



Linguaggio C++

Vettori e matrici



Problema

Leggere un insieme di numeri interi positivi, fare la somma, calcolare la media, comunicare inoltre quanti dei numeri letti superano la media.

Osservazioni:

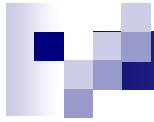
Per poter rispondere all'ultima richiesta del problema è necessario:

- Non poter dimenticare i dati in lettura
- Conservarli tutti in memoria.



Tipi di dato

- Fino ad ora noi abbiamo visto solo i tipi di dati semplici o predefiniti o scalari:
 - integer → int
 - real → float, double
 - boolean → int
 - char → char, int



Tipo di dato

- Ritornando al nostro problema se ad es. devo leggere 1000 numeri interi non posso certo definire 1000 variabili di tipo integer (o meglio lo posso fare, ma che lavoro!).
- I linguaggi di programmazione (quindi anche il C/C++) mi vengono in aiuto con un tipo di dato composto o strutturato: **array (vettore)**.

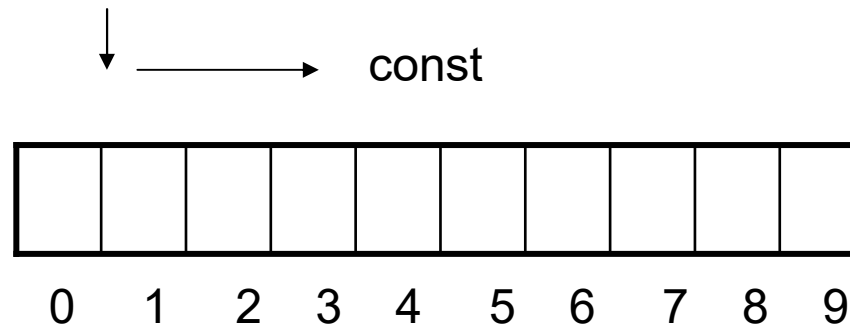
Vettori - array



- L'array può essere considerato come una variabile multipla in grado di contenere tutti i dati di un certo insieme, strutturata in modo che sia possibile individuare in essa, attraverso un indice posizionale, ogni singolo elemento.
- Dunque l'array può essere assimilato ad una struttura lineare di contenitori tutti dello stesso tipo, identificata da un nome collettivo, in cui ogni contenitore è individuato da un indice posizionale.

Vettori - array

- Array di 10 elementi (componenti) interi
- In C/C++:
 - `int vett [10];`



- $0 \leq \text{indice} \leq \text{numComponenti} - 1$
- Operazioni di I/O: componente per componente



Soluzione problema

```
#define DIM_MAX 5
```

```
int main() {  
    int i;  
    int num;  
    int vettore[DIM_MAX];  
    int somma=0;  
    double media;  
    int numSupMedia=0;  
    int numElem=0;
```



Soluzione problema

//inserimento

```
do {  
    cout<<"Quanti elementi vuoi inserire ?\n";  
    cin>>numElem;  
}while(numElem <= 0 || numElem > DIM_MAX);  
  
for(i=0; i < numElem; i++) {  
    cout<<"Digita un numero intero:\n";  
    cin>>num;                //oppure con un'unica istruzione  
    vettore[i]=num;          //cin>>vettore[i];  
}
```




Soluzione problema

//calcolo somma

```
for(i=0; i<numElem; i++)  
    somma+=vettore[i];
```

//calcolo media

```
media = (double)somma/numElem;
```



Soluzione problema

//calcolo del numero di elementi che superano la media

```
for(i=0; i<numElem; i++)  
    if(vettore[i] > media)  
        numSupMedia++;
```

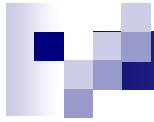
//visualizzazione

```
cout<<"NumElem = "<<numElem<<"Media = "<<media<<"NumSupMedia = "<<numSupMedia;  
  
cin.get();  
return 0;  
}
```



Caratteristiche degli array

- Struttura dati statica = la dimensione della struttura deve essere definita a priori e non può essere modificata durante l'esecuzione. Il compilatore deve conoscere il numero di componenti che quindi deve essere una costante.
- Locazioni contigue di memoria.



