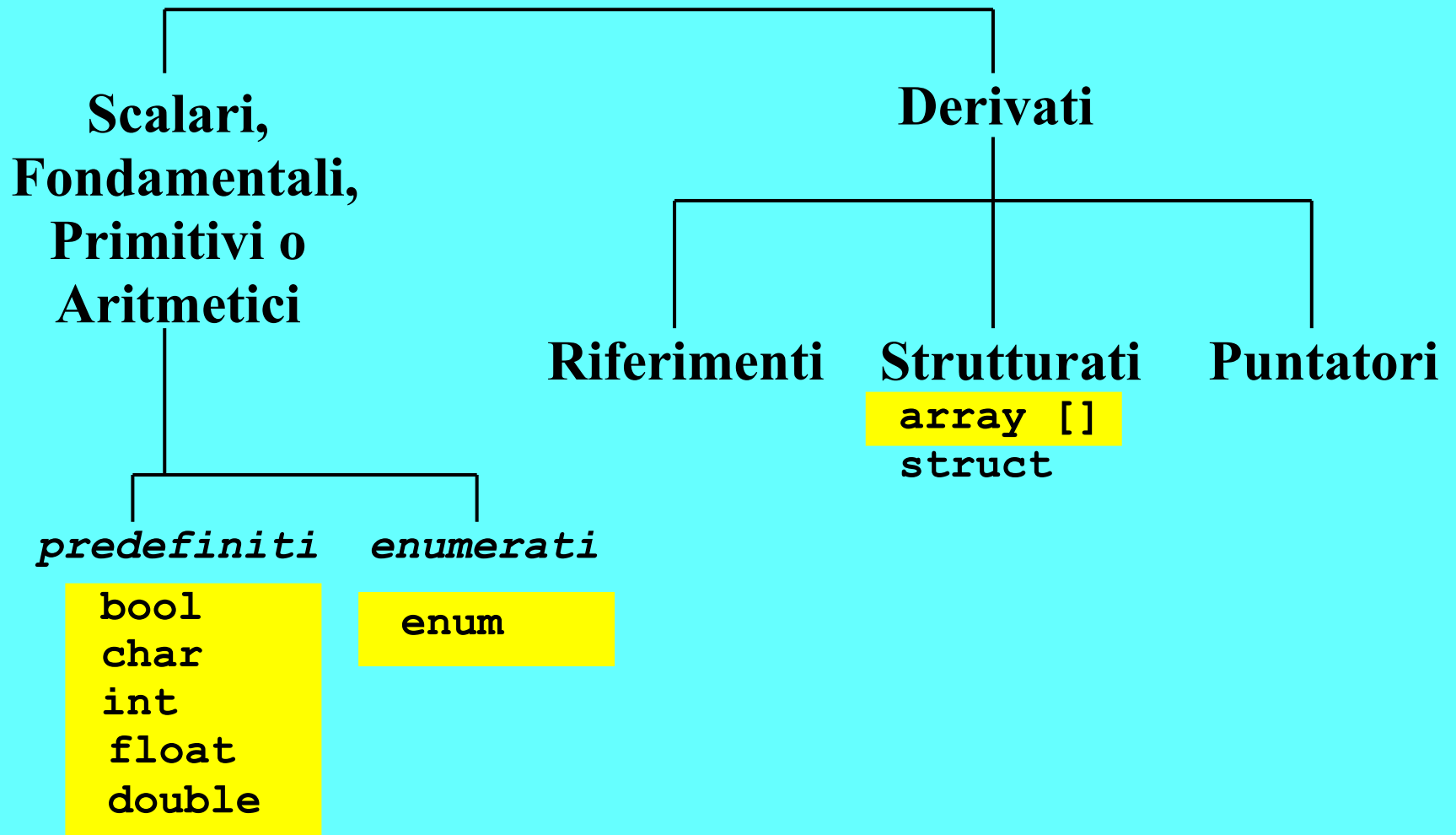


# **Tipi di dato strutturati**

**(Parte 1: Array, Matrici)**

# Tipi di dato



# Tipi di dato strutturati

- Il linguaggio C fornisce due *costruttori* fondamentali:
  - [ ]            (*array: vettori, matrici*)
  - struct        (*strutture*)

## ARRAY

Un array è una *ennupla* di  $N$  oggetti dello stesso tipo, ognuno identificato da un ***indice intero*** compreso fra 0 ed  $N-1$

## STRUTTURE

Una struttura è un insieme finito di oggetti non necessariamente dello stesso tipo, ognuno identificato da un ***nome*** distinto (in altri linguaggi è detta “record”)

# Tipo di dato Array

# Array

- Un array è una *ennupla* di  $N$  oggetti dello stesso tipo
  - Allocati in posizioni contigue in memoria
- **Selezione con indice:** ciascun elemento dell'array è identificato da:
  - nome dell'array
  - un ***indice intero*** compreso fra 0 e  $N-1$

