Matlab:
Logicals e Strutture di Controllo

Informatica B a.a. 2021/2022

Francesco Trovò

26 Ottobre 2021

francesco1.trovo@polimi.it

Tipo di Dato Logico

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)

Esempi

```
>> a = true;
>> whos a
 Name   Size    Bytes  Class     Attributes
  a     1x1       1    logical
```

a è un vettore 1x1 che occupa 1 byte e appartiene alla classe "tipo logico"

```
>> a = 1 > 7
a =
     0
```

Operatori Relazionali

- Operano su tipi numerici o 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, stringhe (della stessa dimensione)
  - OP: `==, ~=, >, >=, <, <=`

Es.

- `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

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) - 0 ) < = eps`

Operatori Relazionali tra Vettori

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 >= v2);
```

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

Es.

```
>> v1 = [5 4 3 2 1];
>> v2 = [1 2 3 4 5];
>> v3 = v1 >= v2
        [1 1 1 0 0]
```

Operatori Relazionali tra Vettori

- Si possono confrontare stringhe di lunghezza uguale

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

- Il confronto vale anche tra matrici della stessa dimensione

```
>> [1 0; -2 1] >= [2 -1; 0 0]
ans = [false true; false true]
```

- Come per l'assegnamento, è possibile confrontarsi con uno scalare, il confronto avviene tra tutti gli elementi e lo scalare

```
>> [1 0; -2 1] < 0
ans = [false false; true false] ([0 0; 1
0])
```

Operatori Logici

- Gli operatori logici (**and, or, not**) sono definiti su array di logicals in **maniera puntuale**
- Gli operatori unari (**not**) possono essere eseguiti
  - Su array di logicals
- Gli operatori binari (**and, or**) possono essere eseguiti
  - Su array di logicals aventi le stesse dimensioni

Es.

```
>> v1 = [5 4 3 2 1], v2 = [1 2 3 4 5];
>> v3 = v1 > 3 & v2 >= 2;
(>> v3 = [1 1 0 0 0] & [0 1 1 1 1])
   v3 = [0 1 0 0 0]
```

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`

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

- Valori numerici di **`a`**, **`b`** vengono interpretati come logici:
  - **`0`** come falso
  - tutti i numeri diversi da **`0`** come vero

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
a XOR b == a OR b AND (~(a AND b))

&& vs & e || vs |

`&&` (**`||`**) **funziona con gli scalari** e valuta prima l'operando più a sinistra

Se questo è sufficiente per decidere il valore di verità dell'espressione non va oltre

- **`a && b`**: se **`a`** è falso non valuta **`b`**
- **`a || b`**: se **`a`** è vero non valuta **`b`**

`&` (**`|`**) **funziona con scalari e vettori** e valuta **tutti** gli operandi prima di valutare l'espressione complessiva

Es. **`a / b > 10`** (se **`b`** è zero non voglio eseguire la divisione)

- **`(b~=0) && (a/b>10)`** è la soluzione corretta: **`&&`** controlla **`b~=0`**, se questo è falso non valuta **`a/b>10`**
- Invece **`(b~=0) & (a/b>10)`** porterebbe ad una divisione per 0 quando **`b == 0`**

Esempi

"Hai tra 25 e 30 anni?"

```
(eta >= 25) & (eta <= 30)
```

Con i vettori:

```
Voto = [12, 15, 8, 29, 23, 24, 27]
 C = (Voto > 22) & (Voto < 25)
-> C = [0 0 0 0 1 1 0]
 D = (mod(Voto,2) == 0) | (Voto > 18)
-> D = [1 0 1 1 1 1 1]
 E = xor((mod(Voto,2)==0), (Voto>18))
-> E = [1 0 1 1 1 0 1]
```

Utile per contare quanti elementi soddisfano una condizione

```
nVoti= sum(Voto > 22 & Voto < 25)
```

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 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]
```

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);
```

NB qui non si può usare ||

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

Costrutti Condizionali

Costrutto Condizionale: `if`, la sintassi

- Il costrutto condizionale permette di eseguire istruzioni a seconda del valore di un'espressione booleana
- **`if, else, elseif, end`** sono keywords riservate
- **`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 expr1
   istruzione 1.1
   istruzione 1.2
elseif expr2
        istruzione 2.1
        istruzione 2.2
else
   istruzione 3.1
   istruzione 3.2
end
```

Le istruzioni 1.1 e 1.2 vengono eseguite solo se vale expr1

Le istruzioni 2.1 e 2.2 vengono eseguite solo se non vale expr1 ma vale expr2

Le istruzioni 3.1 e 3.2 vengono eseguite solo se non vale nessuna delle espressioni sopra indicate

NB: i rami elseif e else non sono obbligatori!