Caratteristiche degli array

- Dati omogenei = tutti dello stesso tipo.
- Accesso sequenziale e diretto.
- Individuazione univoca della locazione (componente) in base ad un solo indice.



Vettori - array

- Array → 1° esempio di struttura dati
- **Algoritmi + Strutture dati = Programmi**
(N.Wirth)
- I vettori sono strutture dati che permettono di raggruppare informazioni di uno stesso tipo (anche complesso).



Vettori - array

- In C/C++ non è possibile fare l'assegnazione diretta fra array.
- Array e indici
 - L'indice utilizzato per individuare un elemento in un array deve essere una espressione di tipo intero.
 - Il valore dell'espressione calcolato run-time deve essere sempre compreso tra 0 e la dimensione dell'array meno 1.



Vettori – array: calcolo del max

```
int voti[DIM_MAX];  
int numElem;  
  
.....  
max = voti[0];  
for(i=1; i < numElem; i++)  
    if(voti[ i ] > max)  
        max = voti [ i ];
```



Vettori – array: Inizializzazione di array

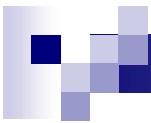
```
int vettore[3] = {1, 2, 3};
```

```
int vettore[3] = {1, 2};
```

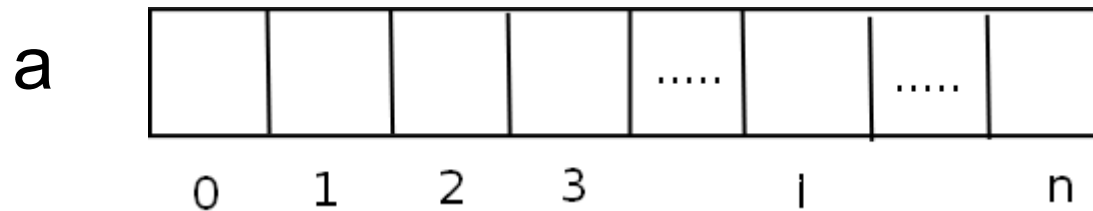
```
int vettore[3] = {1, 2, 3, 4};           //ERRORE!
```

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

```
numElem = sizeof(vettore)/sizeof(int);
```

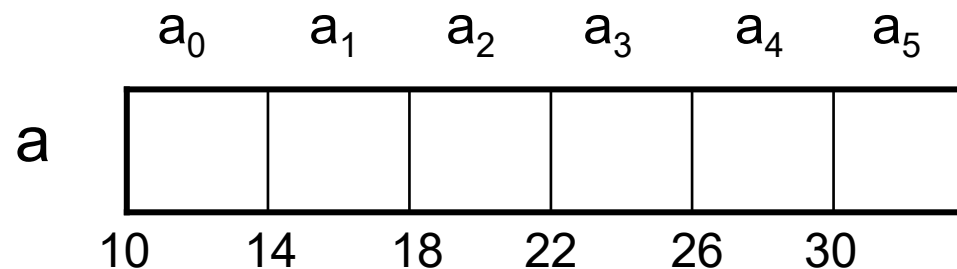



Vettori – array: Implementazione in memoria



■ $\text{Ind}(a_i) = \text{Ind}(a_0) + \underbrace{i * \text{sizeof}(a_i)}_{\text{offset}}$

■ `int a [6];`



$\text{Ind}(a_2) = \text{Ind}(a_0) + 2 * \text{sizeof}(a_0) = 10 + 2 * 4 = 18 \text{ byte}$



Vettori – array: operazioni

- Somma fra vettori
- Differenza fra vettori
- Moltiplicazione di un vettore per uno scalare
- Divisione per un numero $\neq 0$
- Prodotto scalare fra vettori = somma dei prodotti delle componenti di ugual indice.

- Esercizio: scrivi un frammento di codice C++ che calcoli il prodotto scalare fra due vettori.



