Tipi di Dato Strutturati: Array e Strutture

Informatica (ICA) AA 2020 / 2021

Giacomo Boracchi

16 Ottobre 2020

giacomo.boracchi@polimi.it

Calcolare la lunghezza di un vettore

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

Il commando length restituisce il numero di elemenit lungo la dimensione maggiore

```
>> v = [1, 2, 3];
>>length(v)
ans = 3
```

per questo motive 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)

[3 : 2 : 9]

[3, 5, 7, 9]

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

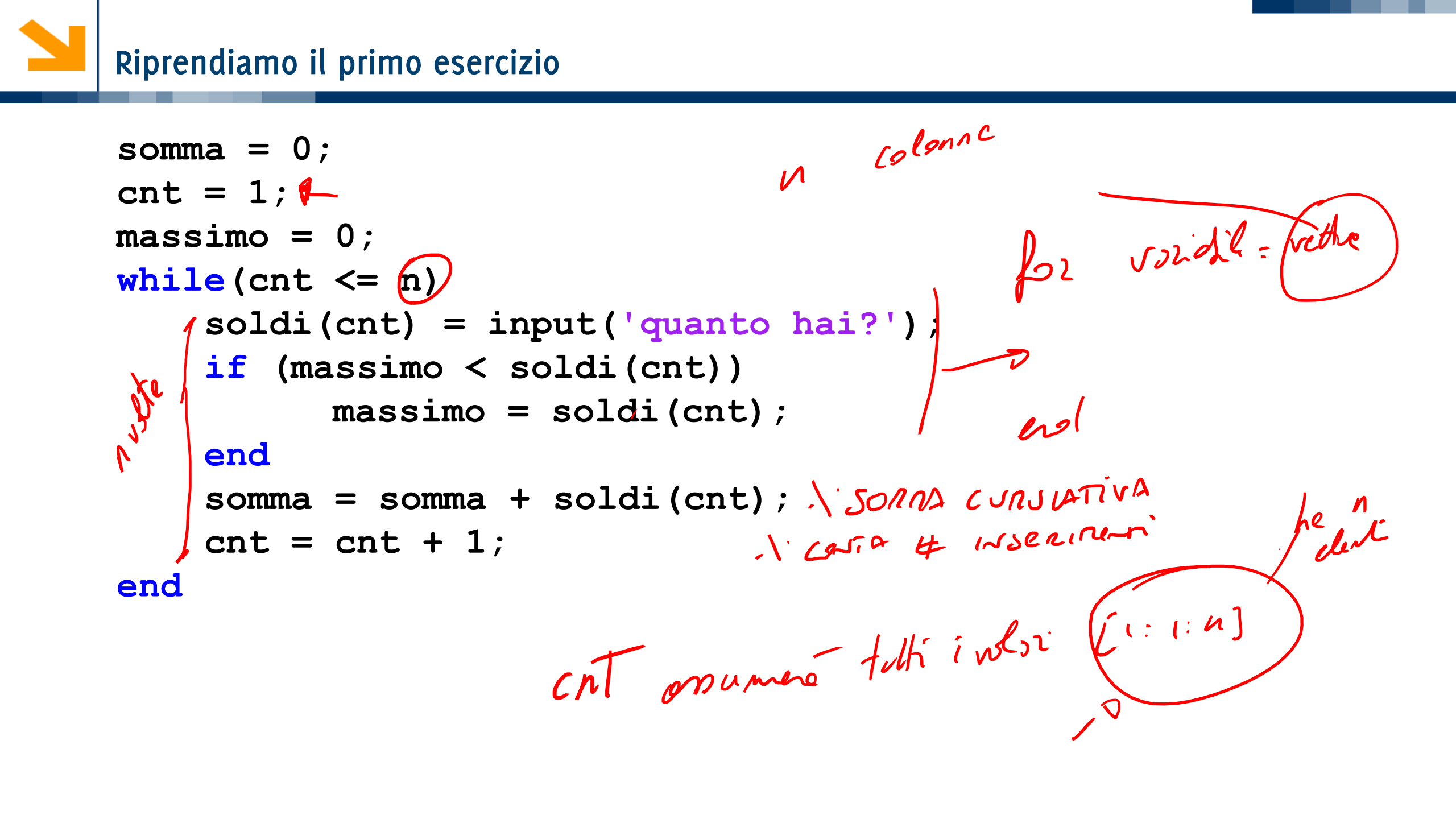
`s` varrà 50 la prima volta, 45 la seconda volta, poi 23