## Esempio

Dato **A**, l'identificatore di un array di dimensione **N**

L'elemento *i*-esimo è denotato da **A[i]**, dove  **$0 \leq i < N$**

# Array (sintassi)

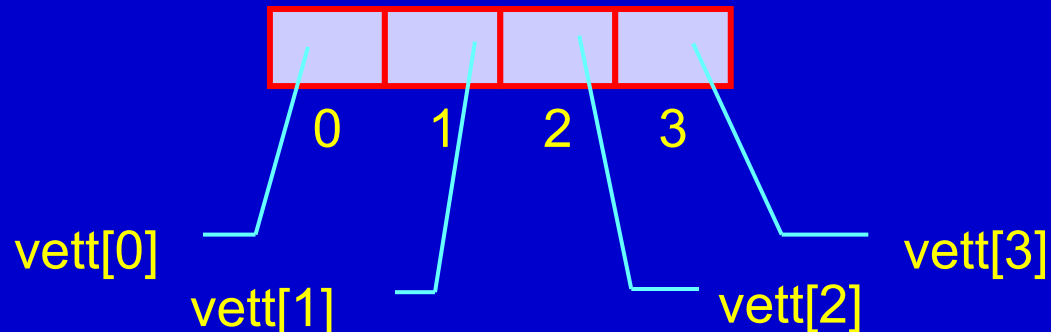
SINTASSI della *definizione* di una *variabile* di tipo vettore:

**<tipo>** **<identificatore>** [ **<espr-costante>** ] ;

**Importante!**

Es: Array di 4 elementi di tipo int

**int vett[4]** ; alloca spazio per 4 variabili int contigue



## FARE

```
const int max_vett 500;  
  
int vett2[max_vett];
```

## NON FARE

```
int max;  
cin>>max;  
  
int vett[max];
```

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

# NOTE (da ricordare)

- Contrariamente ad altri linguaggi, il C/C++ non consente di scegliere il valore iniziale dell'indice, che è sempre 0

Quindi, un array di  $N$  elementi ha sempre, necessariamente, indici da 0 a  $N-1$  (inclusi)

- Gli elementi di un array sono allocati in un spazio contiguo di memoria
- Possiamo utilizzare un array per implementare un vettore



# Esercizio 1 (*Specifica*)

- Dato un vettore di  $N$  interi, inizializzati da *stdin* o casualmente, si determini il valore massimo e lo si stampi.

# Esercizio 1 (*Idea*)

- Assumi, come tentativo, che il “massimo momentaneo” sia il primo elemento del vettore
- Poi, confronta via via il “massimo momentaneo” con tutti gli elementi del vettore
- Se 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

# Esercizio 1 (*Algoritmo*)

- Utilizzo una variabile ausiliaria, **max**, a cui assegno il primo elemento del vettore, assumendo che sia il “massimo momentaneo”
- Poi, scandisco il vettore da 1 a N-1 confrontando **max** con ciascun elemento
- Se trovo un elemento del vettore maggiore di **max** sostituisco il valore di **max** 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 **max**

# Esercizio 1 (*Rappresentazione informazioni*)

- Serve una costante (*int*) per denotare la dimensione massima del vettore: **N**
- Serve un vettore di *int* di dimensione pari a **N**:  
**vettore [ ]**
- Servono, poi, due variabili ausiliarie (*int*) come contatore della scansione del vettore (**i**) e come contenitore del massimo momentaneo (**max**)

# Esercizio 1 (*Programma*)

- *max\_elem.cc*

# Esercizio 1bis (*Specifica*)

- 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 questi compare.

## Esercizio 1bis (*Rappresentazione informazioni*)

- Serve una costante (*int*) per denotare la dimensione massima del vettore: **N**
- Serve un vettore di *int* di dimensione pari a **N**:  
**vettore [ ]**
- Servono, poi, *al più* tre variabili ausiliarie (*int*) come contatore della scansione del vettore (**i**), come contenitore del massimo momentaneo (**max**) e come indice della posizione del massimo (**pos\_max**)

# Esercizio 1bis (*Programma*)

- *max\_pos\_elem.cc*
- Si può fare meglio?
  - Servono due variabili *max* e *pos\_max*, o ne basta una?
  - *max\_pos\_elem2.cc*



# Rischi di errore

- Il C/C++ non consente di scegliere il valore iniziale dell'indice, che è sempre 0
- Dove è memorizzata implicitamente la dimensione di un array?
  - Da nessuna parte!
- I compilatori del linguaggio C/C++ non effettuano il controllo sul rispetto degli indici (inferiore e superiore) del vettore

Quindi, nel caso di dichiarazione: `int vettore[100];`  
istruzioni del tipo

`vettore[105]=54;`

`vettore[100]=32;`

verrebbero accettate dal compilatore senza segnalazione di errori, che si verificherebbero in modo impredicibile a tempo di esecuzione.