Soluzione: prodotto scalare

```
int prodScalare = 0;
```

```
.....
```

```
    for (i =0; i < numElem; i++)  
        prodScalare += a[ i ] * b[ i ];
```



Esercizio

- Scrivi un pgm C++ che inserisce dati in un vettore di caratteri.
- Il ciclo termina quando si digita il carattere tappo '\$' (che non deve essere inserito) o quando si raggiunge la dimensione massima del vettore.
- Se si raggiunge la dimensione massima, senza leggere il carattere tappo il pgm deve terminare con un opportuno messaggio.
- Dopo l'inserimento il pgm deve inserire nella variabile numElem il numero di elementi (caratteri) effettivamente inseriti nel vettore.



Soluzione

.....

```
cin>>car;
```

```
for (i=0; i < DIM_MAX && car != '$'; i++) {
```

```
    vett [ i ] = car;
```

```
    cin>>car;
```

```
}
```

```
numElem = i;
```

```
if (car != '$')
```

```
    cout<<"Struttura dati non in grado di contenere tutti gli elementi\n";
```

```
else {
```

```
    ..... //continua elaborazione
```

```
}
```



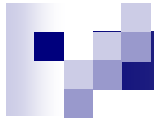
Che cosa fa? Dove sbaglia?

```
cin>>toupper(car);  
while( car != '$') {  
    vett [car - 'A']++;  
    cin>>toupper(car);  
}
```



Struttura dati vettore: operazioni

- Inizializzazione
- Lettura/scrittura (per componenti)
- Assegnazione (per componenti)
- Stampa
- Ricerca
- Sort
- Insert
 - In coda
 - In una certa posizione (voglio mantenere l'ordinamento; attraverso un'operazione di shift a destra)
- Cancellazione
 - Logica
 - Fisica (attraverso un'operazione di shift a sinistra)



Vettore: operazioni

```
#define DIM_MAX 10
```

```
int main() {  
    int vettore[DIM_MAX];  
    int numElem=0;  
  
    .....  
}
```




Inserimento in coda

```
if(numElem < DIM_MAX) {  
    cout<<"Digita l'elemento che vuoi inserire \n";  
    cin>>num;  
    vettore[numElem]=num;  
    numElem++;  
} else  
    cout<<"Struttura piena \n";
```



Stampa vettore

```
if(numElem>0) {  
    cout<<"Numero elementi presenti = "<<numElem<<endl;  
    for(i=0; i<numElem; i++)  
        cout<<vettore[i]<<endl;  
} else  
    cout<<"Struttura vuota \n";
```



Ricerca (sequenziale)

```
if(numElem>0) {  
    cout<<"Digita l'elemento da cercare \n";  
    cin>>num;  
    trovato=0;  
    for(i=0; i<numElem && !trovato; i++)  
        if(vettore[i]==num)  
            trovato=1;  
    if(trovato)  
        cout<<"Elemento presente nella posizione: "<<i<<endl;  
    else  
        cout<<"Elemento non presente \n";  
} else  
    cout<<"Struttura vuota\n";
```



Cancellazione (fisica)

```
int posiz;
if(numElem > 0) {
    cout<<"Digita l'elemento che vuoi cancellare \n";
    cin>>num;
    trovato=0;
    for(i=0; i<numElem && !trovato; i++)           //ricerca
        if(num==vettore[i]) {
            trovato = 1;
            posiz = i;
        }
    if(trovato) {
        for(i=posiz; i<numElem-1; i++)             //cancellazione (fisica)
            vettore[i] = vettore[i+1];
        numElem--;
    } else
        cout<<"Elemento non presente \n";
} else
    cout<<"Struttura vuota \n";
```



