettate dal compilatore senza segnalazione di errori
- Tali errori possono causare fallimenti in modo impredicibile a tempo di esecuzione (incluso corruzione della memoria del programma, come vedremo fra qualche slide)

Inizializzazione array 1/2

- Un array può essere inizializzato (solo) all'atto della sua definizione
- Notazione:

[const] *<tipo_elementi_array>* *<identificatore>* **[***<espr-costante>***] =**
{ *<espr1>*, *<espr2>*, ..., *<esprN>* **}**

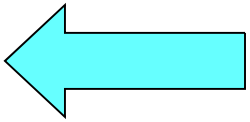
- Esempi:
- **`int a[3] = { 7, 3, 1 } ;`**
- **`char cv[4] = { 't', 'A', '8', '$'} ;`**

Inizializzazione array 2/2

- Se un array è inizializzato, l'indicazione della dimensione si può omettere. In tal caso la dimensione è dedotta dal numero di valori inizializzati. I precedenti esempi sono equivalenti a:
- `int a[] = { 7, 3, 1 } ;`
- `char cv[] = { 't', 'A', '8', '$'} ;`
- Se invece la dimensione è indicata esplicitamente
  - Non si possono inizializzare più elementi della dimensione dell'array
  - Se se ne inizializzano meno, i restanti contengono il valore 0 o valori casuali a seconda che l'oggetto sia globale o locale
- Un array costante va inizializzato obbligatoriamente

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array



Sintassi

<dichiarazione_parametro_formale_di_tipo_array> ::=
[const] *<tipo_elementi>* *<identificatore>***[ ]**

- Esempio definizione di una funzione con un parametro di tipo array:
- **`void fun(char c, int v[], …) { … }`**
- Se si tratta di un prototipo non è ovviamente necessario l'identificatore. Ad esempio:
- **`void fun(char c, int [], …)`**
-
- Per passare un array ad una funzione si passa semplicemente il **nome dell'array**
- Esempio invocazione con passaggio di un array:
-
```
int A[4];
fun('b', A, ...);
```

Tipologia passaggio 1/2

- Gli array sono **automaticamente passati per riferimento**
  - Se una funzione modifica l'array preso in ingresso (parametro formale), di fatto modifica l'array originario (parametro attuale)
    - Cosa si può fare evitare che questo possa accadere?

Tipologia passaggio 2/2

- Basta aggiungere il qualificatore **`const`** nella dichiarazione del parametro
-
- Esempio:
- **`void fun(const int v[], …) ;`**

Domanda

- Le informazioni sulle dimensioni di un array passato ad una funzione dove sono?

sizeof param. formali array

- Da nessuna parte!
- Inoltre, a differenza del caso dell'applicazione dell'operatore **sizeof** al nome di un *array*
  - Se si applica l'operatore **sizeof** ad un parametro formale di tipo *array*, restituisce il numero di byte necessari per memorizzare l'indirizzo dell'*array* in memoria
  - Vedremo fra qualche lezione come mai

Domanda

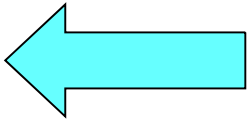
- Come può fare allora il codice di una funzione a conoscere le dimensioni di un *array* passato alla funzione?

Dimensioni array funzioni

- Come può fare allora una funzione a sapere le dimensioni dell'array che le è stato passato?
  - Se ne assicura il programmatore con due tipiche soluzioni
    - Parametro addizionale, contenente le dimensioni dell'array, passato alla funzione
    - Variabile/costante globale
      - Soluzione potenzialmente più efficiente (permette di passare un parametro in meno) ma affetta dai problemi degli oggetti globali già discussi