Vedrò a schermo:

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

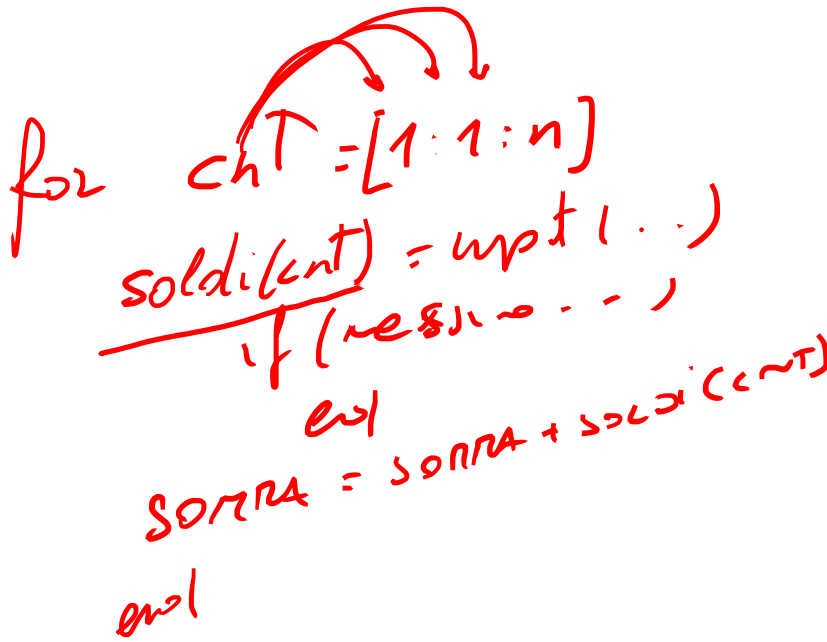
Riprendiamo il primo esercizio

```
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
```

for x = vettore

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 ≡ foreach

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

10 20 30

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

```
c = 1
while (c < 20)
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
```

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:
```

```
% chiedi all'utente di inserire una durata in secondi e
stampa il alla rovescia
```

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
```

input in formato Matlab

Stampa dei valori dell'array

fprintf permette comunque di stampare gli array, usando i fattori di conversione per gli scalari. L'istruzione

```
fprintf(' %d ', vettore);
```

Itera la stampa ' %d ' per ogni elemento di **vettore**

Altrimenti, per avere maggiore controllo, occorre iterare manualmente su ogni elemento

```
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
```

Operazioni su Array

Definizione ed Estensione Automatica di Array

Un **assegnamento** in una posizione in cui il vettore non è definito **estende l'array** inserendo o

