Strutture di Controllo in Matlab e Algebra di Boole

Informatica (ICA) AA 2020 / 2021

Giacomo Boracchi

29 Settembre 2020

giacomo.boracchi@polimi.it

Esempio

Scrivere un programma che richiede di inserire la lunghezza di tre lati e determina se questi corrispondono ad i lati di un triangolo

- La condizione è che ciascun lato deve essere minore della somma degli altri due e maggiore della loro differenza.

In caso in cui i lati identifichino un triangolo il programma determina se tale triangolo è:

- Equilatero
- Isoscele
- Scaleno
- Rettangolo

Matlab: Costrutto Condizionale

Istruzioni composta: **`if`**, **`switch`**

Costrutto Condizionale: `if`, la sintassi

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

`if, else, end` keywords

`expression` espressione booleana (vale 0 o 1)

`statement` sequenza di istruzioni da eseguire (corpo).

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

NB: indentatura irrilevante

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

```
if(expression1)
      statement1
else
      statement0
end
```

Costrutto Condizionale: `if`, l'esecuzione

1. Terminata **`instrBefore`**, valuto **`expression`**,
2. Se **`expression`** è vera (**`!=0`**), allora eseguo **`statement1`**, altrimenti eseguo **`statement0`**. (se è presente **`else`**)
3. Terminato lo statement dell'**`if`** , procedi con **`instrAfter`**, la prima istruzione fuori dall'**`if`**