# Funzioni

**(Passaggio di vettori come parametri)**

# Passaggio di vettore come parametro (*come si fa*)

## Esempio

### **DICHIARAZIONE**

```
<tipo> fun(int v[ ], ...)  
    { ... }
```

### **CHIAMATA**

```
int A[4];  
fun(A, ...);
```

# Tipologia passaggio

- I vettori sono automaticamente passati per **riferimento**
- Le informazioni sulle dimensioni dove sono?
  - Da nessuna parte!
- Come fa una funzione a sapere le dimensioni del vettore che le è stato passato?
  - Parametro addizionale
  - Variabile/costante globale

# Evitare modifiche

- Se una funzione modifica i contenuti del vettore (*parametro formale*) che le è stato passato, le modifiche si ripercuotono sul vettore originario(*parametro attuale*)
- Se questo “rischio” o “opportunità” si vuole evitare, bisogna il qualificatore **const**:

**const int v[ ]**

**Es.**

**<tipo> fun(const int v[ ], ...)**

# Esempio

" *calcola\_somma.cc*

# Vettori parzialmente occupati

- Un vettore è un insieme finito di  $\text{max\_M}$  variabili dello stesso tipo, ognuna identificata da un *indice* compreso fra 0 e  $\text{max\_M}-1$ .
- La dimensione massima del vettore è specificata da un valore costante (e non può essere determinata o modificata a tempo di esecuzione)
  1. Il vettore in C/C++ non è un tutt'uno e non si debbono per forza *usare* tutte le celle disponibili
  2. **Spesso, la porzione di vettore realmente utilizzata *dipende dai dati di ingresso*.** La dimensione  $M$  della porzione del vettore realmente utilizzata ( $M \leq \text{max\_M}$ ) può essere stabilita a tempo di esecuzione

# Vettori parzialmente occupati

1. Come gestire l'occupazione parziale?
  2. Se il vettore è costituito da due parti, una prima parte totalmente occupata, seguita da un'altra completamente vuota, vi sono due tipiche soluzioni:
    1. Memorizzare il numero di elementi validi in una ulteriore variabile
    2. Identificare il primo elemento non utilizzato con un valore che non appartiene all'insieme dei valori ammissibili per quel vettore
- Es.,     **0** per un vettore di valori non nulli  
          **-1** per un vettore di valori positivi



## Esercizio 2 (*Specifica*)

- **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**

## Esercizio 2 (*Idea*)

- 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**

## Esercizio 2 (*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 i valori del vettore finché non raggiungi un valore negativo oppure hai raggiunto il numero massimo di elementi del vettore (bisogna considerare anche il caso che **M==100**)
- La media cercata è data dalla somma precedente suddivisa per **M**

## Esercizio 2 (*Rappresentazione informazioni*)

- Serve una costante (*int*) per denotare la dimensione massima del vettore: **max\_M=100** (°)
- Serve un vettore di *float* di dimensione pari a **max\_M**
- Serve una variabile (*int*) per indicare il numero di valori di temperature effettivamente letti da input: **M** ( $0 < M \leq \text{max\_M}$ )
- Servono, poi, una o due variabili ausiliarie (*float* e *int*), una non strettamente necessaria ove inserire temporaneamente i valori delle temperature lette (**temp**), ed una da usare come contatore della scansione del vettore (**i**)
- Serve, infine, una variabile ausiliaria (*float*) con il doppio ruolo di accumulatore delle somme parziali e di media (**val\_temper**)

(°) Il vettore dovrà essere dimensionato a 100 celle, perché le temperature possono essere fino a 100, anche se le celle realmente utilizzate potranno essere di meno

# Esercizio 2 (*Programma*)

- *media\_temp.cc*
- E se volessimo realizzarne una variante in cui si assegna un valore non valido al primo elemento non valido e lo si utilizza per determinare la fine della porzione di vettore contenente elementi validi?

# Esercizio 2 (*Programma bis*)

```
main()
{
    const int max_M = 100, M ;
    float vett_temper[max_M], val_temper=0.;

    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);

    vett_temper[M] = -1 ;

    for (int i=0; vett_temper[i] >= 0 ; i++)
        val_temper += vett_temper[i];

    val_temper /= M ;

    cout<<"Media: "<<val_temper;
}
```

**E' CORRETTO?**

# Esercizio 2 (*Programma ter*)

- Versione corretta:  
*media\_temp2.cc*

## Esercizio 3 (*Specifica*)

- Data una serie di rilevazioni di massimo 100 temperature espresse in gradi Kelvin memorizzata in un vettore (*la serie delle temperature lette termina quando viene inserito -1*), si calcoli la media delle temperature fornite.  
[NOTA: La sequenza di valori da leggere non è data, così come non è noto il numero totale dei valori da sommare]



# Inizializzazione di un array

- Un array può essere inizializzato (solo) all'atto delle sua definizione

- Notazione (per array di N elementi):  
= { **espr1**, **espr2**, ..., **esprN** }

- Esempi:

```
int a[3] = { 7, 3, 1 } ;
```

```
char cv[4] = { 't', 'A', '8', '$' } ;
```