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array

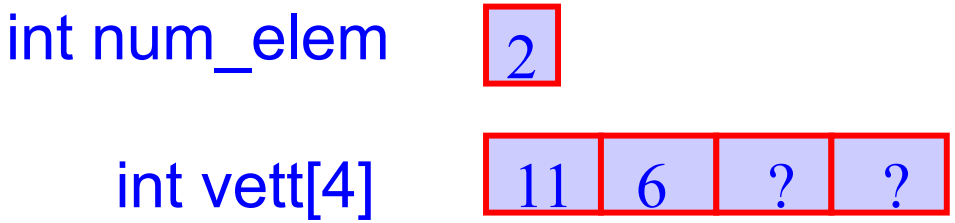


Vettori dinamici

- La dimensione *N* di un array statico non può essere determinata o modificata a tempo di esecuzione
- Al contrario, in molte applicazioni è necessario utilizzare vettori di dimensioni **variabili** o di dimensioni costanti ma **note solo a tempo di esecuzione**
- Uno dei modi più semplici per implementare un vettore di dimensioni variabili mediante un array è memorizzare il vettore in un array di dimensione pari alla dimensione massima del vettore
- Questo vuol dire però che in determinati momenti dell'esecuzione del programma **non tutte le celle** dell'array saranno **utilizzate**

Implementazioni 1/2

- Come gestire l'occupazione parziale?
- Vi sono due tipiche soluzioni:
  - La prima è memorizzare il numero di elementi validi, ossia il numero di elementi del vettore dinamico, in una ulteriore variabile
- Esempio di implementazione di un vettore dinamico di al più 4 interi, e che ha al momento lunghezza 2

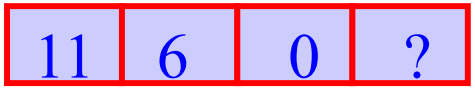


- I valori degli elementi dell'array successivi al secondo non hanno **nessuna importanza**, perché solo i primi due elementi sono validi

Implementazioni 2/2

- L'altra tipica soluzione è identificare il primo elemento non utilizzato con un valore che non appartiene all'insieme dei valori ammissibili per gli elementi di quel vettore. Tale elemento viene tipicamente detto **terminatore**. Ad esempio si può utilizzare
  - 0 per terminare un vettore di valori non nulli
  - -1 per terminare un vettore di valori positivi
- Esempio di implementazione di un vettore dinamico di al più 4 interi maggiori di zero, utilizzando il valore zero come terminatore

int vett[4]



- Il valore degli elementi dell'array successivi al terminatore non ha **nessuna importanza**, perché solo i primi due elementi sono validi

Esercizio

- Vediamo un esercizio propedeutico per i vettori dinamici implementati mediante array statici
  - In questo esercizio, il vettore dinamico non cambia dimensioni durante l'esecuzione del programma
  - Ma il numero di elementi che deve contenere si scopre solo a tempo di esecuzione del programma
    - Un semplice array statico non va quindi bene
- Data una serie di rilevazioni di al più 100 temperature espresse in gradi Kelvin da memorizzare in un vettore, si calcoli la media delle temperature effettivamente fornite e si ristampino solo le temperature al di sopra della media
- Che valore hanno gli elementi dell'array non inizializzati?

Idee

- Chiedi in input il numero di valori *M* effettivamente rilevati, e leggi tutti questi valori
- Somma tutti i valori inseriti
- La media cercata è data dalla somma precedente divisa per *M*

Algoritmo

- Assumi che vi possano essere fino a 100 temperature, ma chiedi in input il numero di valori *M* effettivamente rilevati
- Leggi *M* valori non negativi e inseriscili in un vettore nelle posizioni da 0 a *M*-1
- Somma tutti gli *M* valori del vettore
- La media cercata è data dalla somma precedente suddivisa per *M*
- Per tutti gli elementi del vettore stampa solo quelli di valore maggiore della media

Struttura dati

- Costante (int) per denotare la dimensione massima del vettore: **`max_M`**=100
- Array di reali di dimensione pari a **`max_M`**
- Variabile (int) per indicare il numero di valori di temperature effettivamente letti da input:
- **`M`** (**`0 < M <= max_M`**)
- Una variabile ausiliaria (int) da usare come contatore della scansione del vettore
- Infine, una variabile ausiliaria reale con il doppio ruolo di accumulatore delle somme parziali e contenitore della media

Programma

```
main()
{
  const int max_M = 100 ;        // dimensioni array statico
  double vett_temper[max_M] ;    // array statico
  int M;                         // num elementi array dinamico
  double media = 0. ;            // accumulo temperature e media

  do cin>>M;  while ((M<=0) || (M>max_M));

  for (int i = 0; i < M; i++) {
    do cin>>vett_temper[i];  while (vett_temper[i]<0);
    media += vett_temper[i];
  }

  media /= M ;

  cout<<"Media: "<<media<<endl<<endl ;
  cout<<«Valori superiori alla media: "<<endl ;

  for (int i = 0; i < M; i++)
    if (vett_temper[i] > media)
      cout<<vett_temper[i]<endl ;

  return 0;
}
```

