



Array in Matlab

Informatica, AA 2020 / 2021

Francesco Trovò

1 Ottobre 2021

<https://trovo.faculty.polimi.it/>

francesco1.trovo@polimi.it



Warm up

Scrivere un programma per conteggiare quanto la vostra aula ha speso in totale per il pranzo Venerdì scorso.

Calcolare la spesa media per il pranzo e chi ha speso di più



Soluzione

```
n = input('quanti siete?');  
somma = 0; cnt = 1; massimo = 0;  
while(cnt <= n)  
    soldi = input('quanto hai speso?');  
    if (massimo < soldi)  
        massimo = soldi;  
    end  
    somma = somma + soldi;  
    cnt = cnt + 1;  
end  
media = somma / n;  
  
fprintf('avete speso %d (media %f)', somma, media);
```



Soluzione

```
n = input('quanti siete?');  
somma = 0; cnt = 1; massimo = 0;  
while(cnt <= n)  
    soldi = input('quanto hai speso?');  
    if (massimo < soldi)  
        massimo = soldi;  
    end  
    somma = somma + soldi;  
    cnt = cnt + 1;  
end  
media = somma / n;  
  
fprintf('avete speso %d (media %f)', somma, media);
```

cnt è una variabile contatore
che garantisce che il corpo del
while venga eseguito
esattamente n volte



Soluzione

```
n = input('quanti siete?');  
somma = 0; cnt = 1; massimo = 0;  
while(cnt <= n)  
    soldi = input('quanto hai speso?');  
    if (massimo < soldi)  
        massimo = soldi;  
    end  
    somma = somma + soldi;  
    cnt = cnt + 1;  
end  
media = somma / n;  
  
fprintf('avete speso %d (media %f)', somma, media);
```

Il contributo corrente è nella variabile `soldi` che viene sovrascritta ad ogni iterazione del ciclo



Soluzione

```
n = input('quanti siete?');  
somma = 0; cnt = 1; massimo = 0;  
while(cnt <= n)  
    soldi = input('quanto hai speso?');  
    if (massimo < soldi)  
        massimo = soldi;  
    end  
    somma = somma + soldi;  
    cnt = cnt + 1;  
end  
media = somma / n;  
  
fprintf('avete speso %d (media %f)', somma, media);
```

Accumulo in `somma` il totale versato. Il contributo corrente è nella variabile `soldi`



Altri quesiti

Altre domande:

- chi ha speso di più di tutti
- se qualcuno ha speso più di tutti gli altri messi assieme
- Si supponga di «fare alla romana» e che quindi tutti devono pagare il prezzo medio. Dire a chi ha pagato di quanto deve ricevere e a chi ha pagato quanto deve versare.



Altri quesiti

Altre domande:

- chi ha speso di più di tutti
- se qualcuno ha speso più di tutti gli altri messi assieme
- Si supponga di «fare alla romana» e che quindi tutti devono pagare il prezzo medio. Dire a chi ha pagato di quanto deve ricevere e a chi ha pagato quanto deve versare.

Per rispondere all'ultima domanda servirebbe **tener traccia** di quanto viene «versato» da ciascuno (i.e. i valori assegnati alla variabile `soldi`).

Riprendendo il paragone variabili-foglietti su cui scrivere, servirebbe, al posto di un foglietto `soldi`, una **sequenze di foglietti**, ed ciascun foglietto tiene traccia dei valori inseriti

Quindi sequenze di variabili: gli array


```
n = input('quanti siete?');  
somma = 0; cnt = 1; massimo = 0;  
while(cnt <= n)  
    soldi(cnt) = input('quanto hai speso?');  
    if (massimo < soldi(cnt))  
        massimo = soldi(cnt);  
    end  
    somma = somma + soldi(cnt);  
    cnt = cnt + 1;  
end  
media = somma / n;  
fprintf('avete speso %d (media %f)', somma, media);  
if(massimo > somma - massimo)  
    fprintf('qualcuno ha speso più di tutti gli altri\n');  
end
```

Aggiungo un indice alla variabile `soldi`. In Matlab questo basta a farla diventare una sequenza di variabili.
`soldi(cnt)` rappresenta il valore del foglietto alla posizione `cnt`.

```
n = input('quanti siete?');  
somma = 0; cnt = 1; massimo = 0;  
while(cnt <= n)  
    soldi(cnt) = input('quanto hai speso?');  
    if (massimo < soldi(cnt))  
        massimo = soldi(cnt);  
    end  
    somma = somma + soldi(cnt);  
    cnt = cnt + 1;  
end  
media = somma / n;  
fprintf('avete speso %d (media %f)', somma, media);  
if(massimo > somma - massimo)  
    fprintf('qualcuno ha speso più di tutti gli altri\n');  
end
```

Utilizzo la variabile
`soldi(cnt)` come una
qualsiasi altra variabile

...

```
m = 0; %studenti che hanno speso sopra la media
```

```
cnt = 1;
```

```
while(cnt <= n)
```

```
  if(soldi(cnt) > media)
```

```
    m = m + 1;
```

```
  end
```

```
  cnt = cnt + 1;
```

```
end
```

```
fprintf('spesa tot: %d (media %f) e %d persone sono sopra la media', somma, media, m);
```

```
if(max > somma - max)
```

```
  fprintf('qualcuno ha più di tutti gli altri\n');
```

```
end
```

Ora è possibile accedere ai valori inseriti precedentemente scrivendo `soldi(cnt)` nel codice.