# Inizializzazione di un array

- Se l'array è inizializzato, l'indicazione della dimensione non è necessaria. Le dimensioni sono dedotte dal numero di valori inizializzati. I precedenti esempi sono equivalenti a:  

```
int a[] = { 7, 3, 1 } ;  
char cv[] = { 't', 'A', '8', '$' } ;
```
- Se invece le dimensioni sono indicate esplicitamente
  - Non si possono inizializzare più elementi di quanti se ne dichiara contenere l'array
  - Se se ne inizializzano meno, i restanti possono contenere valori casuali
- Un array costante va inizializzato

# Array, vettori e dati astratti

- Mediante un array, si può definire un **oggetto astratto** vettore.
  - lunghezza variabile
    - mediante variabile esterna o valori nulli
  - assegnamento
    - mediante funzione in cui si assegnano gli elementi uno ad uno
- In C++ esistono oggetti astratti di tipo vettore che forniscono queste ed altre operazioni

# Assegnamento array

- Non è possibile assegnare il valore di un array ad un altro

Es.:

```
int a[10], b[10] ;
```

```
...
```

```
a = b ; // SBAGLIATO !!!!!!!!!!!!!!!
```

- L'unica soluzione è assegnare gli elementi uno alla volta
- In C++ esistono anche oggetti astratti di tipo vettore che permettono tutte le operazioni che ci si aspetterebbe

# Accesso fuori dall'array

- Errore logico
- Errore di gestione della memoria
  - Corruzione della memoria nel caso di accesso in scrittura

## Esercizio 4 (*Specifica*)

- Dato un vettore di al più  $\text{max\_M}=5$  elementi interi non nulli, si copino in un altro vettore solo gli elementi compresi tra 10 e 500
- Al termine, si stampi il numero di valori copiati nel secondo vettore

# Esercizio 4 (*Idea*)

- **Ipotesi**: Nel caso in cui il vettore iniziale contenga meno di 5 elementi, la sequenza di elementi da considerare è terminata da uno 0
- Scandisci tutti gli elementi del primo vettore
- Mentre scandisci il vettore, copia ogni valore accettabile nel nuovo vettore ed incrementa un contatore diverso da quello utilizzato per scandire il primo vettore
- Stampa il valore finale del contatore

# Esercizio 4 (*Algoritmo*)

- Inizializza il vettore a tempo di scrittura del programma
- Scandisci tutti gli elementi del primo vettore fino a trovare il valore 0 oppure finché non si sono letti 5 elementi
- Mentre scandisci il vettore, copia ogni valore compreso tra 10 e 500 nella posizione nel nuovo vettore ed incrementa un contatore, precedentemente inizializzato a 0. Questo stesso contatore è utilizzato per scandire il secondo vettore
- Stampa il valore finale del contatore



## Esercizio 4 (*Rappresentazione informazioni*)

- Serve una costante (*int*) per denotare la dimensione massima dei due vettori: **max\_M=5** (°)
  - Servono due vettori di **double** di dimensione pari a **max\_M**
  - Servono, poi, due variabili ausiliarie (*int*) come contatore della scansione del primo vettore (**i**) e come contatore dei valori effettivamente copiati (**conta**)
- (°) Probabilmente, il secondo vettore avrà meno valori ammissibili del primo, ma perché il programma funzioni in tutti i casi, entrambi i vettori devono essere dimensionati per contenere fino a 1000 elementi

# Esercizio 4 (*copia\_in\_intervallo.cc*)

```
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;
```

```
    cout<<"Sono stati copiati "<<conta<<" elementi"<<endl;
```

```
}
```

**COSA STAMPA?  
COME MAI?**

## Esercizio 5 (*Specifica*)

- Letto da input un valore intero, si verifichi se il valore dato è presente o meno in un vettore di 500 elementi interi.

## Esercizio 5 (*Idea*)

- Confronta il valore dato con tutti gli elementi del vettore
- Se trovi un elemento del vettore uguale al valore dato, stampa una scritta opportuna altrimenti stampa che l'elemento non è contenuto nel vettore

# Esercizio 5 (*Algoritmo - versione 1*)

- Leggi un valore da input e inizializza una variabile “logica” a “false”
- Scandisci tutto il vettore da 0 ad  $N-1$  confrontando ciascun elemento con il valore dato
- Se trovi un elemento del vettore uguale al valore dato, cambia il valore della variabile “logica”
- Stampa una scritta opportuna che indichi se l’elemento è contenuto o meno nel vettore

## Esercizio 5 (*Rappresentazione informazioni*)

- Serve una costante (*int*) per denotare la dimensione massima del vettore: **max\_N**
- Serve un vettore di *int* di dimensione pari a **max\_N**: **vettore[ ]**
- Servono, poi, due variabili ausiliarie (*int*) come contatore della scansione del vettore (**i**) e come contenitore del valore di input dato (**valore**)
- Serve, infine, una variabile “logica” (*int*) inizializzata a “false”: **trovato**

# Esercizio 5 (Programma - versione 1)

```
main()
```

```
{  
  const int max_N = 500;  
  int valore; bool trovato=false;  
  int vettore[max_N];
```

```
  cin>>valore;
```