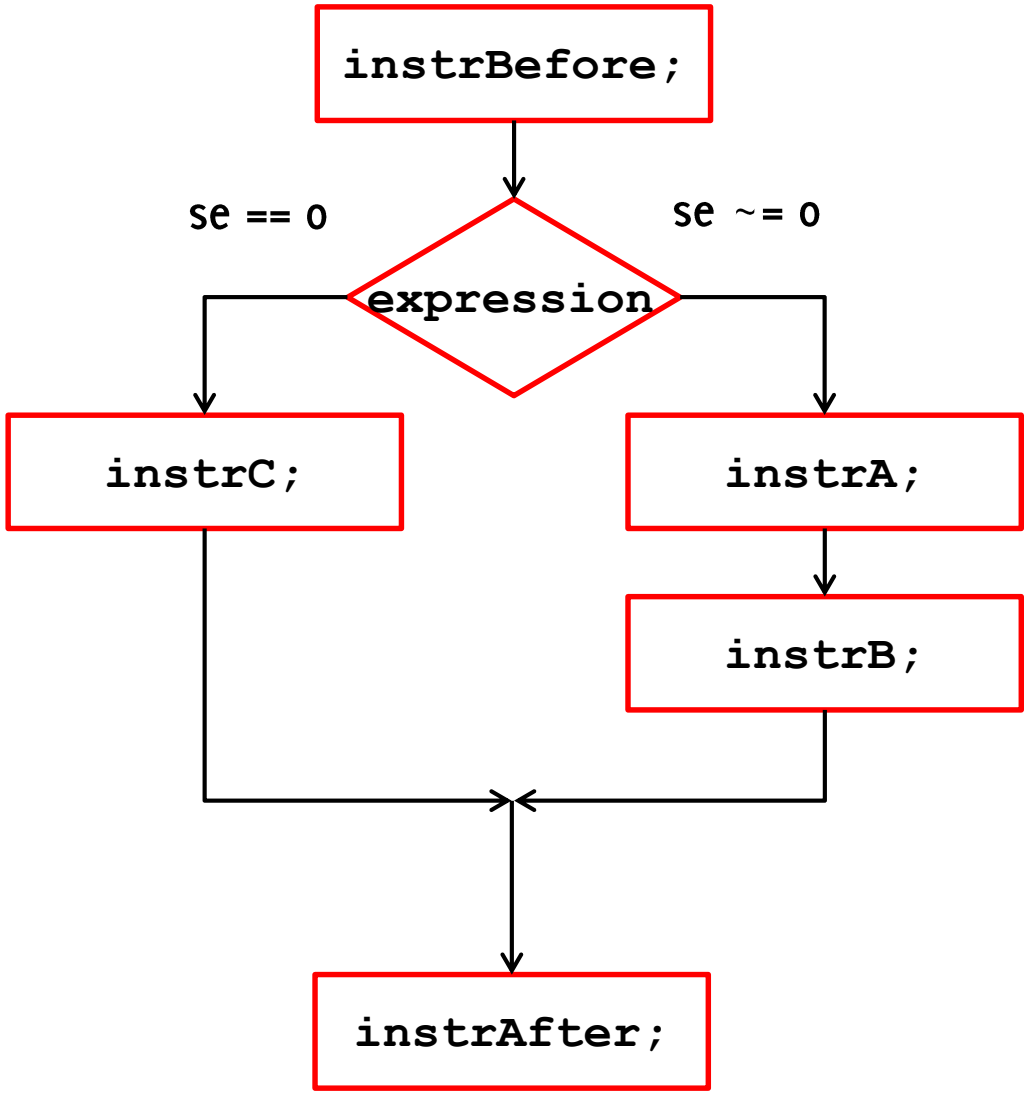
N.B. **`else`** è opzionale

N.B **`if(expression)`** **non** richiede il **`;`** perché l'istruzione non termina dopo **`)`**

```
instrBefore;
if(expression)
        statement1;
else
        statement0;
end
instrAfter;
```

Costrutto Condizionale: `if`, l'esecuzione

```
instrBefore;
if(expression)        instrA;
      instrB;
else
      instrC;
end
instrAfter;
```



`if` Annidati

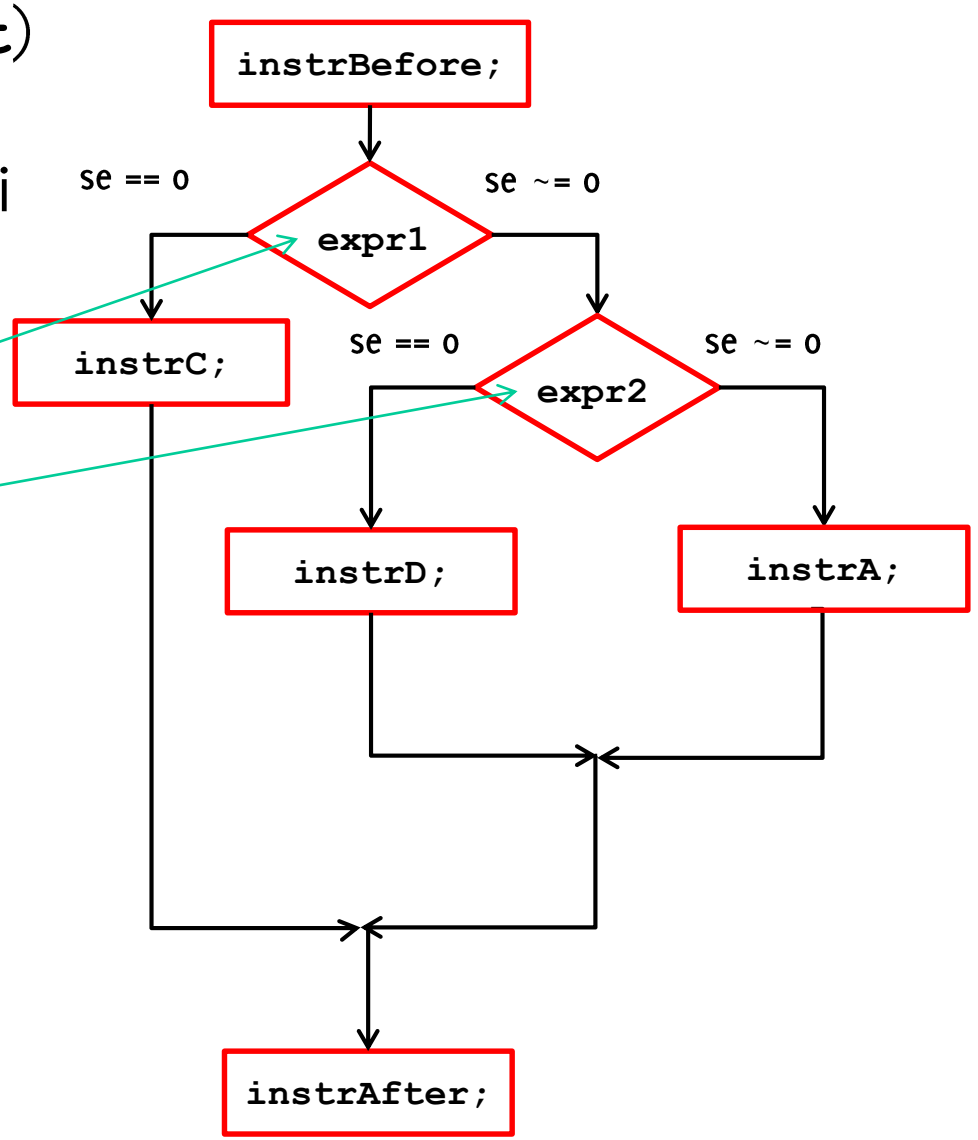
Il corpo di un **`if`** (cioè uno **`statement`**)
può a sua volta contenere costrutti **`if:`**
si realizzano quindi istruzioni condizionali
annidate

```
instrBefore;
if(expr1)
 if(expr2)
     instrA;
  else
     instrD;
  end
else
 instrC;
end
instrAfter;
```