Oggetto astratto

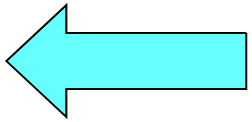
- Le due implementazioni di un vettore dinamico viste finora sono due esempi di implementazione di un oggetto astratto, il vettore dinamico, mediante uno o più oggetti concreti
  - nel primo caso un array più un contatore del numero di elementi validi
  - nel secondo caso un array
- Cogliamo l'occasione di questo esempio per evidenziare in modo concreto il processo di astrazione:
  - l'oggetto astratto vettore dinamico astrae dai dettagli su come è implementato
  - sia che sia implementato con un array più un contatore, o che sia implementato con solo un array, l'oggetto astratto vettore dinamico ha comunque le stesse identiche caratteristiche

Array e vettori astratti

- Riassumendo, mediante un array si può definire un oggetto astratto vettore, di cui si realizzano:
  - lunghezza variabile
    - mediante contatore numero di elementi validi o mediante elemento terminatore
  - assegnamento
    - mediante per esempio una funzione in cui si assegnano gli elementi uno ad uno
- Nella libreria standard di oggetti del C++ esiste anche l'oggetto astratto di tipo vettore (chiamato *vector*) che fornisce già queste e molte altre operazioni
  - Non useremo tale oggetto astratto in questo corso, ma implementeremo i vettori da noi, con le due tecniche appena viste

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array



Esercizio

- E se volessimo realizzare una variante del precedente esercizio in cui l'utente non comunica neanche il numero *M* di temperature da memorizzare, ma bensì segnala la fine della lettura inserendo un valore negativo?
- Supponiamo inoltre di implementare il vettore utilizzando un valore terminatore
  - Attenzione a cosa accade nel caso in cui si inseriscono *max_M* elementi !!!
- Pensateci un po' sopra, dopodiché valutiamo la soluzione proposta nella prossima slide

Proposta programma

```
main()
{
  const int max_M = 100 ; double vett_temper[max_M] ;


  for (int i = 0; i < max_M; i++) {
    cin>>vett_temper[i];
    if (vett_temper[i] < 0) {
     vett_temper[i] = -1 ; // inseriamo il terminatore
       break ;
    }
  }


  double media = 0. ;
  // in uscita dal ciclo conterra' il num di valori letti
  int num_val_letti ;
  for (num_val_letti = 0; vett_temper[num_val_letti] >= 0; num_val_letti++)
    media += vett_temper[num_val_letti];


  media /= num_val_letti ;


  cout<<"Media: "<<media<<endl<<endl ;
```

IL PROGRAMMA E' CORRETTO?

Commento aggiuntivo

```
main()
{
  const int max_M = 100 ; double vett_temper[max_M] ;


  for (int i = 0; i < max_M; i++) {
    cin>>vett_temper[i];
    if (vett_temper[i] < 0) {
     vett_temper[i] = -1 ; // inseriamo il terminatore
       break ;
    }
  } // nota: se inseriamo max_M valori non vi sarà alcun terminatore!


  double media = 0. ;
  // in uscita dal ciclo conterra' il num di valori letti
  int num_val_letti ;
  for (num_val_letti = 0; vett_temper[num_val_letti] >= 0; num_val_letti++)
    media += vett_temper[num_val_letti];


  media /= num_val_letti ;


  cout<<"Media: "<<media<<endl<<endl ;
```

IL PROGRAMMA E' CORRETTO?

Risposta

- Errore:
  - Non si controlla di essere sempre dentro l'array nella fase di calcolo della somma delle temperature
  - Se nell'array fossero stati inseriti *max_M* elementi, allora non vi sarebbe alcun terminatore, e si rischierebbe poi di leggere fuori dall'array
    - Errore logico
    - Possibile terminazione forzata del programma, oppure lettura di valori casuali
      - Come mai terminazione forzata o lettura di valori casuali?

- ***La risposta nella prossima lezione***