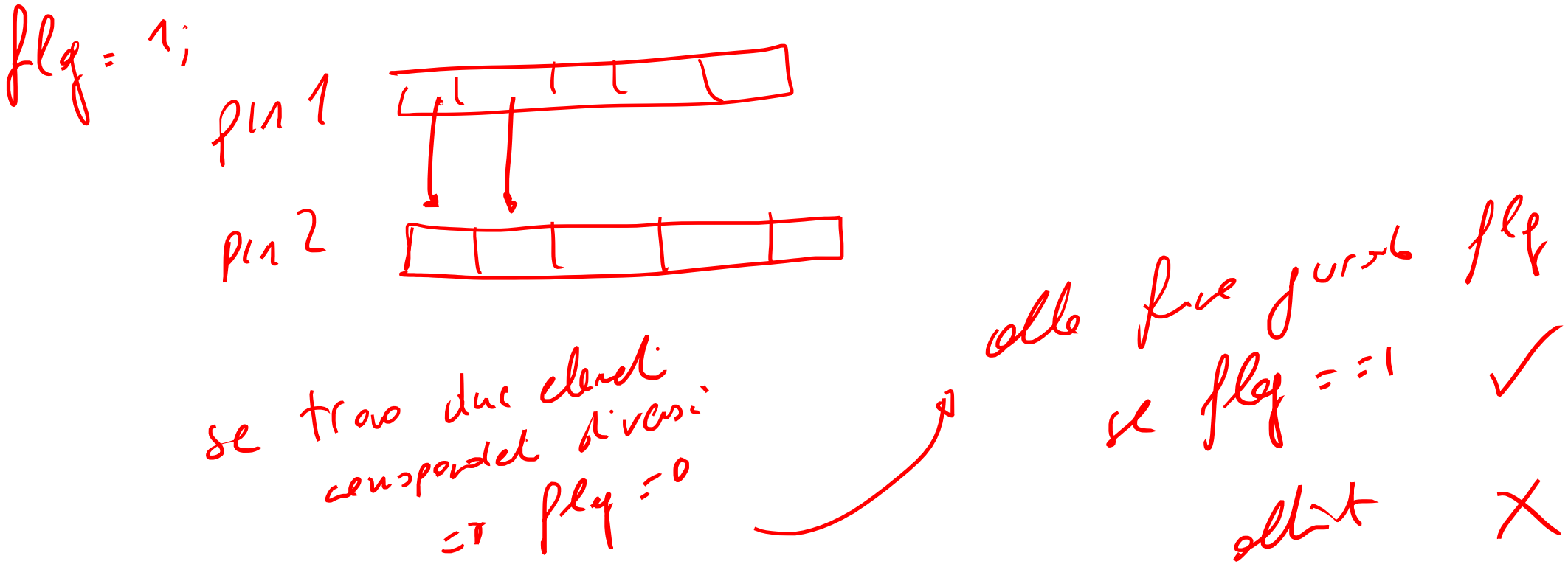
```
>> c = 1
c =
     1

>> c(3) = 3
c =
     1     0     3

>> c(2,3) = 5
c =
     1     0     3
     0     0     5
```

N.B. Assegnare un valore ad un elemento è diverso da accedere

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** ~= 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] ~= 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 (**ugualli**) 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 && ugualli == true)
        if(v1(ii) ~= v2(ii))
            ugualli = false;
        else
            ugualli = 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`

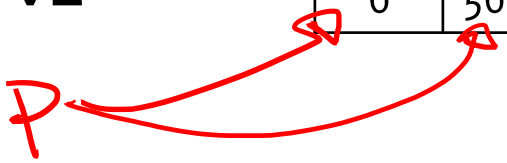
5	6	7	89	568	68	657	989	96	98

`v2` ✖

0	6	0	0	568	68	0	0	96	98

`v2`

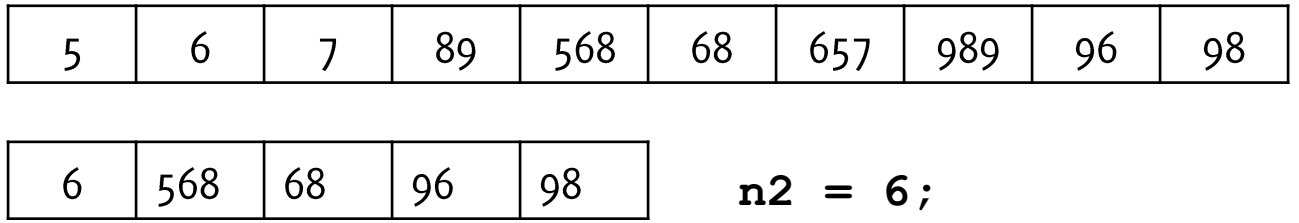
6	568	68	96	98



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**



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 v1
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]
```

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]
```

