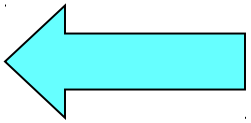
Lezione 13

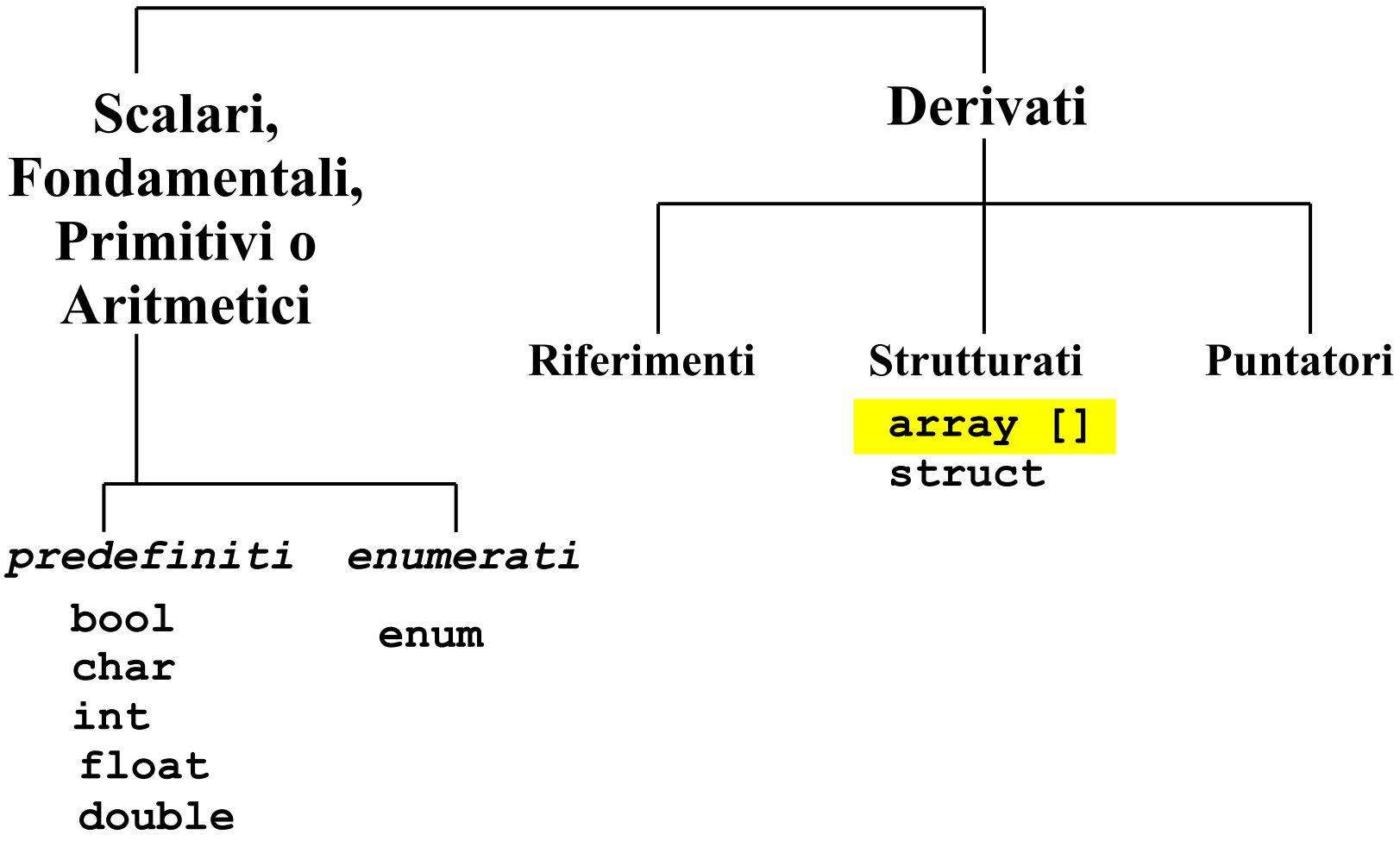
Array statici

Contenuto lezione

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



Tipi di dato


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

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***

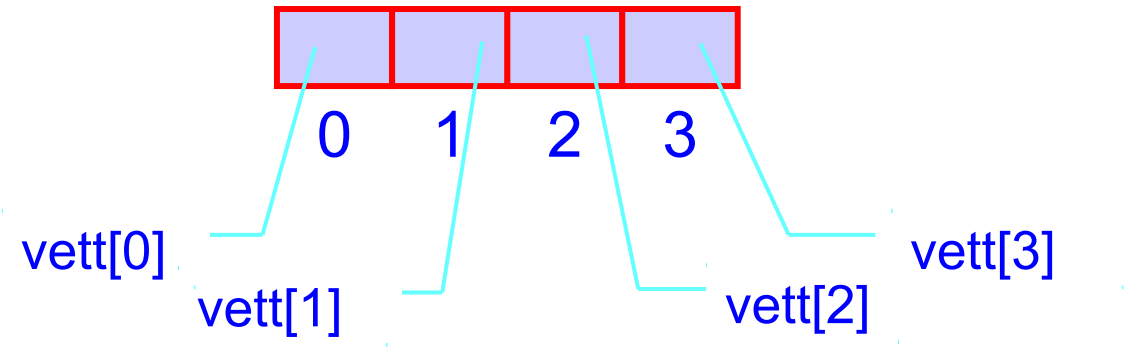
Definizione