Inserimento ordinato

```
if(numElem < DIM_MAX) {  
    int posiz;  
  
    cout<<"Digita l'elemento che vuoi inserire \n";  
    cin>>num;  
    trovato=0;  
    posiz = numElem;                                //gli assegno la prima componente libera  
    for(i=0; i<numElem && !trovato; i++)  
        if(num < vettore[i]) {  
            posiz = i;                                //modifico con la posizione corretta perché sia in ordine  
            trovato = 1;                                //non modifico solo se il valore da inserire è maggiore di tutti gli altri  
        }  
    for(i=numElem; i>posiz; i--)  
        vettore[i] = vettore[i-1];  
    vettore[posiz] = num;  
    numElem++;  
}  
else  
    cout<<"Struttura piena \n";
```

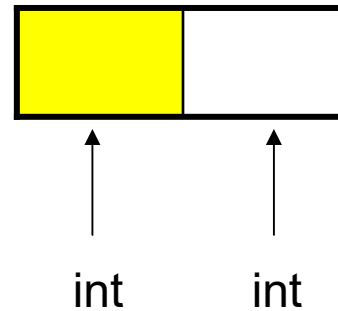


Matrici

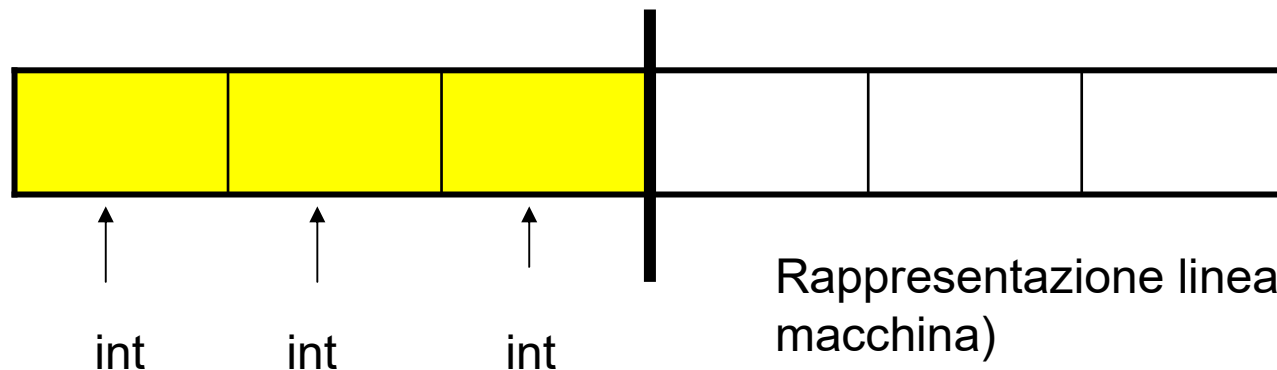
- Abbiamo visto:
 - array di int
 - array di char
 - array di float
 - array di double
- Il tipo base di un array non deve essere necessariamente un tipo elementare (int, char, ...); nel caso in cui esso sia ancora un array si ottiene una struttura dati nota come matrice.

Matrici

■ `int vett[2];`



Se ciascuna componente (dell'array vett) è a sua volta un array (ad es. di 3 interi) si ha:



Matrici

- In C/C++ una struttura di questo tipo viene così definita:

- `int matrix [2] [3];`

 ↑ ↑
 Num. Num.
 righe colonne

Esempio:

$$\text{matrix}_{(2,3)} = \begin{pmatrix} 1 & -1 & 0 \\ 5 & 3 & 1 \end{pmatrix}$$

Rappresentazione
bidimensionale



Matrici

- Una matrice è un array a due dimensioni, in cui è quindi necessario denotare attraverso due indici gli elementi che la compongono.
- Per accedere ad un elemento generico della matrice:
 - `matrix [i] [j]`
 - `i`: indice di riga ($0 \leq i \leq \text{MAX_RIGHE}-1$)
 - `j`: indice di colonna ($0 \leq j \leq \text{MAX_COLONNE}-1$)

Matrici

- In generale una matrice $A_{(m,n)}$ viene così rappresentata:

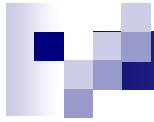
$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n-1} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \dots & a_{m-1,n-1} \end{pmatrix}$$

$a_{i,j} \rightarrow$ elemento generico della matrice $A_{(m,n)}$
con $0 \leq i \leq m-1$ e $0 \leq j \leq n-1$



Matrici

- Se $m \neq n$ la matrice A si dice rettangolare, altrimenti ($m = n$) la matrice è quadrata.
- La dimensione di una matrice è il numero di elementi ($m * n$) e si indica $A_{(m,n)}$.
- Se la matrice A è quadrata, si dice: matrice quadrata di ordine n .