*< il vettore viene inizializzato in qualche modo con 500 elementi inseriti dalla posizione 0 alla posizione 499 >*

```
  for (int i=0; i<max_N; i++)
```

```
    if (vettore[i]==valore)
```

```
      trovato=true;
```

**E se max\_N fosse uguale a 100000000?**

```
    if (trovato) cout<<"Trovato!"<<endl ;
```

```
    else cout<<"Non c'è"<<endl ;
```

```
  }
```

# Esercizio 5 (*Algoritmo - versione 2*)

- Leggi un valore da input e inizializza una variabile “logica” a “false”
- Scandisci tutto il vettore da 0 ad N-1 confrontando ciascun elemento con il valore dato
- **Non appena trovi un elemento del vettore uguale al valore dato, cambia il valore della variabile “logica” ed interrompi la scansione del vettore**
- Stampa una scritta opportuna che indichi se l’elemento è contenuto o meno nel vettore



# Esercizio 5 (*Programma - versione 2*)

```
main()
```

```
{  
  const int max_N = 500;  
  int valore; bool trovato=false;  
  int vettore[max_N];
```

```
  con>>valore;
```

*< il vettore viene inizializzato in qualche modo con 500 elementi inseriti dalla posizione 0 alla posizione 499 >*

```
  for (int i=0; i<max_N && !trovato ; i++)  
    if (vettore[i]==valore)  
      trovato=true;
```

oppure



```
  for (i=0; i<max_N; i++)  
    if (vettore[i]==valore)  
      { trovato=true;  
        break; }
```

```
  if (trovato) cout<<"Trovato!"<<endl ;  
  else cout<<"Non c'è"<<endl ;
```

```
}
```

# Problemi con vettori e funzioni

## 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

# Problemi con vettori e funzioni (*cont.*)

## 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 ricbin (int v[], int N, int E);**

Ricerca binaria di un elemento E in un vettore *ordinato* V di N elementi

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

Prima di chiamarla si richiami la funzione **ord** per accertarsi *prima* che il vettore sia ordinato

## 8) **int ord (int v[], int N);**

Verifica che il vettore V sia ordinato

*Riceve:* V, N - *Restituisce:* 1 se v è ordinato, 0 altrimenti

## 9) **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.

# Problemi con vettori e funzioni (*cont.*)

- 1) Si scriva una funzione **somma()** che riceve come parametri 3 vettori  $v_1$ ,  $v_2$ ,  $v_3$  e la loro dimensione  $N$ . La funzione confronta il primo elemento di  $v_1$  e il primo elemento di  $v_2$  e copia il maggiore in  $v_3$  come primo elemento; confronta il secondo elemento di  $v_1$  e il secondo elemento di  $v_2$  e copia il maggiore in  $v_3$  come secondo elemento; ... e così via. La funzione restituisce il numero di volte in cui un elemento di  $v_1$  è risultato maggiore dell'elemento di  $v_2$  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  $v_1$ , un secondo vettore  $v_2$  che contiene solo gli elementi unici di  $v_1$ , cioè presenti una sola volta in  $v_1$ . Esempio:

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

**v2:** 4 7 5 8 9

# ESAMI

- **PROVA DI AUTOVALUTAZIONE: 20 Novembre**  
(Argomenti: fino ad array inclusi)

- **ESAME (PRE-APPELLO)**

**Scritto:** intorno al 19 dicembre

**Prova di programmazione:** intorno al 19 dicembre

Solo per chi è preparato: non venite solo per provare l'esame! Chi consegna e va molto male, salta l'appello successivo

- **ESAMI**

**gennaio**

**febbraio**

# REGOLE

- **OBBLIGATORIO: ISCRIVERSI ALL'ESAME VIA E-MAIL**
- Non si possono utilizzare testi/appunti né alla prova scritta
- L'uso del manuale del linguaggio sarà concesso nella prova di programmazione (prova pratica) su richiesta
- **Si accede alla prova PRATICA SOLO DOPO AVER SUPERATO LA PROVA SCRITTA**
- Se si supera la prova scritta e non la prova pratica, è possibile rifare solo la prova pratica
- **II VOTO DELLA PROVA SCRITTA E' VALIDO FINO A QUANDO NON SI CONSEGNA UN'ALTRA PROVA SCRITTA (Non vale il max dei due voti!!!)**

# Tipo di dato “Matrice”

# Matrici

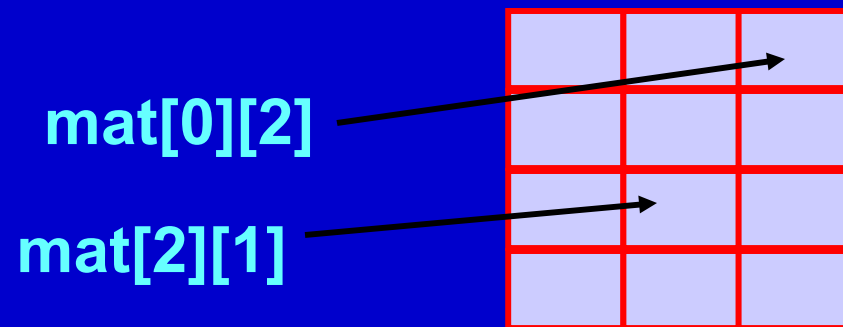
SINTASSI della *definizione* di una *variabile* di tipo matrice 2D:

```
<tipo> <identificatore> [ <costante> ] [ <costante> ] ;
```