- 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

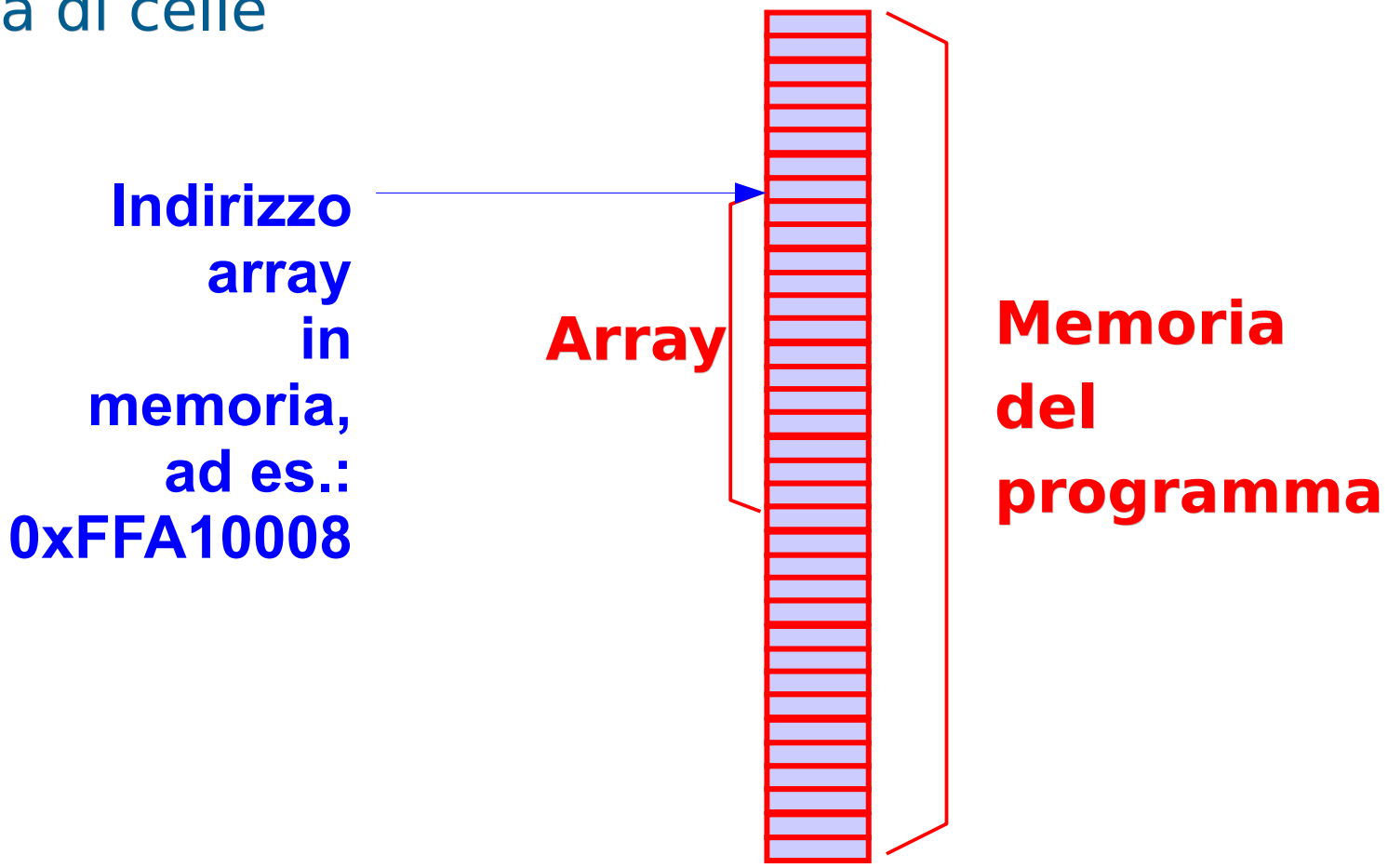
- Contrariamente ad altri linguaggi, il C/C++ non consente di scegliere il valore iniziale dell'indice di un array
  - Parte sempre da 0
    - L'indice del primo elemento è quindi 0
  - Pertanto, un array di *N* elementi ha sempre, necessariamente, indici da 0 a *N*-1 (inclusi)

