Matlab: Logicals e Strutture di Controllo

Informatica B AA 2017/2018

Luca Cassano

luca.cassano@polimi.it

8 Novembre 2017

Tipo di Dato Logico

…e operazioni su vettori

Tipo di Dato Logico

È un tipo di dato che può avere solo due valori

- **`true`** (vero) 1
- **`false`** (falso) 0

Tipo di Dato Logico

È un tipo di dato che può avere solo due valori

- **`true`** (vero) 1
- **`false`** (falso) 0

I valori di questo tipo possono essere generati

- direttamente da due funzioni speciali (true e false)
- dagli operatori relazionali
- dagli operatori logici

I valori logici occupano un solo byte di memoria (i numeri ne occupano 8)

Operatori Relazionali

Operano su tipi numerici o array/stringhe.

Possono essere usati per confrontare

- due scalari
- due vettori aventi la stessa dimensione

Forma generale: a OP b

- a,b possono essere espressioni aritmetiche, variabili, array/stringhe (della stessa dimensione)
- OP: `==, ~=, >, >=, <, <=`

Esempi:

- `3 < 4` true(1)
- `3 == 4` false(0)
- `'A' < 'B'` true(1)

Note

Come in C: non confondere == e =

- == è un operatore di confronto
- = è un operatore di assegnamento

Note

Come in C: non confondere == e =

- == è un operatore di confronto
- = è un operatore di assegnamento

La precisione finita può produrre errori con == e ~ =

- sin(0) == 0 → 1
- sin(pi) == 0 → 0
- eppure logicamente sono vere entrambe!!

Note

Come in C: non confondere == e =

- == è un operatore di confronto
- = è un operatore di assegnamento

La precisione finita può produrre errori con == e ~ =

- sin(0) == 0 → 1
- sin(pi) == 0 → 0
- eppure logicamente sono vere entrambe!!

Per i numeri piccoli conviene usare una soglia

- abs(sin(pi)) < = eps

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Il risultato di un confronto tra v1 e v2 è un vettore v3 di tipo boolean, aventi le stesse dimensioni di v1 (e v2)

v3 = v1 OP v2 <=> v3(i) = v1(i) OP v2(i) per ogni i

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Il risultato di un confronto tra v1 e v2 è un vettore v3 di tipo boolean, aventi le stesse dimensioni di v1 (e v2)

v3 = v1 OP v2 <=> v3(i) = v1(i) OP v2(i) per ogni i

v3 = (v1 >= v2); $v3(i) = \begin{cases} 1, & \text{se } v1(i) \geq v2(i) \\ 0, & \text{se } v1(i) < v2(i) \end{cases}$

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Il risultato di un confronto tra v1 e v2 è un vettore v3 di tipo boolean, aventi le stesse dimensioni di v1 (e v2)

v3 = v1 OP v2 <=> v3(i) = v1(i) OP v2(i) per ogni i

v3 = (v1 >= v2); $v3(i) = \begin{cases} 1, & \text{se } v1(i) \geq v2(i) \\ 0, & \text{se } v1(i) < v2(i) \end{cases}$

Esempi:

>> [1 0; -2 1] < 0 **[false false; true false]**

>> [1 0; -2 1] >= [2 -1; 0 0] **[false true; false true]**

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Il risultato di un confronto tra v1 e v2 è un vettore v3 di tipo boolean, aventi le stesse dimensioni di v1 (e v2)

v3 = v1 OP v2 <=> v3(i) = v1(i) OP v2(i) per ogni i

v3 = (v1 >= v2); $v3(i) = \begin{cases} 1, & \text{se } v1(i) \geq v2(i) \\ 0, & \text{se } v1(i) < v2(i) \end{cases}$

Si possono confrontare stringhe di lunghezza uguale

```
>> 'pippo'=='pluto'
ans = [1 0 0 0 1]
```

Operatori Logici: Forma Generale

Operatori binari:

- **AND** (**`&&`**, oppure **`&`**, oppure **`and`**)
- **OR** (**`||`**, oppure **`|`**, oppure **`or`**),
- **XOR** (**`xor`**):

`a OP1 b` per la notazione simbolica
`OP(a,b)` per la notazione testuale

Operatori unari:

- **NOT** (**`~`**):