Esercizio Challenging

Scrivere un programma che richiede in ingresso due vettori e

- Rimuove elementi duplicati
- Calcola l'intersezione dei due vettori
- Calcola l'unione dei due vettori



Tipi di Dato Strutturati

Gli array



Andiamo oltre i tipi visti la volta scorsa...

Permettono di immagazzinare informazione aggregata

- vettori e matrici in matematica
- Testi (sequenza di caratteri)
- Immagini
- Rubriche
- Archivi
- ...

Le variabili strutturate memorizzano diversi elementi informativi:

- omogenei
- eterogenei

Oggi vedremo gli **array**



Gli Array

Gli array sono **sequenze di variabili omogenee**

- **sequenza:** hanno un ordinamento (sono indicizzabili)
- **omogenee:** tutte le variabili della sequenza sono dello stesso tipo

Ogni elemento della sequenza è individuato da un indice



Creazione di un array in Matlab

Come abbiamo visto nell'esempio, basta aggiungere un indice ad una variabile per trasformarla in un array

```
>> vet(1) = 10;
```

```
>> vet(2) = 20;
```

```
>> vet(3) = 30;
```

Quindi, in Matlab, gli array si dichiarano come le variabili: mediante assegnamento

Espressioni alternative per dichiarare un vettore di tre elementi sono:

```
vet = [10, 20, 30];
```

```
vet = [10 20 30]; (virgole non necessarie)
```




Accedere agli elementi dell'array

È possibile accedere agli elementi dell'array specificandone **un indice** tra parentesi tonde ()

`vet (1)` è il primo elemento della sequenza

`vet (20)` è il ventesimo elemento della sequenza

`vet (end)` è l'ultimo elemento della sequenza

Attenzione che la keyword **end** ha due significati diversi:

- Termina un costrutto
- Indica l'ultimo elemento di un vettore (non occorre conoscere la lunghezza del vettore)



Gli elementi dell'array

Ogni elemento dell'array è una **variabile** del **tipo** dell'array:

`vet (7)` conterrà un valore intero

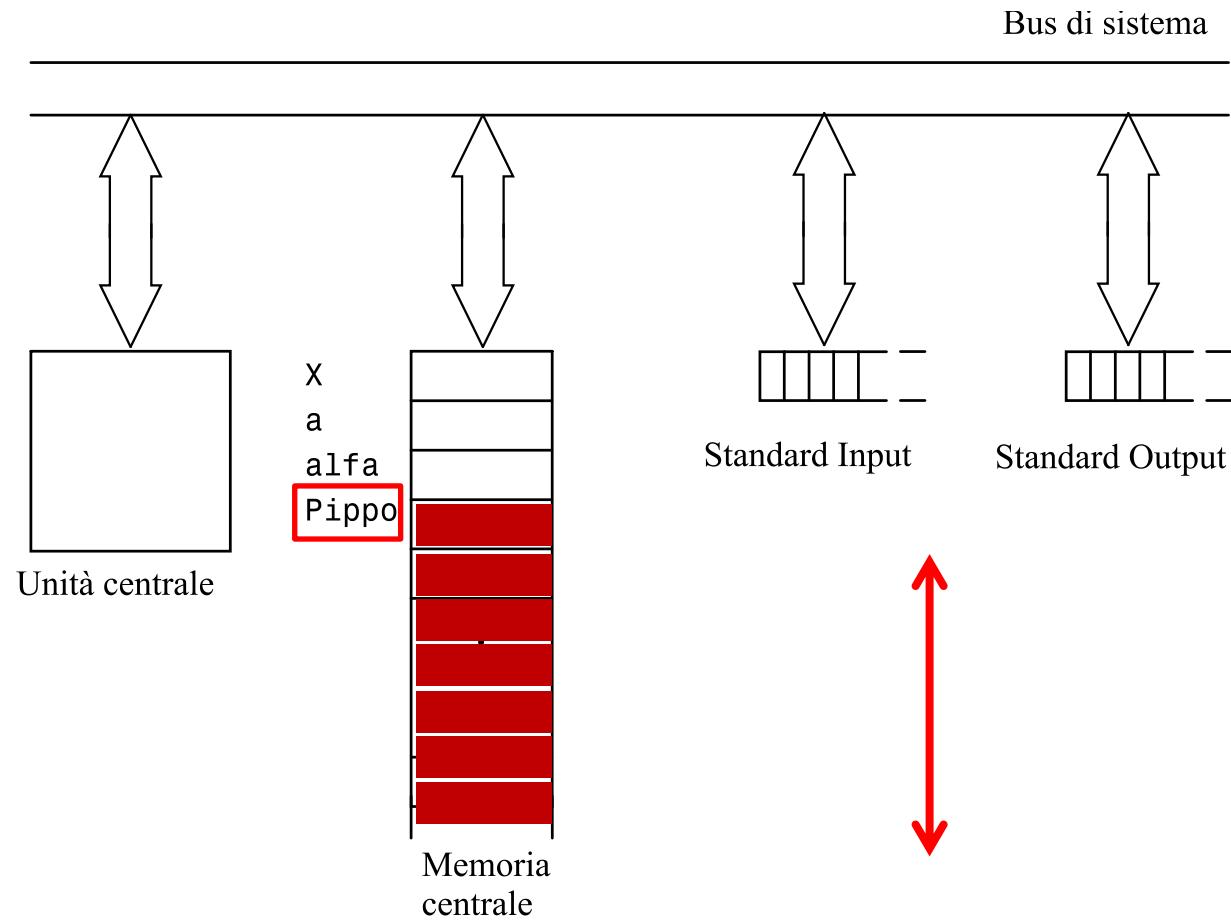
Una volta **fissato l'indice**, non c'è differenza tra un elemento dell'array ed una qualsiasi **variabile** dello stesso tipo

```
a = vet (1) ; vet (1) = a ; vet (1) = vet (1) + a ;
```