Esercizi

- Svolgere i seguenti esercizi dell'ottava esercitazione:
  - *ins_stampa_array.cc*
  - Per casa
    - *array_casuali.cc*

Array in memoria 1/3

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

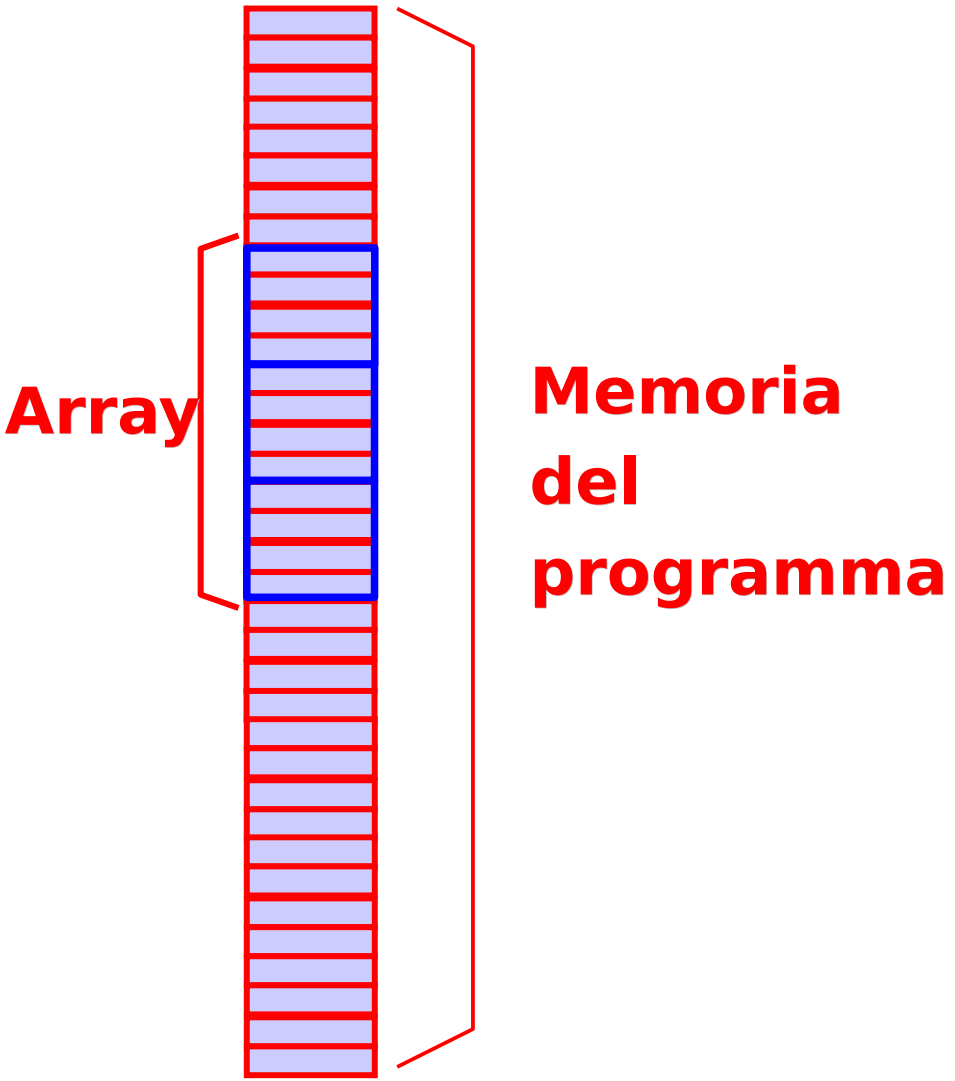


Array in memoria 2/3

- 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

- 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`**

Programma

- *max_elem.cc* dell'ottava esercitazione

Variante per casa

- 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

Proposta 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 ausiliaria **`pos_massimo`** destinata a contenere la posizione dell'elemento di valore massimo alla fine dell'algoritmo
- Variabile contatore ausiliaria per scandire l'array

Programma

- *max_pos_elem.cc*
- Si può fare meglio?
  - Servono due variabili **`massimo`** e **`pos_massimo`**, o ne basta una?
  - *max_pos_elem2.cc*