Matrici

Un significato speciale va attribuito alle matrici:

- 1×1 , cioè gli scalari
- $1 \times N$ o $N \times 1$, cioè ai vettori riga o colonna



Prime operazioni in C++

- Assegnazione del numero 0 a tutti gli elementi della matrice:

```
int matrix [2][3];  
    for(i=0; i < 2; i++)  
        for(j=0; j < 3; j++)  
            matrix[i][j] = 0;
```



Prime operazioni in C++

■ Caricamento (lettura dati da tastiera):

```
int matrix [2][3];  
    for(i=0; i < 2; i++)  
        for(j=0; j < 3; j++) {  
            cout<<"Digita .....\\n";  
            cin>>matrix[i][j];  
        }
```



Prime operazioni in C++

■ Stampa (visualizzazione dati sul monitor):

```
int matrix [2][3];
```

```
    for(i=0; i < 2; i++) {  
        for(j=0; j < 3; j++)  
            cout<<"\t"<<matrix[i][j];  
        cout<<"\n";  
    }
```



Prime operazioni in C++

■ Inizializzazione:

```
int matrix[2][3]={{1,2,3}, {4,5,6}};
```

```
int matrix[ ][2]={{1,1}, {2,2}, {3,3}};
```

Nota: esempio di array tridimensionale

```
int arrayTriDim [4][5][3];
```




Esercizio

- Determinare la riga con somma maggiore in una matrice di interi.
 1. insert
 2. calcolo max
 3. stampa



Soluzione

```
#define MAX_RIGHE 3
#define MAX_COLONNE 4

int main() {
    int matrix[MAX_RIGHE][MAX_COLONNE];
    int i,j;
    int max=INT_MIN;
    int somma;
    int numRiga;

    for(i=0; i<MAX_RIGHE; i++)
        for (j=0; j<MAX_COLONNE; j++) {
            cout<<"Inserisci un numero\n";
            cin>>matrix[i][j];
        }
}
```

```
for(i=0; i<MAX_RIGHE; i++) {
    for (j=0, somma=0; j<MAX_COLONNE; j++)
        somma+=matrix[i][j];
    if (somma > max) {
        numRiga = i+1;
        max = somma;
    }
}

cout<<"La riga con somma maggiore e' la
numero" <<numRiga<< " e la somma e' "
<<max;

return 0;
}
```



Tipo di dato

- Def.

1. Insieme di valori
2. Insieme di operazioni/funzioni
3. Implementazione (in memoria)

- Implementazione tipi di dato

- Tipi semplici (int, char, ...): vedi corso di tecnologie
- Tipi strutturati: array (già visto), matrici (vedi prossima slide).

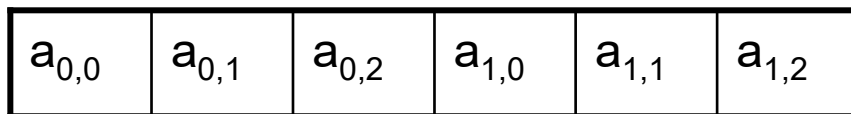
Matrici: Implementazione in memoria

■ `int matrix [2][3];`

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{pmatrix}$$

Struttura astratta

$$\text{Ind}(\text{matrix}[i][j]) = \text{Ind}(\text{matrix}[0][0]) + \underbrace{(\text{DIM_COLONNE} * i + j) * \text{sizeof}(a_{i,j})}_{\text{offset}}$$



Struttura concreta

10 14 18 22 26 30

$$\text{Ind}(\text{matrix}[1][1]) = 10 + (3 * 1 + 1) * 4 = 10 + 16 = 26 \text{ byte}$$



Matrici: Implementazione in memoria

- Il vettore bidimensionale (= matrice) è un'astrazione, visto che la memoria del computer è unidimensionale.
- E' il compilatore che crea una corrispondenza tra la matrice e l'effettivo vettore che costituisce la memoria del computer.