Lo spazio allocato per gli array

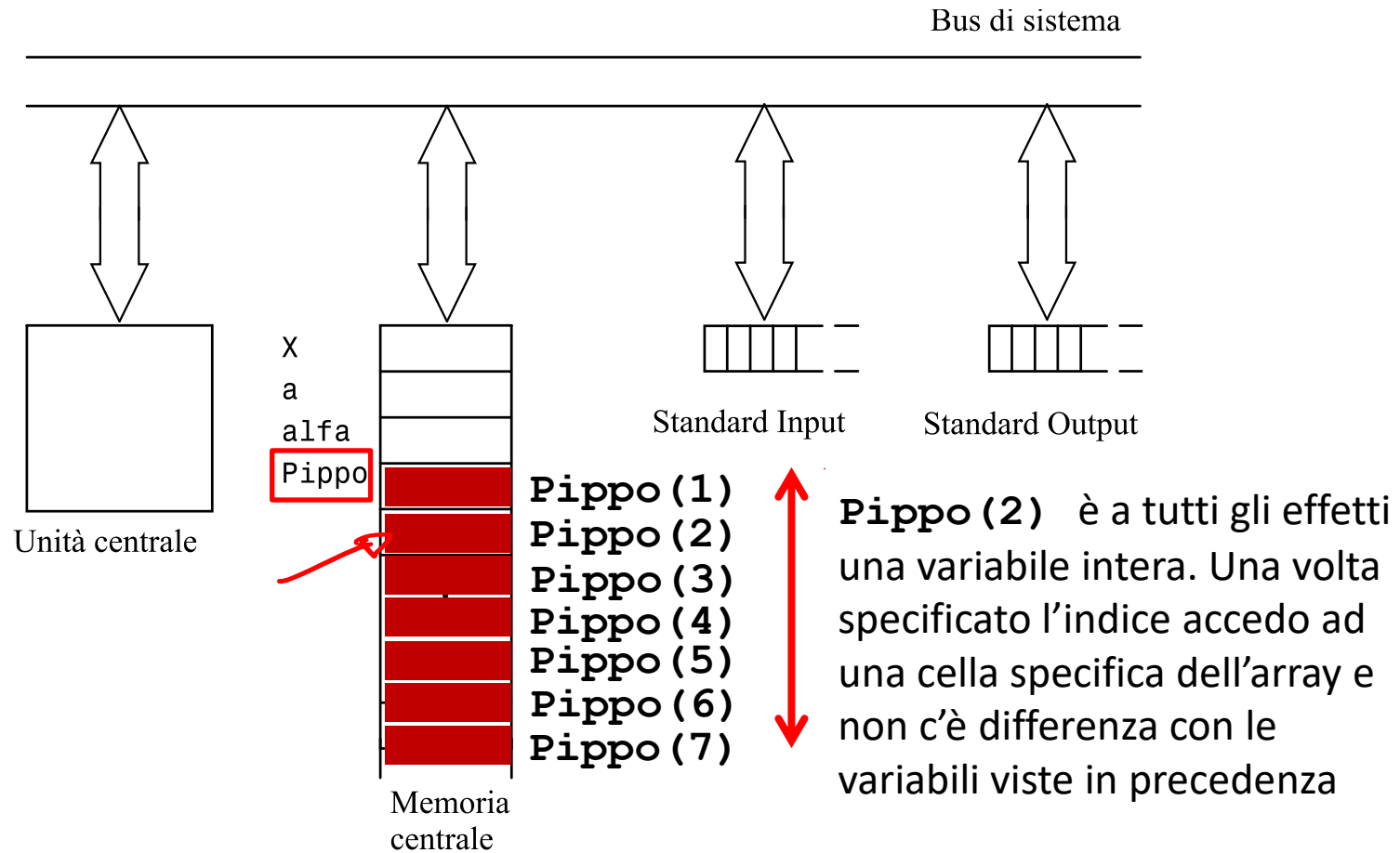
```
Pippo = [1, 2, 3, 4, 5, 6, 7];
```





Lo spazio allocato per gli array

`Pippo = [1, 2, 3, 4, 5, 6, 7];`





Accedere agli elementi dell'array

Il valore **dell'indice** è un intero positivo o un logical (vedremo poi)