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

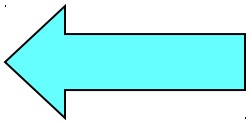
- Se un array è inizializzato, l'indicazione della dimensione si può omettere. In tal caso la dimensione è dedotta dal numero di valori inizializzati. I precedenti esempi sono equivalenti a:

```
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

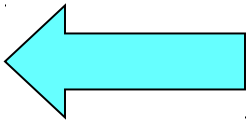
- Come può fare allora una funzione a sapere le dimensioni dell'array che le è stato passato?
  - Se ne assicura il programmatore con due tipiche soluzioni
    - Parametro addizionale, contenente le dimensioni dell'array, passato alla funzione
    - Variabile/costante globale
      - Soluzione potenzialmente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

Esercizio

- Svolgere l'ottava esercitazione fino all'esercizio *calcola_somma.cc*

Contenuto lezione

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

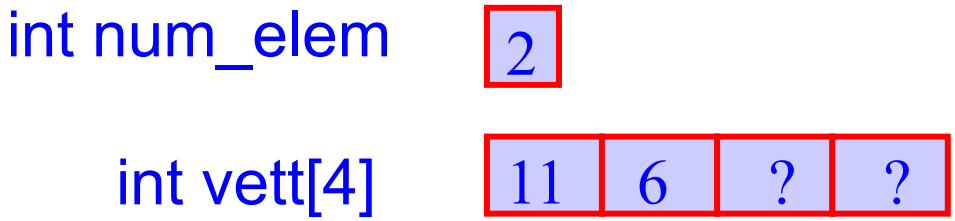


Vettori dinamici

- 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

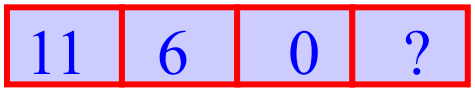


- I valori degli elementi dell'array successivi al secondo non hanno **nessuna importanza**, perché solo i primi due elementi sono validi

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

- *media_temp.cc*

Oggetto astratto

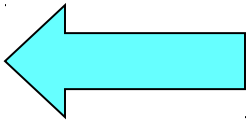
- Le due implementazioni di un vettore dinamico viste finora sono due esempi di implementazione di un oggetto astratto, il vettore dinamico, mediante uno o più oggetti concreti
  - nel primo caso un array più un contatore del numero di elementi validi
  - nel secondo caso un array
- Cogliamo l'occasione 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 è nelle seguenti slide

Contenuto memoria 1/3

- Sappiamo già che la memoria di un programma è utilizzata per memorizzare le variabili e le costanti con nome definite nel programma (e quelle allocate dinamicamente, come vedremo nelle prossime lezioni)
- Ma non solo, la memoria contiene anche delle informazioni non manipolabili direttamente dal programmatore
  - Codice delle funzioni (tradotto in linguaggio macchina)
  - Codice e strutture dati aggiuntivi, di supporto all'esecuzione del programma stesso
- In merito alle parti aggiuntive, senza entrare nei dettagli consideriamo solo che, quando il compilatore traduce il programma in linguaggio macchina, non si limita a creare solo il codice macchina che esegue ciascuna delle istruzioni esplicitamente inserite nel codice sorgente

Contenuto memoria 2/3

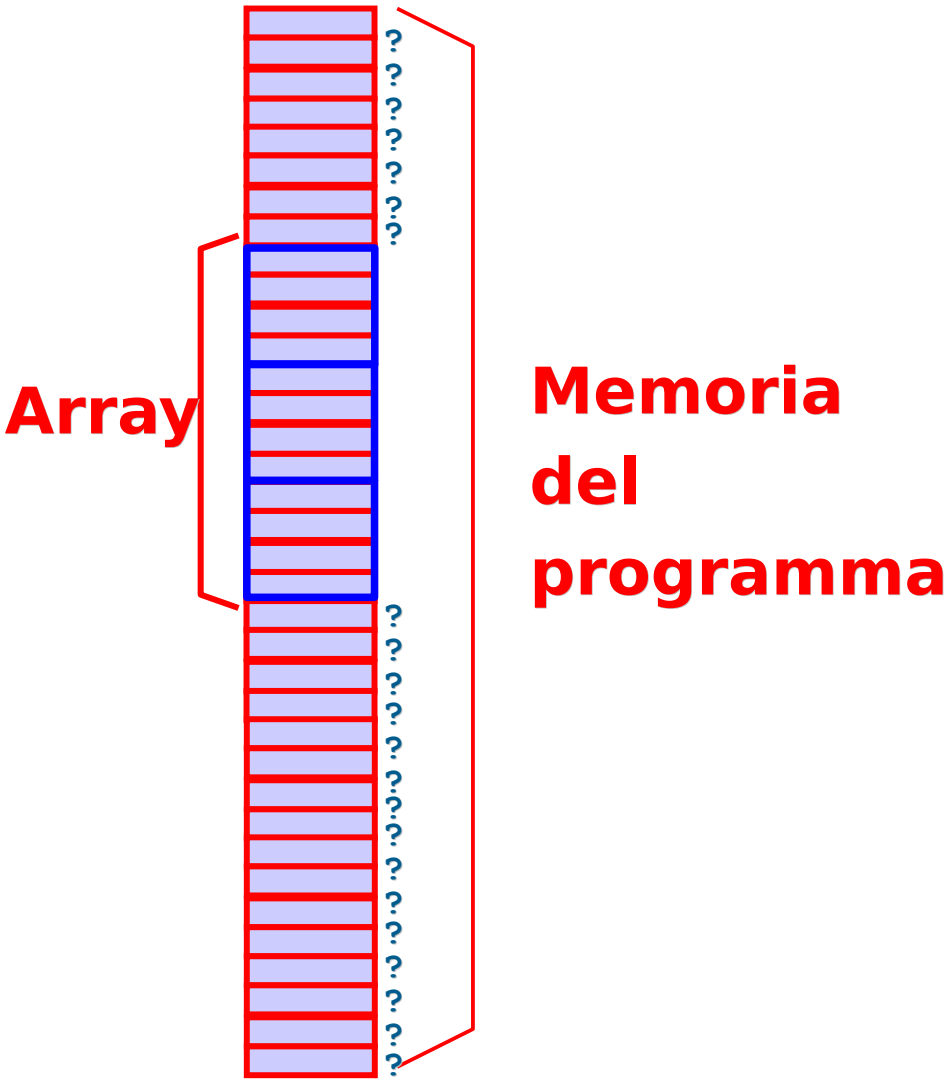
- Il compilatore di fatto aggiunge codice ulteriore, di cui vedremo qualche dettaglio in alcune delle prossime lezioni
- Tale codice effettua operazioni di supporto all'esecuzione del programma stesso (e che non sono e non possono essere esplicitamente inserite nel codice sorgente in base alle regole del linguaggio)