Caratteristiche delle matrici

- Le matrici essendo array bidimensionali godono delle proprietà/caratteristiche che abbiamo visto per ai vettori.
- E precisamente:
 - ☐ Struttura dati statica.
 - ☐ Locazioni contigue di memoria.
 - ☐ Dati omogenei.
 - ☐ Accesso sequenziale e diretto.
 - ☐ Individuazione univoca della locazione (componente) in base a **due** indici.
- Le strutture come gli array/matrici sono tipi derivati in quanto per essere specificati devono riferirsi a un tipo base (int, double, ...).




Operazioni con le matrici

- Somma di matrici
- Differenza di matrici
- Moltiplicazione di una matrice per uno scalare
- Divisione di una matrice per uno scalare diverso da zero
- Moltiplicazione di matrici (vedi prossima slide)
- Trasposta di una matrice



Moltiplicazione di matrici: prodotto righe per colonne

$$\blacksquare A_{(m,n)} * B_{(n,p)} = C_{(m,p)}$$


Devono essere uguali

Date le matrici: $A_{(2,3)}$ e $B_{(2,2)}$ il prodotto $A * B$ non è definito, è definito invece $B * A$.

Nota: La moltiplicazione fra matrici non gode della proprietà commutativa.



Moltiplicazione di matrici: prodotto righe per colonne

$$\blacksquare A_{(2,3)} * B_{(3,2)} = C_{(2,2)}$$

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 0 & 1 \end{pmatrix}$$

Ti ricordi il prodotto scalare fra vettori?

$$C = \begin{pmatrix} 2*1 + 1*1 + 0*0 = 3 & 2*0 + 1*2 + 0*1 = 2 \\ 1*1 + 2*1 + 1*0 = 3 & 1*0 + 2*2 + 1*1 = 5 \end{pmatrix}$$



Altre operazioni con le matrici

- In matematica studierai:
 - Determinante di una matrice (quadrata)
 - Inversa di una matrice (quadrata)
 - Soluzione di sistemi di equazioni lineari con m equazioni in n incognite:
 - Metodi matematici: metodo di Cramer, ...
 - Metodi numerici: metodo di Gauss, Jacobi, ...
 - ...



Tipi di matrici

- Matrice simmetrica: matrice quadrata con elementi simmetrici rispetto alla diagonale principale.
- A è simmetrica se $A = A^T$.



Tipi di matrici

■ Matrice diagonale

In una matrice quadrata, se tutti gli elementi fuori diagonale principale sono nulli ($a_{ij} = 0$ per $i \neq j$), la matrice si dice diagonale.

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & a_{nn} \end{bmatrix}$$

- Matrice unità (identità) è una matrice quadrata in cui tutti gli elementi della diagonale principale sono costituiti dal numero 1, mentre i restanti elementi sono costituiti dal numero 0.

Tipi di matrici

- Matrice triangolare superiore/inferiore: è una matrice quadrata in cui tutti gli elementi che si trovano al di sotto/al di sopra della diagonale principale sono nulli.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

triangolare alta

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

triangolare bassa



Tipi di matrici

- Una matrice sparsa è una matrice i cui valori sono quasi tutti pari a zero.
- Una matrice a banda è una matrice sparsa i cui elementi diversi da zero sono tutti posti in una *banda* diagonale che comprende la diagonale principale.
- Ricorda: Vettore/matrice di booleani.



Interpretazione geometrica

Vettore o array monodimensionale	Retta (spazio ad una dimensione)
Matrice o array bidimensionale	Piano (spazio a due dimensioni)
Array tridimensionale	Spazio a tre dimensioni
Array pluridimensionale	Iperspazio ($n=4,5,6\dots$)