È quindi possibile utilizzare una **variabile per definire l'indice** all'interno dell'array

L'espressione: **vet(i)**

va interpretata nel seguente modo:

1. Leggi il valore di **i**
2. Accedi all'elemento di **vet** alla posizione di indice **i**
3. Leggi il valore che trovi in quella cella di memoria (**vet(i)**)

Allo stesso modo viene prima valutata qualsiasi espressione tra le parentesi del vettore come:

```
vet(i + 1) ;
```



Esempi di Operazioni su Array

Una volta **fissato l'indice** in un array si ha una **variabile del tipo dell'array** che può essere usata per

- assegnamenti

```
vet(2) = 7; vet(4) = 8 / 3;  
i = 1; vet(i) = vet(i+1);
```

- operazioni logiche

```
vet(1) == vet(9); vet(1) < vet(4);
```

- operazioni aritmetiche

```
vet(1) == vet(9) / vet(2) + vet(1) / 6;
```

- operazioni di I/O

```
vet(9) = input('inserire valore');  
fprintf('valore pos %d = %d', i, vet(i));
```



.. e senza Array

```
a = 1;
```

```
b = 2;
```

```
c = 3;
```

vs

```
vet(1) = 1;
```

```
vet(2) = 2;
```

```
vet(3) = 3;
```

Come faccio a richiamare "il secondo valore inserito"?

- Con le variabili devo salvare da qualche parte che **a** contiene il primo valore, **b** il secondo... perché le variabili non hanno un ordinamento
- Con il vettore mi basta accedere a **vet(2)** perché gli elementi di un vettore seguono un ordinamento



.. e senza Array

```
a = 1;
```

```
b = 2;
```

```
c = 3;
```

vs

```
vet(1) = 1;
```

```
vet(2) = 2;
```

```
vet(3) = 3;
```

La soluzione diventa decisamente impraticabile quando si richiedono molte variabili:
occorre usare array

- perché sono indicizzati
- perché posso popolarli/elaborarli con un ciclo



Esempio

Scrivere un programma che mette in ogni cella di un array un numero da 1 a 300.



Esempio

Scrivere un programma che mette in ogni cella di un array un numero da 1 a 300.

```
cnt = 1;  
while (cnt <= 300)  
    vet(cnt) = cnt;  
    cnt = cnt + 1;  
end
```



Definizione di Vettori

I vettori sono definiti tra parentesi quadre:

- In un vettore riga gli elementi sono separati da virgole (o spazi)
- In un vettore colonna gli elementi sono separati da ; (o andando a capo)

Es:

```
>> a = [1 2 3]
```

```
a =
```

```
1 2 3
```

```
>> a = [1, 2, 3]
```

```
a =
```

```
1 2 3
```

```
>> a = [1; 2; 3]
```

```
a =
```

```
1
```

```
2
```

```
3
```



Le dimensioni degli array

```
>> a = [1 2 3]
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x3	24	double	

```
>> a = [1; 2; 3]
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	3x1	24	double	



Operatori per Array: Trasposizione

L'operatore ' esegue la **trasposizione** (i.e. trasforma un vettore riga ad uno colonna e viceversa)

```
>> a = [1 2 3]
      a =
      1     2     3
```

```
>> a'
      ans =
      1
      2
      3
```



Calcolare la lunghezza di un vettore

È spesso necessario dover sapere quanti elementi sono stati inseriti in un vettore

Il comando `length` restituisce il numero di elementi lungo la dimensione maggiore

```
>> v = [1, 2, 3];
```

```
>>length(v)
```

```
ans = 3
```

per questo motivo `length` funziona anche sui vettori colonna

```
>> v = [1; 2; 3];
```

```
>>length(v)
```

```
ans = 3
```



Definizione di array mediante incremento regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`[inizio : step : fine]`

Definisce un vettore che ha:

- primo elemento `inizio`
- secondo elemento `inizio + step`
- terzo elemento `inizio + 2*step`
- ...
- fino al più grande valore `inizio + k*step` che non supera `fine` (`fine` potrebbe non essere incluso)



Note

Il valore di **step** può essere qualsiasi, anche negativo.