Il Costrutto if

- **expr1** può coinvolgere vettori:
  - in tal caso **expr1** è vera solo se tutti gli elementi di **expr1** sono non nulli

Occorre inserire il vettore tra parentesi quadre

Es.

```
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`**

Si ricorda che un anno è bisestile se è multiplo di 4 ma non di 100 oppure se è multiplo di 400

(Storia: Il calendario gregoriano si applica dal 1582, anno della sua introduzione. Benché sia possibile estenderlo anche agli anni precedenti, per questi si usa il calendario giuliano. Perciò sono bisestili tutti gli anni divisibili per 4, compresi quelli secolari, dal 4 al 1580 dell'era volgare. Per gli anni precedenti l'era volgare, invece, non si applicano gli anni bisestili, visto che Ottaviano Augusto regolò l'applicazione degli anni bisestili nell'8 a.C.)

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);
```

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

Differenze con il C

Come per il C:

- **`valore1`**, **`valore2`**, 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

Case Multipli

- È possibile mettere più valori nel case, separati da 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

Costrutti Iterativi

Il Ciclo while

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

- **expr** assume valore 0/1 e può contenere operatori relazionali (==, <, >, <=, >=, ~=)
- **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 (come per **if**)

NB: il valore di espressione deve cambiare nelle ripetizioni

Esempio

Stampare, utilizzando un ciclo i numeri da 100 a 1

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

Delle due soluzioni la seconda è preferibile in Matlab perché più efficiente

In alternativa

```
[100 : - 1 : 1]'
```

Esercizio

Calcoliamo gli interessi composti fino al raddoppio del capitale, si assuma un interesse annuo del 8%

```
value = 1000;
year = 0;
while value < 2000
    value = value * 1.08
    year = year + 1;
    fprintf('%f years: $%f\n', year, value)
end
```

Esercizio

Calcolare il quadrato di un numero inserito da utente, sapendo che il quadrato di N è uguale alla somma dei primi N numeri dispari

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

Homework

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

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

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

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

Scrittura meno compatta dello stesso ciclo

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

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`**

Esercizi

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

% stampa conto alla rovescia in secondi
partendo da un valore scelto dall'utente
   time = input('how long? ');
   for count = time:-1:1
       pause(1);
       fprintf('%d seconds left \n', count);
   end
   disp('Boom!!!');
```

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)`**

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

Esempio

```
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
```

Nota sulla Creazione di Array

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

Questa soluzione non permette all'utente di inserire:

```
>> enter value [1 13]
```

```
In an assignment  A(I) = B, the number of
elements in B and I must be the same
```

Per risolvere questo problema si può acquisire l'input in una variabile temporanea **`temp`** (non un elemento di un vettore) e poi accodare il contenuto di **`temp`** ad una variabile accumulatore

Array Accumulatore

```
% leggi 7 numeri e mettili in un vettore
vettore = [];
for ii = [1 : 7]
    temp = input('inserire numero ');
    vettore = [vettore, temp];
end
disp(vettore)
```

- È necessario inizializzare **`vettore`** a vuoto, altrimenti la prima esecuzione del ciclo genera un errore perché **`vettore`** non esiste
- L'indice **`ii`** in questo caso non è più necessario, serve solo perché il ciclo si ripeta per il numero corretto di volte

- **`[vettore, temp];`** ⟶ accodo al vettore
- **`[temp, vettore];`** ⟶ metto in testa

Break e Continue

- I cicli contengono una serie di istruzioni che vogliamo ripetere
- Però potremmo aver bisogno di:
  - Saltare all'iterazione successiva
  - Terminare il ciclo
- Come nel C:
  - `continue` salta all'iterazione successiva
  - `break` interrompe l'esecuzione del ciclo
- Come nel C sono da evitare, perché sono sostituibili da variabili di flag

Esercizio

Acquisiamo numeri da tastiera finché non viene inserito un numero negativo (non accettiamo più di 1000 numeri)

```
vector = [ ]; %crea il vettore vuoto
for count = 1:1000  %Raccoglierà al max 1000 valori
      value = input('Next number ');
      if value < 0
          break  %Se value negativo usciamo dal ciclo
      else
          vector(count) = value;
      end
end
vector %visualizza il contenuto di vector
```

Esercizio

Stampare solo la parte triangolare bassa delle tabelline

```
T; %matrice delle tabelline
fprintf('\ntriangolare bassa\n');
for r = 1 : size(T, 1)
    for c = 1 : r
        fprintf('%4d', T(r, c));
    end
    fprintf('\n');
end
```

Restituisce le dimensioni di una matrice

Esercizio

Stampare solo la parte triangolare alta delle tabelline

```
T; %matrice delle tabelline
fprintf('\ndiagonale alta\n');
for r = 1 : size(T, 1)
    for c = 1 : size(T, 2)
        if r <= c
            fprintf('%4d', T(r, c));
        else
            fprintf('    ');
        end
    end
    fprintf('\n');
end
```

Homework

Si scriva un programma che prende in ingresso una matrice **quadrata** e controlla che sia simmetrica

Definizione: una matrice è simmetrica se per ogni elemento vale la seguente proprietà: l'elemento alla riga i, colonna j coincide con l'elemento alla riga j colonna i

Es.

1	**12**	**1**
12	**0**	**3**
1	**3**	**23**

Homework

Scrivere un programma che acquisisce un vettore di interi definito dall'utente e quindi calcola

- Il numero di multipli di 8
- Il numero di multipli di 4
- Il numero di multipli di 2

Presenti nel vettore e quindi stampa un istogramma (verticale) per visualizzare le occorrenze

- *Es.* dato `[8 12 1 3 2]`

```
      *
    * *
* * *
- - -
8 4 2
```