`OP2 a`

Operatori Logici: Forma Generale

```
a OP1 b
```

a,b possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)

Valori numerici di **a**, **b** vengono interpretati come logici:

- 0 come falso
- tutti i numeri diversi da 0 come vero

`&&` e `||` funzionano con gli scalari

`&` e `|` funzionano sia con scalari che con vettori

Richiamo Tabelle di Verità

a	b	a && b	a || b	~a	`xor`(a, b)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Or esclusivo: vero quando è vera solo uno delle due espressioni coinvolte

Precedenze tra gli Operatori

Ogni espressione logica viene valutata rispettando il seguente ordine:

- operatori aritmetici
- operatori relazionali da sinistra verso destra
- NOT (`~`)
- AND (`&` e `&&`) da sinistra verso destra
- OR (`|` e `||`) e XOR da sinistra verso destra

Vettori Logici per Selezionare

I vettori con valori logici possono essere usati per selezionare gli elementi di un array al posto di un vettore di indici

```
nomeVettore(vettoreLogico)
```

vengono estratti gli elementi di **`nomeVettore`** alle posizioni per cui **`vettoreLogico`** vale 1

Per esempio

Vettori Logici per Selezionare

I vettori con valori logici possono essere usati per selezionare gli elementi di un array al posto di un vettore di indici

`nomeVettore(vettoreLogico)`

vengono estratti gli elementi di **`nomeVettore`** alle posizioni per cui **`vettoreLogico`** vale 1

Per esempio

```
>> x = [6,3,9]; y = [14,2,9];
>> b = x <= y ; % b = 1     0     1
>> z = x(b)
z = 6     9
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Visualizzare solamente i numeri maggiori di 10

```
>> a(a > 10);
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Visualizzare solamente i numeri maggiori di 10

```
>> a(a > 10)
```

Portare a zero tutti gli elementi negativi

```
>> a(a < 0) = 0;
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Visualizzare solamente i numeri maggiori di 10

```
>> a(a > 10);
```

Portare a zero tutti gli elementi negativi

```
>> a(a < 0) = 0;
```

Sommare 10 ai numeri minori di 10

```
>> a(a < 10) = a(a < 10) + 10;
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Visualizzare solamente i numeri maggiori di 10

```
>> a(a > 10);
```

Portare a zero tutti gli elementi negativi

```
>> a(a < 0) = 0;
```

Sommare 10 ai numeri minori di 10

```
>> a(a < 10) = a(a < 10) + 10;
```

Cambiare il segno a tutte le occorrenze di -7 o 17

```
>> a(a == -7 | a == 17) = -a(a == -7 | a == 17);
```

Note

`nomeVettore` e `vettoreLogico` devono avere la **stessa dimensione**

Note

`nomeVettore` e **`vettoreLogico`** devono avere la **stessa dimensione**

Per **creare un vettore logico non basta** creare un **vettore di 0 e 1** (numeri), bisogna convertirlo con la funzione logical

Note

`nomeVettore` e **`vettoreLogico`** devono avere la **stessa dimensione**

Per **creare un vettore logico non basta** creare un **vettore di 0 e 1** (numeri), bisogna convertirlo con la funzione logical

```
>> ii = [1,0,0,0,1];
>> jj = logical(ii); %oppure jj = (ii == 1)
>> A = [1 2 3 4 5];
>> A(jj) → [1 5]
>> A(ii) → Subscript indices must either
     be real positive integers or logicals.
```

Strutture di Controllo

Costrutto Condizionale: `if`, la sintassi

Il costrutto condizionale permette di eseguire istruzioni a seconda del valore di un'espressione booleana

`if, else, elseif, end` keywords

`expression` espressione booleana (vale 0 o 1)

`statement` sequenza di istruzioni da eseguire.

NB: il corpo è delimitato da **`end`**

NB: indentatura irrilevante

```
if(expression)
      statement
end
```

```
if(expression1)
      statement1
elseif(expression2)
      statement2
else
      statement0
end
```

Il Costrutto if

```
if espressione1
      istruzione 1-1
      istruzione 1-2
      ..........
elseif espressione2
       istruzione 2-1
        istruzione 2-2
       .......... .....
else
      istruzione k-1
      istruzione k-2
      ..........
 end
```

Le istruzioni 1-1 e 1-2 vengono eseguite solo se vale espressione 1

Le istruzioni 2-1 e 2-2 vengono eseguite solo se non vale espressione1 ma vale espressione2

Le istruzioni k-1 e k-2 vengono eseguite solo se non vale nessuna delle espressioni sopra indicate

rami elseif e else non sono obbligatori!