Se non precisato, **step** vale 1

Le parentesi [] possono essere omesse

Attenzione che i vettori definiti per incremento regolare possono essere vuoti
(es >> [10 : -1 : 20])

È ovviamente possibile modificare i valori di un array mediante assegnamento

- Di un singolo elemento
- Di una parte dell'array (vedremo poi)



Ad esempio

`vet = [1 : 1 : 10]`

`vet = [1 : 0.1 : 10]`

`vet = [1 : 2 : 10]`

`vet = [10 : -1 : 0]`

`vet = [10 : 1 : 0]`

`vet = [1 : 3]`

`vet = 1 : 3`

`vet = [1 : 3]'`



Ad esempio

```
vet = [1 : 1 : 10] % 11 numeri interi da 1 a 10
```

```
vet = [1 : 0.1 : 10] % [1, 1.1, ..., 9.9, 10]
```

```
vet = [1 : 2 : 10] % 5 numeri dispari da 1 a 9
```

```
vet = [10 : -1 : 0] % 12 numeri da 10 a 0
```

```
vet = [10 : 1 : 0] % empty matrix
```

```
vet = [1 : 3] % [1,2,3] (passo 1 implicito)
```

```
vet = 1 : 3 % [1,2,3] parentesi non necessarie
```

```
vet = [1 : 3]' % traspone il vettore e ottiene un vettore colonna
```



Assegnamento tra Vettori

In Matlab è possibile eseguire direttamente assegnamenti tra array

```
nomeArray1 = espressione
```

Valuta espressione e copia il risultato in **nomeArray1**

```
>> a = a + 1
```

```
a =
```

```
2 3 4
```

Non è necessario che **gli array abbiano la stessa dimensione**

```
>> b = [1 : 4]
```

```
b =
```

```
1 2 3 4
```

```
>> a = b
```

```
a =
```

```
1 2 3 4
```



Accedere agli Elementi di una Array

Viene segnalato un **errore** quando si **accede** ad una **posizione che non corrisponde ad un elemento** dell'array (vale anche per matrici e array multidimensionali)

```
>> a = [1 : 3]
a =
     1     2     3

>> a(2)
ans =
     2

>> a(4)
Index exceeds matrix dimensions.

>> a(1.3)
Subscript indices must either be real positive integers or logicals.
```



Accedere agli Elementi di una Array

Viene segnalato un **errore** quando si **accede** ad una **posizione che non corrisponde ad un elemento** dell'array (vale anche per matrici e array multidimensionali)

```
>> a = [1 : 3]
a =
     1     2     3
>> ii = 2;
>> a(ii)
ans =
     2
```

È possibile utilizzare una variabile per definire l'indice

```
>> a(ii) = a(ii - 1) + a(ii + 1)
a =
     1     4     3
```



Il ciclo for



Il ciclo for

```
for variabile = array
    istruzioni
end
```

Tipicamente **array** è un vettore, quindi **variabile** assume valori scalari

- Alla prima iterazione **variabile** è **array(1)**
- Alla seconda iterazione **variabile** è a **array(2)**
- All'ultima iterazione **variabile** è **array(end)**

NB: Non esiste alcuna condizione da valutare per definire la permanenza nel ciclo. Il numero di iterazioni dipende dalle dimensioni di array

NB: se **array** è un'espressione booleana viene scandito come il vettore logico.



Ad esempio

```
soldi = [50 45 23]
for s = soldi
    s
end
```

Il ciclo verrà eseguito 3 volte, perchè `soldi` è lungo 3
`s` varrà 50 la prima volta, 45 la seconda volta, poi 23

Vedrò a schermo:

```
s =
    50
s =
    45
s =
    23
```




Riprendiamo il primo esercizio

```
somma = 0;
cnt = 1;
massimo = 0;
while(cnt <= n)
    soldi(cnt) = input('quanto hai?');
    if (massimo < soldi(cnt))
        massimo = soldi(cnt);
    end
    somma = somma + soldi(cnt);
    cnt = cnt + 1;
end
```



Riprendiamo il primo esercizio

```
somma = 0;  
cnt = 1;  
massimo = 0;  
while (cnt <= n)  
    soldi(cnt) = input('quanto hai?');  
    if (massimo < soldi(cnt))  
        massimo = soldi(cnt);  
    end  
    somma = somma + soldi(cnt);  
    cnt = cnt + 1;  
end
```

La variabile cnt assumerà i seguenti valori durante l'esecuzione del ciclo
1, 2, ..., n
Quindi posso farla variare nel vettore [1 : n]



Riprendiamo il primo esercizio

```
somma = 0;  
massimo = 0;  
for cnt = [1 : n]  
    soldi(cnt) = input('quanto hai?');  
    if (massimo < soldi(cnt))  
        massimo = soldi(cnt);  
    end  
    somma = somma + soldi(cnt);  
end
```



Il ciclo for

Non è equivalente al `while`, ha meno potere espressivo: ad esempio non è possibile eseguire infinite volte il corpo di un `for`

Ogni `for` può essere scritto come un `while`

```
for c = [10, 22, 43]
    fprintf("%d", c)
end
```

c assumerà ad ogni iterazione un carattere diverso nel vettore [1,2,3]



Il ciclo for

Non è equivalente al `while`, ha meno potere espressivo: ad esempio non è possibile eseguire infinite volte il corpo di un `for`