  - Inizializzare i parametri formali con i parametri attuali all'atto della chiamata di una funzione
  - Allocare spazio in memoria quando viene eseguita la definizione di una variabile/costante con nome
  - Deallocare spazio dalla memoria quando finisce il tempo di vita di una variabile/costante con nome
  - ...

Contenuto memoria 3/3

- Per realizzare alcune di queste operazioni il compilatore inserisce nella versione in linguaggio macchina del programma le strutture dati aggiuntive precedentemente menzionate
- In definitiva, se si accede a celle di memoria al di fuori di quelle dedicate ad oggetti correttamente definiti, si rischia di accedere a
  - Porzioni della memoria non ancora utilizzate per alcuno scopo, tipicamente contenenti valori casuali
  - Memoria occupata da altri oggetti definiti dal programmatore
  - Memoria occupata da codice o strutture dati aggiunte dal compilatore

Array in memoria

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



Errore gestione della memoria

- In ogni caso, accedere al di fuori delle celle di memoria dedicate ad oggetti correttamente definiti è un **errore di gestione della memoria**
  - Se l'accesso avviene in lettura si leggono di fatto valori casuali
  - Se l'accesso avviene in scrittura si rischia la cosiddetta corruzione della memoria del programma, ossia la sovrascrittura del contenuto di altri oggetti definiti nel programma o delle strutture dati aggiuntive precedentemente menzionate
    - Se si corrompono le strutture dati aggiuntive il comportamento del programma diventa **impredicibile**

Diritti di accesso alla memoria

- I processori moderni permettono di suddividere la memoria in porzioni distinte e stabilire quali operazioni si possono o non si possono effettuare su certe porzioni della memoria, nonché quando si ha diritto di effettuarle
  - Ad esempio, il codice di un programma viene tipicamente collocato in una porzione della memoria del programma stesso che è etichettata come non modificabile
  - Alcune porzioni della memoria gestite direttamente dal sistema operativo possono non essere accessibili, neanche in lettura, dal programma

Eccezione di accesso illegale

- Se un programma tenta di scrivere/leggere in una porzione della memoria in cui non ha il diritto di effettuare tale operazione, viene tipicamente generata dal processore una eccezione hardware di accesso illegale alla memoria
  - Le eccezioni hardware sono un meccanismo del processore che fa interrompere l'esecuzione dell'istruzione in corso e saltare immediatamente all'esecuzione di codice speciale dedicato alla gestione dell'eccezione
  - Tipicamente il codice di gestione delle eccezioni hardware di accesso illegale alla memoria termina immediatamente il processo (che quindi fallisce)

Segmentation Fault

- In ambiente UNIX l'eccezione scatenata da un accesso illegale alla memoria è tipicamente chiamata *Segmentation Fault*