Il Costrutto if

`espressione1` può coinvolgere vettori:

- in tal caso **`espressione1`** è vera solo se tutti gli elementi di **`espressione1`** sono non nulli (quindi veri)

Esempio

Il Costrutto if

espressione1 può coinvolgere vettori:

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

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
```

Esercizio

Scrivere un programma che richiede all'utente di inserire un numero e determina se corrisponde ad un anno bisestile

È possibile usare la funzione **`mod(a,b)`** che restituisce il resto della divisione tra **`a`** e **`b`**

Un anno è bisestile se è multiplo di 4 ma non di 100 oppure se è multiplo di 400

Soluzione

```
n = input('inserire anno ');
div_4 = (mod(n , 4) == 0);
div_100 = (mod(n , 100) == 0);
div_400 = (mod(n , 400) == 0);

bisestile = ((div_4) && ~(div_100)) || (div_400);

stringa_output = num2str(n);
if(bisestile == 0)
    stringa_output = [stringa_output , ' non è '];
else
    stringa_output = [stringa_output ,' è '];
end
stringa_output = [stringa_output , 'bisestile'];
disp(stringa_output);
```

Esercizio

Scrivere un programma che richiede all'utente di inserire una stringa e controlla se questa è palindroma

Esercizio

Scrivere un programma che richiede all'utente di inserire una stringa e controlla se questa è palindroma

```
parola = input('inserire parola ' , 's');
if (parola == parola(end : -1 : 1))
    disp([parola , ' è palindroma'])
else
    disp([parola , ' NON è palindroma'])
end
```

Il Costrutto switch

```
switch variabile %scalare o stringa
  case valore1
   istruzioni caso1
  case valore2
   istruzioni caso2
  ...
  otherwise
   istruzioni per i restanti casi
end
```

L'istruzione condizionale switch consente una scrittura alternativa ad if/elseif/else

Qualunque struttura switch può essere tradotta in un if/elseif/else equivalente

Note

Come per il C:

- **`valore1`** etc… devono essere delle espressioni costanti e si confrontano con **`variabile`** per verificarne l'uguaglianza

Note

Come per il C:

- **`valore1`** etc… devono essere delle espressioni costanti e si confrontano con **`variabile`** per verificarne l'uguaglianza

A differenza del C:

- solamente un caso viene eseguito: quando **`variabile`** corrisponde ad uno specifico **`valore`** non si eseguono tutti gli statement in cascata, si esce dal ciclo (è come se ci fosse sempre un break)
- è inutile usare break

Note

Come per il C:

- **`valore1`** etc… devono essere delle espressioni costanti e si confrontano con **`variabile`** per verificarne l'uguaglianza

A differenza del C:

- solamente un caso viene eseguito: quando **`variabile`** corrisponde ad uno specifico **`valore`** non si eseguono tutti gli statement in cascata, si esce dal ciclo (è come se ci fosse sempre un break)
- è inutile usare break
- è possibile confrontare vettori
  - Sebbene **`variabile`** venga confrontata con **`valore1`** non è richiesto che queste abbiano la stessa lunghezza
  - Il case viene eseguito se tutti gli elementi corrispondono

Note

È possibile mettere più valori nel case, contenuti in graffe

```
str = 'pluto';
switch str
    case {'pippo', 'pluto', 'paperino',
'clarabella'}
        disp('Walt Disney')
    otherwise
        disp('no Walt Disney')
end
```

In questo caso basta che ci sia un match tra str e un elemento tra le parentesi graffe

Il Ciclo while

```
while expr
  istruzioni da ripetere finché expr è
  vera
end
```

Il Ciclo while

```
while expr
  istruzioni da ripetere finché expr è
  vera
end
```

expr assume valore 0 o 1e può contenere con operatori relazionali (==, <, >, <=, >=, ~=) e logici (&&, &, ||, |, ~)

Il Ciclo while