Ogni `for` può essere scritto come un `while`

```
for c = [10, 22, 43]
    fprintf("%d", c)
end
```

```
vet = [10, 22, 43]
ii = 1;
while (ii <= length(vet))
    fprintf("%d", vet(ii))
    ii = ii + 1;
end
```



Il ciclo for

Non è equivalente al `while`, ha meno potere espressivo: ad esempio non è possibile eseguire infinite volte il corpo di un `for`

Ogni `for` può essere scritto come un `while`

```
for c = [10, 22, 43]
    fprintf("%d", c)
end
```

```
vet = [10, 22, 43]
ii = 1;
while (ii <= length(vet))
    fprintf("%d", vet(ii))
    ii = ii + 1;
end
```

Occorre usare un indice esplicito `ii`

Occorre scorrere il vettore calcolandone la lunghezza

Occorre incrementare `ii`



Il ciclo for

Non è equivalente al `while`, ha meno potere espressivo: ad esempio non è possibile eseguire infinite volte il corpo di un `for`

Ogni `for` può essere scritto come un `while`

```
for c = [10, 22, 43]
    fprintf("%d", c)
end
```

```
vet = [10, 22, 43]
ii = 1;
while (ii <= length(vet))
    fprintf("%d", vet(ii))
    ii = ii + 1;
end
```

Per scorrere un vettore noto, il ciclo `for` è molto più comodo del `while`, se invece il numero di iterazioni da eseguire non è noto a priori è preferibile usare `while`



Il ciclo `for`, la variabile del ciclo

```
for variabile = array
    istruzioni
end
```

`array` può essere generato “al volo”, molto spesso tramite l’operatore di incremento regolare, i.e., “inizio : step : fine”

- Nel primo esempio precedente l’array è
[1 2 3 4 5 6 7]

N.B : questo

```
for i = [1:n]
    istruzioni
end
```

è un utilizzo molto frequente del ciclo `for` ma non è l’unico! La definizione è più generale e quella sopra!



Esempi

```
% richiedi all'utente 7 numeri in un vettore number:
```

```
% stampa conto alla rovescia in secondi
```



Esempi

```
% richiedi all'utente 7 numeri in un vettore number:
for n = 1:7
    number(n) = input('enter value ');
end

% stampa conto alla rovescia in secondi
time = input('how long? ');
for count = time:-1:1
    pause(1);
    fprintf('%d seconds left \n',count);
end
fprintf('BOOM!');
```



I/O con vettori



Acquisizione di un array

Ci sono due modi per richiedere all'utente un vettore:

- Richiedendo elemento per elemento
- Sfruttando input che permette di inserire qualsiasi valore in formato Matlab

```
>> v = input('inserire vettore')
```

```
inserire vettore [12,23,45]
```

```
v =
```

```
    12    23    45
```



Stampa dei valori dell'array

Con `fprintf` non esiste un fattore di conversione per stampare gli array di numeri. Quindi occorre procedere iterando

```
vettore = [0 : 5 : 20];  
fprintf('[ ');  
for v = vettore  
    fprintf(' %d ', v);  
end  
fprintf(']');
```



Stampa dei valori dell'array

`disp` invece permette di stampare vettori e array multidimensionali

```
>> disp(v1)
```

```
    2    3    5
```

È possibile anche stampare sequenze di caratteri

```
>> disp('ciao mondo')
```

```
ciao mondo
```



Una nota sul costrutto if

espressione1 può coinvolgere vettori:

- in tal caso **espressione1** è vera solo se tutti gli elementi di **espressione1** sono non nulli

Esempio

```
v = input('inserire vettore: ');  
if (v >= 0)  
    disp([num2str(v), ' tutti pos. o nulli']);  
elseif (v < 0)  
    disp([num2str(v), ' tutti negativi']);  
else  
    disp([num2str(v), ' sia pos. che neg.']);  
end
```



Operazioni su Array



Confronto tra array

Come fare a controllare che due array coincidano (quindi che abbiano lo stesso numero di elementi e che l' i -simo elemento del primo corrisponde con l' i -simo del secondo)?

Operiamo su ogni singolo elemento, richiedendo che in ogni posizione coincidano (il che equivale a dire che in nessuna posizione siano diversi)

```
v1 = input('inserire vettore1');
v2 = input('inserire vettore2');
uguali = true;
if length(v1) == length(v2)
    l = length(v1);
    ii = 1;
    while(ii <= l && uguali == true)
        if(v1(ii) ~= v2(ii))
            uguali = false;
        end
        ii = ii + 1;
    end
else
    uguali = false;
end
disp(v1); disp('e'); disp(v2);
if uguali
    disp('sono uguali');
else
    disp('sono diversi');
end
```

```
v1 = input('inserire vettore1');
v2 = input('inserire vettore2');
uguali = true;
if length(v1) == length(v2)
    l = length(v1);
    ii = 1;
    while(ii <= l && uguali == true)
        if(v1(ii) ~= v2(ii))
            uguali = false;
        end
        ii = ii + 1;
    end
else
    uguali = false;
end
disp(v1); disp('e'); disp(v2);
if uguali
    disp('sono uguali');
else
    disp('sono diversi');
end
```