Esempio:

```
double mat[4][3] ;
```

Matrice di 4x3 oggetti dello stesso tipo double,  
ognuno identificato/selezionata da un *indice intero di riga*  
compreso fra 0 e 3, ed un *indice intero di colonna* compreso  
fra 0 e 2 :





# Definizione Matrice

La *definizione* di una *variabile* di tipo matrice ricalca quella del vettore:

```
<tipo> <identificatore> [ dim_1 ] [ dim_2 ] ... [ dim_K ];
```

dove  $dim\_i$  denota il numero di elementi della dimensione *i*-esima.

- Di conseguenza, ci sono tanti indici quante sono le dimensioni e ciascun indice può assumere valori compresi fra 0 e  $(dim\_i - 1)$
- La selezione di un generico elemento di una matrice è denotata dal nome della matrice seguito dai valori degli indici racchiusi tra []

## Esercizio 6 (*Specifica*)

- **Data una matrice di dimensione  $100 \times 100$  di valori reali a precisione elevata, si calcoli la differenza tra la somma degli elementi della diagonale principale e della diagonale secondaria**

# Esempio: matrice 5x5

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

# Esercizio 6 (*Algoritmo*)

- IDEA:

Gli elementi della diagonale principale sono caratterizzati dagli  
indici:  $[i][i]$

Gli elementi della diagonale secondaria sono caratterizzati dagli  
indici:  $[i][M-1-i]$

Quindi, per scandire tutti gli elementi delle due diagonali è  
sufficiente un unico ciclo [e quindi un solo indice]

- Inizializzare due variabili a 0 e sommarvi tutti gli elementi della  
prima e seconda diagonale
- Stampa il valore finale della variabile che contiene la differenza  
tra le due variabili

## Esercizio 6 (*Rappresentazione informazioni*)

- Serve una costante (*int*) per denotare la dimensione della matrice: **M=100**
- Serve una matrice bidimensionale di double pari a **M x M**
- Serve un indice (*int*) per scandire la matrice: **i**
- Servono, infine, due variabili ausiliarie (*double*) per sommarvi i valori delle diagonali: **somma\_d1, somma\_d2**

# Esercizio 6 (*Programma*)

```
main()
```

```
{ const int M=100 ;
```

```
  double somma_d1=0., somma_d2=0.;
```

```
  double mat[M][M];
```

*< si ipotizza che la matrice venga inizializzata >*

```
  for (int i=0; i<M; i++) {
```

```
    somma_d1 = somma_d1+mat[i][i];
```

```
    somma_d2 = somma_d2+mat[i][M-1-i];
```

```
  }
```

```
  cout<<"Differenza valori " << somma_d1-somma_d2 << endl;
```

```
}
```

# Esercizio 7 (*Specifica*)

- **Data una matrice di dimensione 100x1000 di valori interi, si calcoli il numero complessivo di elementi positivi, negativi e nulli**

# Esercizio 7 (*Algoritmo*)

- Per scandire tutti gli elementi della matrice possiamo utilizzare due cicli innestati
- Inizializzare due variabili a 0 e sommarvi tutti gli elementi che risultano positivi e negativi
- Serve un'altra variabile per gli elementi nulli?
- Stampa il valore finale delle tre variabili



## Esercizio 7 (*Rappresentazione informazioni*)

- Servono due costanti (*int*) per denotare la dimensione massima delle righe e delle colonne della matrice: **max\_R=100**, **max\_C=1000**
- Serve una matrice bidimensionale di int pari a **max\_R \* max\_C**
- Servono, infine, due (tre ?) variabili ausiliarie (*int*) come contatori dei valori positivi e negativi (**positivi**, **negativi**)

# Esercizio 7 (Programma)

```
main()
{
    const int max_R = 100, max_C = 1000 ;
    int positivi=0, negativi=0 ;
    int mat[max_R][max_C];
    < si ipotizza che la matrice venga inizializzata >

    for (int i=0; i<max_R; i++)
        { for (int j=0; j<max_C; j++) {
            if (mat[i][j]>0) positivi++;
            else if (mat[i][j]<0) negativi++;
        }
    }

    cout<<"Valori positivi= "<<positivi<<", negativi = "<<negativi
        <<", nulli = "<< (max_R*max_C – positivi – negativi)<<endl ;
}
```