- Come sappiamo, le eccezioni sono poi gestite da codice dedicato e stabilito dal sistema operativo nella fase di avvio
- Il codice di gestione dell'eccezione *Segmentation Fault* tipicamente termina forzatamente il processo che ha generato l'eccezione

Accesso fuori da un array

- In conclusione, accedere fuori da un array è un errore sotto due punti di vista:
  - Errore logico
    - Non ha senso accedere ad un elemento al di fuori dell'array stesso
  - Errore di gestione della memoria
    - Corruzione della memoria nel caso di accesso in scrittura

Versione corretta

- Tornando all'esercizio, la versione corretta del programma è in *media_temp2.cc* nella cartella dell'ottava esercitazione

Domanda

- Dato un vettore dinamico di interi rappresentato mediante
  - un *array*
  - un contatore **cont** del numero di elementi
- A quanto è uguale in ogni istante il numero di elementi del vettore dinamico?

Risposta ed altra domanda

- E' uguale ovviamente a **cont**
- Se, in un certo istante si volesse inserire un nuovo elemento in fondo al vettore
- A cosa sarebbe uguale l'indice dell'elemento dell'*array* in cui memorizzare il nuovo elemento?
- Cosa bisognerebbe fare dopo aver aggiunto il nuovo elemento?

Risposta

- L'indice del nuovo elemento sarebbe uguale a **cont**
- Dopo aver inserito il nuovo elemento **cont** va incrementato di 1

Doppia funzione contatore

- Il contatore del numero di elementi ha quindi una doppia funzione
  - Indica il numero di elementi attualmente presenti nel vettore dinamico
  - Fornisce l'indice dell'elemento dell'*array* in cui memorizzare il nuovo elemento nel caso di inserimento in coda

Esercizio

- Dato un vettore di al più *max_M*=5 elementi interi non nulli, si copino in un altro vettore solo gli elementi compresi tra 10 e 500
- Al termine, si stampi (solo) il numero di valori copiati nel secondo vettore
- Proviamo a risolverlo assieme …

Idee

- Ipotesi: implementiamo il vettore utilizzando il valore 0 come terminatore
- Bisognerà ovviamente scandire tutti gli elementi del primo vettore
  - Mentre si scandisce il vettore, si può copiare ogni valore accettabile nel nuovo vettore ed incrementare, per il secondo vettore, un contatore diverso da quello utilizzato per scandire il primo vettore
- Infine bisognerà stampare il valore finale del contatore relativo al secondo vettore

Struttura dati

- Costante (int) per denotare la dimensione massima dei due vettori: max_M
  - Probabilmente il secondo vettore avrà meno valori ammissibili del primo, ma perché il programma funzioni in tutti i casi, gli array utilizzati per rappresentare entrambi i vettori devono essere dimensionati per contenere max_M elementi
- Servono due array di double, ciascuno di dimensione pari a max_M
- Servono, poi, due variabili ausiliarie (int) come contatore della scansione del primo vettore e come contatore dei valori effettivamente copiati

Proposta programma