Variabile di flag, diventa false appena trova una cella per cui **v1** e **v2** differiscono

Scorro tutti gli elementi dei vettori. Mi arresto appena trovo due elementi diversi

Sta per **uguali** \approx 0



Variabili di Flag per Verificare Condizioni su Array

Per controllare che una condizione (uguaglianza in questo caso) sia soddisfatta da tutti gli elementi del vettore

```
uguali = true;  
while(ii <= l && uguali == true)  
    if(v1(ii) ~= v2(ii))  
        uguali = false;  
    end  
    ii = ii + 1;
```

End

Al termine del ciclo, se uguali è rimasta **1** sono certo che la condizione da verificare **non è mai stata negata** (i.e., $v1[i] \neq v2[i]$ è sempre stata falsa). Quindi che **tutti** gli elementi degli array coincidono.




Variabili di Flag per Verificare Condizioni su Array

La variabile di flag (`uguali`) può solo cambiare da **1** in **0**

Ovviamente il ruolo di **0** e di **1** possono essere invertiti in maniera consistente

Errore frequente: modificare il valore della variabile di flag nel anche nel verso opposto.

```
while(ii <= l)
    if(v1(ii) ~= v2(ii))
        uguali = false;
    else
        uguali = true; 
    end
    ii = ii + 1;
end
```




Variabili di Flag per Verificare Condizioni su Array

La variabile di flag (`uguali`) può solo cambiare da **1** in **0**

Ovviamente il ruolo di **0** e di **1** possono essere invertiti in maniera consistente

Errore frequente: modificare il valore della variabile di flag nel anche nel verso opposto.

```
while(ii <= l && uguali == true)
    if(v1(ii) ~= v2(ii))
        uguali = false;
    else
        uguali = true; 
    end
    ii = ii + 1;
end
```



Copiare alcuni elementi da un array ad un altro

In molti casi è richiesto di **scorrere** un array **v1** e di **selezionare** alcuni valori secondo una data condizione.

Tipicamente i valori selezionati in **v1** vengono **copiati in un secondo array, v2**, per poter essere utilizzati.

È buona norma copiare i valori **nella prima parte** di **v2**, eseguendo quindi una copia «senza lasciare buchi».

È anche necessario sapere quali sono i valori significativi in **v2** e quali no.

Esempio : copiare i numeri pari in **v1** in **v2**

v1	5	6	7	89	568	68	657	989	96	98
v2 ✘	0	6	0	0	568	68	0	0	96	98
v2	6	568	68	96	98					



Copiare alcuni elementi da un array ad un altro

Per fare questo è necessario usare **due indici**:

- **i** per **scorrere v1**: parte da **1** e arriva a **n1** (la dimensione effettiva di **v1**) procedendo con **incrementi regolari**.
- **n2** parte da **1** e viene **incrementata solo quando un elemento viene copiato un elemento in v2**
 - **n2** indica quindi **il primo elemento libero in v2**,
 - al termine, **n2** conterrà il **numero di elementi in v2**, quindi la sua **dimensione**

5	6	7	89	568	68	657	989	96	98
---	---	---	----	-----	----	-----	-----	----	----

`i = 10;`
`n1 = 10;`

6	568	68	96	98
---	-----	----	----	----

`n2 = 6;`



Esempio

Chiedere all'utente di inserire un array di interi (di dimensione definita precedentemente) e quindi un numero intero n . Il programma quindi:

- salva gli elementi inseriti in un vettore $v1$.
- Copia tutti gli elementi di $v1$ che sono maggiori di n in un secondo vettore $v2$.
- La copia deve avvenire nella parte iniziale di $v2$, senza lasciare buchi.

```
v1 = input(['inserire primo vettore ']);

% copiamo tutti gli elementi pari da v1 a v2
j = 1; % la prima posizione disponibile in v2
for x = 1 : length(v1)
    % scorro v1 regolarmente
    if mod(v1(x), 2) == 0
        %v1(x) è pari e va copiato in v2
        v2(j) = v1(x);
        j = j + 1; % incremento j solo quando copio
        % j indica la prima posizione disponibile in v2
    end
end
disp(v2);
```



Concatenare i Vettori

L'operatore , e ; permettono di concatenare vettori, purché le dimensioni siano compatibili (devono essere entrambi riga o colonna).

Esempio:

```
>> a = [1, 2, 3]
```

```
a =
```

```
    1    2    3
```

```
>> b = [a, a + 3, a + 6]
```

```
b =
```

```
    1
```

```
>>
```

```
b :
```

```
    1
```



Concatenare i Vettori

L'operatore , e ; permettono di concatenare vettori, purché le dimensioni siano compatibili (devono essere entrambi riga o colonna).