```
while expr
  istruzioni da ripetere finché expr è
  vera
end
```

expr assume valore 0 o 1e può contenere con operatori relazionali (==, <, >, <=, >=, ~=) e logici (&&, &, ||, |, ~)

expr deve essere inizializzata (avere un valore) prima dell'inizio del ciclo

Il Ciclo while

```
while expr
  istruzioni da ripetere finché expr è
  vera
end
```

expr assume valore 0 o 1e può contenere con operatori relazionali (==, <, >, <=, >=, ~=) e logici (&&, &, ||, |, ~)

expr deve essere inizializzata (avere un valore) prima dell'inizio del ciclo

Quando **expr** coinvoge vettori si ha che **expr** è vera se tutti gli elementi sono non nulli (e quindi veri) (come per if)

Il Ciclo while

```
while expr
  istruzioni da ripetere finché expr è
  vera
end
```

expr assume valore 0 o 1e può contenere con operatori relazionali (==, <, >, <=, >=, ~=) e logici (&&, &, ||, |, ~)

expr deve essere inizializzata (avere un valore) prima dell'inizio del ciclo

Quando **expr** coinvoge vettori si ha che **expr** è vera se tutti gli elementi sono non nulli (e quindi veri) (come per if)

Il valore di espressione deve cambiare nelle ripetizioni (altrimenti si ha un ciclo infinito)

Esempio

Stampare, utilizzando un ciclo i numeri da 100 a 1

Esempio

Stampare, utilizzando un ciclo i numeri da 100 a 1

```
n = 100;
while(n > 0)
    disp(n);
    n = n - 1;
end
```

Esempio

Stampare, utilizzando un ciclo i numeri da 100 a 1

```
n = 100;
while(n > 0)
    disp(n);
    n = n - 1;
end
```

In alternativa

```
disp([100 : - 1 : 1]');
```

Esempio

Calcolare gli interessi di un capitale di 1000 fino al raddoppio del capitale; si assuma un interesse annuo del 8%

Esempio

Calcolare gli interessi di un capitale di 1000 fino al raddoppio del capitale; si assuma un interesse annuo del 8%

```
value = 1000;
original_value = value;
year = 0;
while (value < original_value * 2)
    value = value * 1.08
    year = year + 1;
    fprintf('%f years: $%f\n', year, value)
end
```

Esempio

Il quadrato di N è uguale alla somma dei primi N numeri dispari. Calcolare il quadrato di un n inserito da utente (<100)

Esempio

Il quadrato di N è uguale alla somma dei primi N numeri dispari. Calcolare il quadrato di un n inserito da utente

```
n = input('inserire n');
priminumeri = [0 : n - 1];
disp = 2 * priminumeri + 1;
somma  = 0;
ii = 1;
while(ii <= n)
    somma = somma + disp(ii);
    ii = ii + 1;
end
disp(['il quadrato di ' , num2str(n) , ' è '
, num2str(somma)]);
```

Esempio

Richiedere all'utente di inserire un numero e, se questo corrisponde ad un anno bisestile, chiederne un altro. Il programma termina quando viene inserito un numero che non corrisponde ad un anno bisestile.
Al termine, il programma scrive quanti anni bisestili ha inserito l'utente

Soluzione

```
bisestile = 1;
counter = 0;
while(bisestile)
```

```
end
disp(['game over. Hai inserito ' , num2str(counter) , ' bisestili'])
```

Soluzione

```
bisestile = 1;
counter = 0;
while(bisestile)
    n = input('inserire anno ');
    div_4 = (mod(n , 4) == 0);
    div_100 = (mod(n , 100) == 0);
    div_400 = (mod(n , 400) == 0);
    bisestile = ((div_4) && ~(div_100)) || (div_400);




end
disp(['game over. Hai inserito ' , num2str(counter) , ' bisestili'])
```

Soluzione