# Array di array

**Nel linguaggio C/C++, una matrice è a tutti gli effetti un array di array (*aggregati per riga*).**

- Ad esempio

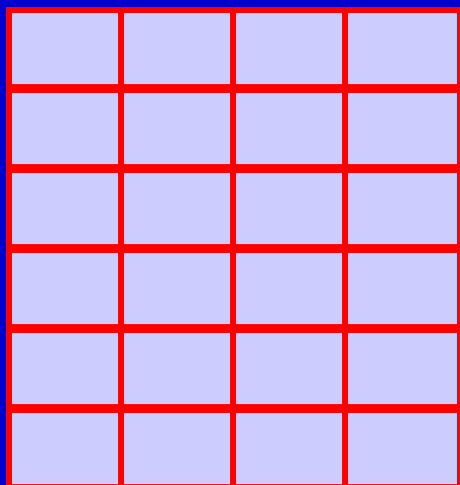
```
int mat[M][N] ;
```

definisce un array di M array da N elementi ciascuno

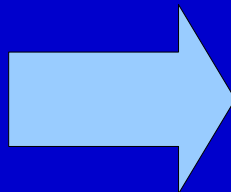
# Array di array

Per fare un esempio numerico, consideriamo

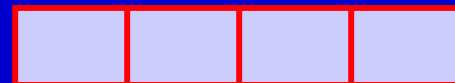
```
int mat[6][4] ;
```



**mat[6][4]**



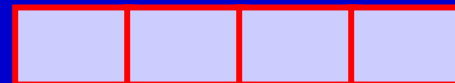
**mat[0]**



**mat[1]**



**mat[2]**



**mat[3]**



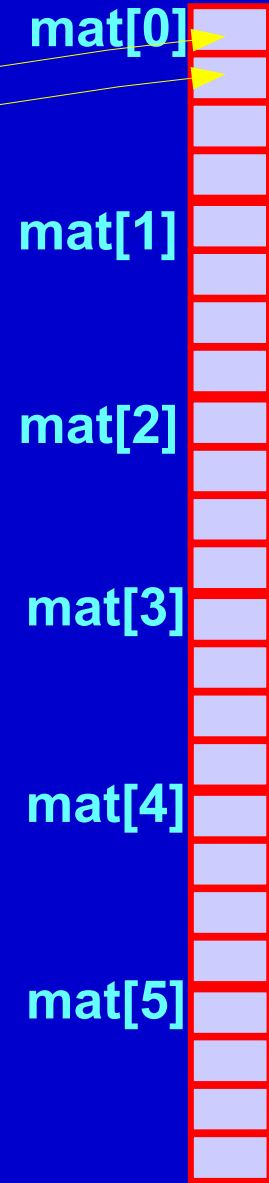
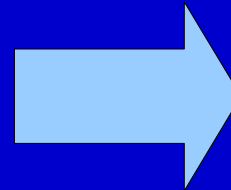
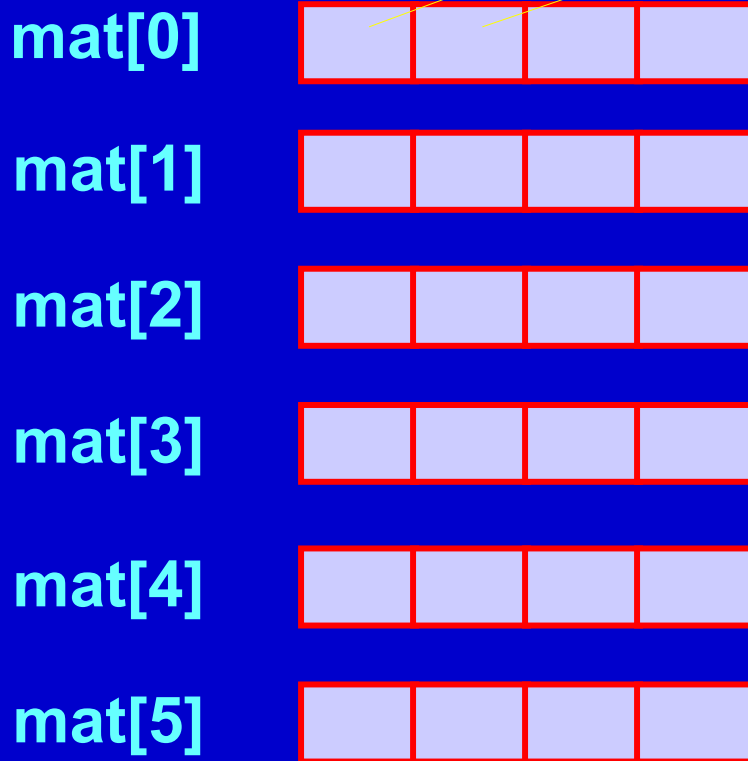
**mat[4]**



**mat[5]**



# Matrici in memoria



**Come conviene scandire tutti gli elementi di una matrice?**

# Passaggio righe di una matrice 2D

- Riassumendo:  
`int mat[M][N] ;`  
definisce un array di M array da N elementi ciascuno
- `mat[i]` con  $i = 0, 1, \dots, M - 1$  ?
  - è un array di N elementi
- Quindi data una matrice di N colonne, come si passa una delle righe ad una funzione che prende in ingresso un array lunghezza N
- Esercizio: *calcola\_somma\_righe.cc*
- A voi la generalizzazione per il passaggio di fette di matrici con più di due dimensioni