Esempio:

```
>> a = [1, 2, 3]
```

```
a =
```

```
 1      2      3
```

```
>> b = [a, a + 3 , a + 6]
```

```
b =
```

```
 1  2  3  4  5  6  7  8  9
```

```
>> b = [a, a + 3]
```

```
b =
```

```
 1
```



Concatenare i Vettori

L'operatore `,` e `;` permettono di concatenare vettori, purché le dimensioni siano compatibili (devono essere entrambi riga o colonna).

Esempio:

```
>> a = [1, 2, 3]
```

```
a =
```

```
 1      2      3
```

```
>> b = [a, a + 3, a + 6]
```

```
b =
```

```
 1  2  3  4  5  6  7  8  9
```

```
>> b = [a, a + 3]
```

```
b =
```

```
 1  2  3  1  2  3  3
```

Viene interpretato
come `b = [a, a, +3]`
ATTENZIONE agli spazi



Concatenare i Vettori

Esempi

- $\mathbf{a} = [0 \ 7+1];$

- $\mathbf{b} = [\mathbf{a}(2) \ 5 \ \mathbf{a}];$

contenuto di a

secondo elemento di a

Risultato

- $\mathbf{a} = [0 \ 8]$

- $\mathbf{b} = [8 \ 5 \ 0 \ 8]$



Soluzione della copia senza lasciare buchi

```
v1 = input(['inserire primo vettore ']);

% copiamo tutti gli elementi pari da v1 a v2
v2 = []; % devo inizializzare v2 al vettore vuoto
for x = 1 : length(v1)
    if mod(v1(x), 2) == 0
        v2 = [v2, v1(x)]; % accoda el corrente di v1 a v2
    end
end
end
```

Questa soluzione non richiede la variabile j per tener traccia dell'inserimento in v2



Operazioni Aritmetiche tra Vettori

Le **operazioni aritmetiche** sono quelle **dell'algebra lineare**

- Somma tra vettori: $\mathbf{c} = \mathbf{a} + \mathbf{b}$
 - E' definita elemento per elemento

$$c(i) = a(i) + b(i), \quad \forall i$$

è possibile solo quando \mathbf{a} e \mathbf{b} hanno la stessa dimensione (che poi coincide con quella di \mathbf{c})

Prodotto tra vettori:

- È il prodotto riga per colonna, restituisce uno scalare

$$\mathbf{c} = \mathbf{a} * \mathbf{b}, \text{ i.e. } c = \sum_i a(i)b(i)$$

\mathbf{a} deve essere un vettore riga e \mathbf{b} colonna e devono avere lo stesso numero di elementi, \mathbf{c} è un numero reale



Operazioni Puntuali

E' possibile eseguire operazioni **puntuali**, che si applicano cioè ad ogni elemento del vettore separatamente

$$\mathbf{c} = \mathbf{a} \ . * \ \mathbf{b}, \text{ restituisce } c(i) = a(i) * b(i) \ \forall i$$

$$\mathbf{c} = \mathbf{a} \ . / \ \mathbf{b}, \text{ restituisce } c(i) = a(i)/b(i) \ \forall i$$

$$\mathbf{c} = \mathbf{a} \ . ^ \ \mathbf{b}, \text{ restituisce } c(i) = a(i)^{b(i)} \ \forall i$$

Come in algebra lineare, le **operazioni tra vettori (array)** e **scalari** sono possibili, e corrispondono ad operazioni puntuali. Se **k** è uno scalare

$$\mathbf{c} = \mathbf{k} * \mathbf{b} = \mathbf{k} \ . * \ \mathbf{b} \quad c(i) = k * b(i) \ \forall i$$



Attenzione: elevamento a potenza

```
>> v1 = [2      3      5      4]
```

```
>> v1^2
```

```
Error using ^
```

```
Inputs must be a scalar and a square matrix.
```

```
To compute elementwise POWER, use POWER (.^) instead.
```

L'elevamento a potenza fa' riferimento al prodotto vettoriale (equivale quindi a $v1 * v1$)

Per elevare a potenza ogni singolo elemento di $v1$ si usa:

```
>> v1.^2
```

```
ans =
```

```
4      9      25     16
```

Che equivale a fare $v1 .* v1$



Operazioni Aritmetiche su Array

Operazione	Sintassi Matlab	Commenti
Array addition	$a + b$	Array e matrix addition sono identiche
Array subtraction	$a - b$	Array e matrix subtraction sono identiche
Array multiplication	$a .* b$	Ciascun elemento del risultato è pari al prodotto degli elementi corrispondenti nei due operandi
Matrix multiplication	$a * b$	Prodotto di matrici
Array right division	$a ./ b$	$\text{risultato}(i,j)=a(i,j)/b(i,j)$
Array left division	$a .\ b$	$\text{risultato}(i,j)=b(i,j)/a(i,j)$
Matrix right division	a / b	$a * \text{inversa}(b)$
Matrix left division	$a \ b$	$\text{inversa}(a) * b$
Array exponentiation	$a .^ b$	$\text{risultato}(i,j)=a(i,j)^b(i,j)$