```
bisestile = 1;
counter = 0;
while(bisestile)
    n = input('inserire anno ');
    div_4 = (mod(n , 4) == 0);
    div_100 = (mod(n , 100) == 0);
    div_400 = (mod(n , 400) == 0);
    bisestile = ((div_4) && ~(div_100)) || (div_400);

    stringa_output = num2str(n);
    if(bisestile == 0)
        stringa_output = [stringa_output , ' non è '];
    else
        stringa_output = [stringa_output ,' è '];
        counter = counter + 1;
    end
    stringa_output = [stringa_output , 'bisestile'];
    disp(stringa_output);
end
disp(['game over. Hai inserito ' , num2str(counter) , ' bisestili'])
```

Il ciclo for

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

- Il numero di iterazioni dipende dal numero di elementi di **`array`** (spesso è un vettore, può anche essere una matrice)
- 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

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** (non è vero il contrario)

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

c assumerà ad ogni iterazione un carattere diverso nel vettore 'ciao'

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** (non è vero il contrario)

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

```
vet = 'ciao'
ii = 1;
while (ii <=length(vet))
   disp(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** (non è vero il contrario)

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

```
vet = 'ciao'
ii = 1;
while (ii <=length(vet))
  disp(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** (non è vero il contrario)

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

```
vet = 'ciao'
ii = 1;
while (ii <=length(vet))
   disp(vet(ii))
   ii = ii + 1;
end
```

Per scorrere un vettore noto, il ciclo for è molto più comodo, 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 è un vettore riga** definito tramite l’operatore di incremento regolare, i.e., “inizio : step : fine”

Esempi

```
% leggi 7 numeri e mettili in un vettore:
```

Esempi

```
% leggi 7 numeri e mettili in un vettore:
   for n = 1:7
      number(n) = input('enter value ');
   end
```

Esempi

```
% leggi 7 numeri e mettili in un vettore:
   for n = 1:7
      number(n) = input('enter value ');
   end

% stampa conto alla rovescia in secondi
```

Esempi

```
% leggi 7 numeri e mettili in un vettore:
   for n = 1:7
      number(n) = input('enter value ');
   end

% stampa conto alla rovescia in secondi
   time = input('how many seconds? ');
   for count = time:-1:1
       pause(1);
       fprintf('%f seconds left \n',count);
   end
   disp('done');
```

Il ciclo for , la variabile del ciclo (2)

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

Quando **`array`** è una matrice, il ciclo viene eseguito un numero volte pari al numero di colonne di **`array`** e ogni volta **`variabile`** assume il valore di una colonna

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

N.B. Quando **`array`** è un vettore colonna, questo viene considerato una matrice e si esegue una sola iterazione in cui **`variabile`** è uguale ad **`array`**

Il ciclo for , la variabile del ciclo

Esempio di for su una **una matrice**

```
board = [ 1 1 1 ; 1 1 -1 ; 0 1 0 ];
for x = board
    disp('colonna:')
    x %stampa in ogni iterazione una colonna di board
end
```

```
colonna:

x =

     1

     1

     0

colonna:
x =

     1

     1

     1

colonna:

x =

     1

    -1

     0
```

Esercizio

Si assuma che il vettore voti contenga i risultati del primo appello

- Si calcoli la media dei voti
- Si calcoli la media dei voti sufficienti (>= 18)
- Si calcoli il numero di voti maggiori o uguali a 15

Esempio

Si calcoli la media

```
voti = input('inserisci il vettore dei voti ');
somma = 0;
for v = voti
      somma = somma + v;
end
media = somma / length(voti);
disp(['la media dei voti è ', num2str(media)]);
```

Esempio

Si calcoli la media dei voti sufficienti (>=18)

```
somma = 0;
aux = voti >= 18;
suff = voti(aux);
for v = suff
        somma = somma + v;
end
media = somma / length(voti);
disp([‘la media dei voti sufficienti è ‘,
num2str(media)]);
```

Esempio

Si calcoli la media dei voti sufficienti (>=18) ALTERNATIVA

```
somma = 0;
for v = voti(voti >= 18)
      somma = somma + v;
end
media = somma / length(voti);
disp([‘la media dei voti sufficienti è ‘,
num2str(media)]);
```

Esempio

Si calcoli il numero di voti maggiori o uguali a 15

```
aux = voti >= 15;

suff = voti(aux);

str = [‘il numero di voti >= 15 è ‘,
num2str(length(suff))];

disp(str);
```

Esempio

Si calcoli il numero di voti maggiori o uguali a 15 ALTERNATIVA

```
disp(['il numero di voti >= 15 è ',
num2str(length(voti(voti >= 15)))]);
```