# Passaggio matrici

- Così come gli array monodimensionali, gli array di array sono passati per **riferimento**
- La dichiarazione/definizione di un parametro formale di tipo matrice bidimensionale è la seguente:  
**<tipo\_elementi> identificatore [][][<numero\_colonne>]**
  - Nessuna indicazione del numero di righe !!!
  - La funzione pertanto non conosce implicitamente il numero di righe della matrice
- Nell'invocazione della funzione, una matrice si passa scrivendone semplicemente il nome

# Passaggio matrici

```
const int num_col = 4 ;  
void fun(int mat[][num_col], int num_righe) ;  
main()  
{  
    const int M = 3 ;  
    int A[M][num_col] ;  
    fun(A, M) ;  
    ...  
}
```

Nel passaggio:

- Nessuna indicazione del numero di righe !!!
- La funzione non conosce implicitamente il numero di righe della matrice

Se si vuole che la matrice non sia modificata:

**const <tipo\_elementi> identificatore [][]<num\_col>**



# Inizializzazione array multidimensionali

- Generalizzazione sintassi degli array monodimensionali

- Esempio:

```
int mat[3][4] = { {2, 4, 1, 3},  
                 {5, 3, 4, 7},  
                 {2, 2, 1, 1} } ;
```

- Il numero di colonne deve essere specificato
- Il numero di righe può essere omesso
- Elementi non inizializzati possono avere valori casuali
- Non si possono inizializzare più elementi di quelli presenti

# Esercizio 8 (*Specifica*)

(Funzioni propedeutiche per)

## **BATTAGLIA NAVALE SEMPLIFICATA**

- Scrivere una funzione INSERT che riceva in input un numero di navi e le inserisca casualmente in una mappa di dimensioni 10x10  
[NOTA: - Si assuma che ciascuna nave occupi 1 cella  
- Si faccia attenzione a non posizionare le navi in celle coincidenti]
- Scrivere una funzione TIRO che riceva in input una coordinata (ovvero due elementi interi), e restituisca se il tiro ha colpito o meno una nave

# Esercizio 8 (*Specifica*)

## BATTAGLIA NAVALE SEMPLIFICATA

Realizzare un programma che, dopo aver fatto creare una mappa 10x10 con 12 navi da 1 cella in posizioni casuali, consenta ad un giocatore di “scoprire” tutte le posizioni delle navi avversarie.

La classifica dei record viene mantenuta rispetto al numero dei colpi necessari per scoprire tutte le navi nemiche.

[Integrazione: si visualizzi la mappa, con la posizione delle navi scoperte, i tiri effettuati andati a vuoto, e quelli andati a buon fine]

# Esercizio 9\* (*Specifica*)

(Funzione propedeutica per) **BATTAGLIA NAVALE**

Data una mappa di dimensione 10x10, si inseriscano casualmente (in posizioni non sovrapposte):

- 1 nave da 4 celle
- 2 navi da 3 celle
- 3 navi da 2 celle
- 4 navi da 1 cella

[Si accettano navi in diagonale?]

Fornita, poi, da input una coordinata, si stampi su video se il tiro ha colpito o meno una nave.

# Esercizio 9\* (*Specifica*)

## BATTAGLIA NAVALE

Nell'ipotesi che il computer abbia creato una mappa con delle navi in posizioni casuali (1 nave da 4 celle, 2 navi da 3 celle, 3 navi da 2 celle, 4 navi da 1 cella), sviluppare un programma che consenta ad un giocatore di “scoprire” le posizioni delle navi avversarie.

La classifica dei record viene mantenuta rispetto al numero dei colpi necessari per scoprire tutte le navi nemiche.

[Integrazione: si visualizzi la mappa, con la posizione delle navi scoperte, i tiri effettuati andati a vuoto, e quelli andati a buon fine]

# Esercizio 10\* (*Specifica*)

## GIOCO DELLA VITA

Una mappa di dimensione  $N \times M$  rappresenta il mondo. Ogni cella può essere occupata o meno da un organismo. Partendo da una configurazione iniziale di organismi, questa popolazione evolve nel tempo secondo tre regole genetiche:

- un organismo sopravvive fino alla generazione successiva se ha 2 o 3 vicini;
- un organismo muore, lasciando la cella vuota, se ha più di 3 o meno di 2 vicini;
- ogni cella vuota con 3 vicini diventa una cella di nascita e alla generazione successiva viene occupata da un organismo.

Si visualizzi l'evoluzione della popolazione nel tempo.

# Esercizio 10\* (*nota*)

## GIOCO DELLA VITA

Il concetto di “vicinanza” in una tabella raffigurante il mondo può essere interpretato in 2 modi:

- Al di là dei bordi c’è il vuoto che non influenza il gioco, per cui ci sono punti interni che hanno 8 potenziali “vicini”, punti sulle righe e colonne estreme che hanno 5 “vicini”, punti ai vertici che hanno 3 “vicini”
- I bordi estremi confinano tra di loro: la colonna “0” è “vicina” alla colonna “M”, così come la riga “0” è “vicina” alla riga “N” (con attenzione a trattare i vertici!)