```
main()
{
    const int max_M = 5 ;
    int vett_uno[max_M] = { 100, 3, 200, 0, 300 } ;
    int vett_due[max_M];

    int conta=0;
    for (int i=0; vett_uno[i] != 0 && i<max_M; i++)
      if (vett_uno[i]>=10  &&  vett_uno[i]<=500) {
          vett_due[conta]=vett_uno[i];
          conta++;
      }
    if (conta < max_M) // altrimenti scriviamo "fuori"
        vett_due[conta] = 0; // inseriamo il terminatore
    cout<<"Sono stati copiati "<<conta<<" elementi"<<endl;
}
```

COSA STAMPA?
COME MAI?
E' CORRETTO?

Versione corretta

- C'è un errore logico: si accede all'elemento *i*-esimo **prima** di controllare di non essere fuori dall'array
- Versione corretta:
  - *copia_in_intervallo.cc*

Esercizio

- Dall'ottava esercitazione:
  - *array_pari.cc*
  - tutti i successivi esercizi

Nuova forma costrutto for

- Molto spesso, un ciclo che lavora su di un array scorre tutti gli elementi dell'array ed effettua una data operazione per ciascun ciclo
- Ad esempio, per stampare il contenuto di un array si scorrono gli elementi l'uno dopo l'altro e, ad ogni iterazione, si stampa il valore dell'elemento corrente
- Per tali tipi di cicli, a partire dallo standard C++11, è si può utilizzare anche una nuova forma del costrutto for, chiamato ***range-based for***

Sintassi

- Dato un qualsiasi array di elementi di un qualche tipo T
  - E chiamando per esempio A tale array
- Si può scrivere

```
for ([const] T &<identificatore>: A)
      <istruzione_composta>
```

ove *<identificatore>* è un qualsiasi identificatore, ed *<istruzione_composta>* è una qualsiasi istruzione composta

- Esempio

```
for (const int &x: A) cout<<x<<endl;
```

Semantica 1/2

- L'identificatore è visibile nell'istruzione composta che costituisce il corpo del **for**
- In particolare è il nome di un oggetto di tipo riferimento (vedi sotto)
- Il ciclo è eseguito tante volte quanti sono gli elementi dell'*array*
  - Alla prima iterazione, il riferimento è inizializzato col primo elemento dell'*array*
  - Alla seconda iterazione, il riferimento è inizializzato col secondo elemento dell'*array*

    ...
  - All'ultima iterazione il riferimento è inizializzato con l'ultimo elemento dell'*array*

Semantica 2/2

- In generale, all'*i*-esima iterazione il riferimento si riferisce all'*i*-esimo elemento dell'array
  - Permette quindi di leggere il valore di tale elemento o di aggiornare il valore di tale elemento
- Esempi