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

- `a = [0 7+1];`
- `b = [a(2) 5 a];`

contenuto di a

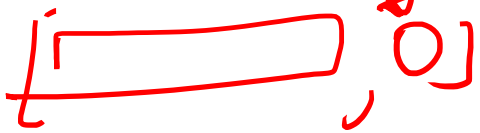
secondo elemento di a

Risultato

- `a = [0 8]`
- `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
```



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: **`c = a + b`**
  - E' definita elemento per elemento

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

è possibile solo quando **`a`** e **`b`** hanno la stesa dimensione (che poi coincide con quella di **`c`**)

Prodotto tra vettori:

- È il prodotto riga per colonna, restituisce uno scalare

`c = a * b`, i.e. $c = \sum_i a(i)b(i)$

`a` deve essere un vettore riga e **`b`** colonna e devono avere lo stesso numero di elementi, **`c`** è un numero reale

Operazioni Puntuali

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

`c = a .* b`, restituisce $c(i) = a(i) * b(i)\ \forall i$

`c = a ./ b`, restituisce $c(i) = a(i)/b(i)\ \forall i$

`c = a .^ b`, 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

`c = k * b = k .* b` $c(i) = k * b(i)\ \forall i$

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' rifermento 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	risultato(i,j)=a(i,j)/b(i,j)
Array left division	a .\ b	risultato(i,j)=b(i,j)/a(i,j)
Matrix right division	a / b	a*inversa(b)
Matrix left division	a \ b	inversa(a)*b
Array exponentiation	a .^ b	risultato(i,j)=a(i,j)^b(i,j)

Stringhe

Stringhe

Le stringhe sono array di caratteri. Per l'assegnamento delimito i caratteri tra apici singoli

```
msg = 'ciao';
```

›› whos msg

Name	Size	Bytes	Class	Attributes
msg	1x4	8	char	

Posso modificare gli elementi della stringa come faccio normalmente con i vettori

Manipolazione di stringhe

Esempi:

```
>> msg = 'ciao mamma';
>> msg = [msg , ' torno per cena']
msg =
ciao mamma torno per cena
```

Concatenazione di stringhe

```
>> msg(1) = 'C'
msg =
Ciao mamma torno per cena
```

Modificare i valori di una stringa

Stringhe: array di caratteri

Con **disp**, come per **fprintf**, è possibile inserire i valori di alcune variabili all'interno del testo visualizzato

Per le stringhe basta concatenare la variabile

```
nome = 'Giacomo'

disp(['ciao, ', nome])

ciao, Giacomo
```

Per le variabili numeriche occorre `num2str` che trasforma un valore numerico in sequenza di caratteri

```
nome = 'Giacomo'
anno = 1
disp(['ciao, ', nome, ' ho questo corso da anni: ', num2str(anno)])
```

Cosa fa?

```
for c = 'ciao'
        disp(c)
end
```

Cosa fa?

```
for c = 'ciao'
      disp(c)
end
```

Stampa a schermo i caratteri all'interno di 'ciao' uno per riga

Se volessi non andare a capo dopo ogni carattere userei:

```
for x = 'ciao'
    fprintf('%c', x);
end
```

Confronto tra Stringhe

Per confrontare due stringhe è possibile procedere:

- Confronto elemento per elemento
- Utilizzando la funzione **`strcmp(s1 , s2)`** che restituisce true o false (comoda anche la funzione **`strcmpi`** che è case-insensitive)
- Operazioni vettoriali (vedremo nella seconda parte)

Esercizio

Scrivere un programma che determina se due parole contengono le stesse vocali nello stesso ordine

Es: *mamma* e *anna* contengono le stesse vocali
cosa e *caso* non contengono le stesse vocali

Hint:

1. Estrarre, da ogni parola, un vettore contenente le vocali
2. Confrontare i due vettori delle vocali

Soluzione

```
% richiedere parole all'utente
p1 = input('inserire parola1: ', 's');
p2 = input('inserire parola2: ', 's');

v1 = [];
jj = 1;
% creo vettore v1 contenente le vocali di p1
for ii = 1 : length(p1)
    if (p1(ii) == 'a' || p1(ii) == 'e' || p1(ii) == 'o' || p1(ii) == 'i' ||
p1(ii) == 'u')
       v1(jj) = p1(ii); jj = jj + 1; % oppure v1 = [v1 , p1(ii)];
    end
end
disp(['le vocali di ', p1, ' sono ', v1])

%% controlla se v1 coincide con v2 (NEXT SLIDE)
```

Soluzione

```
%% controlla se v1 coincide con v2

if(length(v1) == length(v2))
    flag = true;
    ii = 1;
    while(ii < length(v1))
        if(v1(ii)  ~= v2(ii))
            flag = false;
        end
        ii = ii + 1;
    end
end

if flag
    fprintf('\n %s e %s hanno le stesse vocali\n', p1, p2);
else
    fprintf('\n %s e %s NON hanno le stesse vocali\n', p1, p2);
end
```

Esercizio

Scrivere un programma che determina se una parola è palindoma