```
int A[10];
// inizializzazione array
for(int &x: A) x=1;
// stampa gli elementi dell'array A
for (const int &x: A) cout<<x<<endl;
// raddoppia il valore degli elementi
for (int &x: A) x *= 2;
```

Vantaggi

- Semplicità estrema della sintassi
  - Non è necessaria tra l'altro la notazione con le parentesi per accedere agli elementi
- Nessuna inizializzazione da scrivere prima della prima iterazione
- Nessuna condizione da scrivere
- Nessun indice da gestire

Svantaggi

- Il programma non si compila se il compilatore non supporta ed utilizza il nuovo standard
- Proprio a causa di questo svantaggio non si proporrà mai questo nuovo costrutto nelle soluzioni
- In quanto al *gcc*, per informazioni generali su quali versioni del compilatore supportano il nuovo standard, e su quali caratteristiche del nuovo standard sono effettivamente sopportate:

  http://gcc.gnu.org/projects/cxx0x.html

Funzioni per casa 1/2

- Implementare le seguenti funzioni (soluzioni non fornite):

1) void genera (int v[], int N, int TOT);

Creazione di un vettore di interi riempito con un numero casuale da 0 a TOT, per N elementi
Riceve: V, N, TOT - Restituisce: niente

2) void leggiord (int v[], int N);

Lettura di un vettore di interi letto da tastiera, per N elementi, valutando che il vettore sia inserito ordinatamente (cioè un dato è rifiutato se minore di quello inserito nella posizione precedente)

Riceve: V, N - Restituisce: niente

3) int pos (int v[], int N, int E);

Ricerca sequenziale di un elemento E in un vettore V di N elementi

Riceve: V, N, E - Restituisce: posizione dell'elemento (-1 se non esiste)

4) int ins (int v[], int N, int DIM, int E);

Inserimento di un elemento E nella posizione corretta in un vettore V ordinato di N elementi con al massimo DIM elementi, slittando a destra gli elementi successivi alla posizione di inserimento.

Riceve: V, N, DIM, E - Restituisce: il numero di elementi finale (N+1) se l'elemento è stato inserito (cioè se N<DIM), N altrimenti

Funzioni per casa 2/2

5) int canc (int v[], int N, int E);
Cancellazione di un elemento E in un vettore V ordinato di N elementi (slittando a sinistra gli elementi successivi alla posizione di cancellazione)
Riceve: V, N, E - Restituisce: il numero di elementi finale (N-1) se l'elemento è stato trovato e cancellato, N se l'elemento non è stato trovato, 0 se il vettore è vuoto

6) void stampa (int v[], int N);
Stampa di un vettore V di N elementi
Riceve: V, N - Restituisce: niente

7) int ord (int v[], int N);
Verifica che il vettore V sia ordinato
Riceve: V, N - Restituisce: 1 se v è ordinato, 0 altrimenti

8) void fusione(int v1[], int v2[], int v3[], int N);
Fonde i due vettori ordinati v1 e v2 di N elementi, nel vettore (vuoto) v3.
Riceve: V1, V2, N - Restituisce: nulla
Prima di chiamarla si richiami la funzione ord per accertarsi prima che il V1 e V2 sono ordinati . Attenzione che la dimensione di V3 deve essere il doppio di quella di V1 e V2.

Altri esercizi (senza soluzione)

1)Si scriva una funzione somma() che riceve come parametri 3 vettori v1, v2, v3 e la loro dimensione N. La funzione confronta il primo elemento di v1 e il primo elemento di v2 e copia il maggiore in v3 come primo elemento; confronta il secondo elemento di v1 e il secondo elemento di v2 e copia il maggiore in v3 come secondo elemento; ... e così via.
La funzione restituisce il numero di volte in cui un elemento di v1 è risultato maggiore dell'elemento di v2 con cui è stato confrontato.

Si scriva poi un programma che definisce tre vettori vett1, vett2, vett3, chiede a tastiera i valori dei due vettori vett1 e vett2, richiama la funzione sopra descritta e stampa il vettore vett3 risultante e il numero restituito dalla funzione.

2) Realizzare un funzione conta che riceve in ingresso un vettore V di interi ed un elemento E e restituisce quante volte E è ripetuto in V. Scrivere poi un main() che riempie un vettore leggendo dei valori da tastiera e fermandosi quando viene digitato il numero sentinella 999. Poi stampa a video il numero che ha il maggior numero di ripetizioni nel vettore. Esempio:

Input: 15 3 5 3 7 15 5 21 15 6 9 15 5 999
Output: "il numero più ripetuto è il 15 con 4 ripetizioni"

3) Scrivere una funzione contavolte che conta quante volte un elemento x è presente in un vettore v di n elementi. La funzione riceve come parametri x, v, n. Utilizzare contavolte dentro ad una seconda funzione creaunici, per costruire, a partire da un vettore v1, un secondo vettore v2 che contiene solo gli elementi unici di v1, cioè presenti una sola volta in v1. Esempio:

v1: 2 4 3 2 7 1 3 5 1 8 9
v2: 4 7 5 8 9