`if` Annidati

Il corpo di un **`if`** (cioè uno **`statement`**) può a sua volta contenere costrutti **`if:`**
si realizzano quindi istruzioni condizionali **annidate**

```
instrBefore;
if(expr1)
 if(expr2)
     instrA;
  else
     instrD;
  end
else
 instrC;
end
instrAfter;
```



`if` Annidati

Le istruzioni condizionali possono essere annidate, inserendo un ulteriore **`if`** all'interno di **`statement1`** o **`statement0`**

```
if(mod(x,7) ==0)
    fprintf('%d è multiplo di 7', x);
else
    if(mod(x,5) == 0)
        fprintf('%d NON è mutiplo di 7 ma di 5', x);
    else
        fprintf('%d NON è multiplo di 7 e nemmeno di 5', x);
    end
end
```

`if` Annidati

È possibile sostituire if annidati con sequenze di if con condizioni composte

```
x = input('inserire x: ');

if(mod(x,7) ==0)
    fprintf('%d è multiplo di 7', x);
end

if(mod(x,7) ~=0) && (mod(x,5) ==0)
        fprintf('%d NON è multiplo di 7 ma di 5', x);
end

if(mod(x,7) ~=0) && (mod(x,5) ~=0)
        fprintf('%d NON è multiplo di 7 e nemmeno di 5', x);
end
```

Valutare una condizione nell'else: `elseif`

`elseif` permette di valutare un'ulteriore condizione nell ramo **`else`** senza dover annidare un secondo **`if`**

Il corpo dell' **`elseif`** viene eseguito se **`expression1`** è falsa ed **`expression2`** è vera

Se è falsa sia **`expression1`** che **`expression2`** allora eseguo **`statement0`**, il corpo dell' **`else`**

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

Il Costrutto if in Generale

```
if espressione1
   istr_1a
   istr_1b
   ..........
elseif espressione2
    istr_2a
    istr_2b
    ..........
else
   istr_ka
   istr_kb
   ..........
end
```

Le **istr_1a** e **istr_1b** vengono eseguite solo se vale espressione 1

Le **istr_2a** e **istr_2b** vengono eseguite solo se non vale espressione1 ma vale espressione2

Le **istr_ka** e **istr_bka** vengono eseguite solo se non vale nessuna delle espressioni sopra indicate

elseif e **else** non sono obbligatori!

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

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

Esempio

Scrivere un programma che richiede all'utente due operandi (**a,b**) ed un carattere (**OP**) e, se **OP** corrisponde ad un operatore (**'+', '-', '*', '/', '^'**) calcola il risultato di **a OP b,** altrimenti solleva un messaggio di errore.

Nel caso di divisione per zero viene anche mandato un messaggio di errore

Altri Costrutti

`break, continue`

Teorema di Boehm-Jacopini

istruzioni **`if`** e **`while`** (e la possibilità di eseguire istruzioni in sequenza) sono equivalenti a istruzioni che la macchina di Von Neumann che può manipolare registro Contatore di Programma

istruzioni **`if`** e **`while`** sono complete:

➡ bastano per codificare qualsiasi algoritmo

Per praticità e convenienza si usano però molte altre strutture di controllo

`break` e `continue`

L'istruzione **`break`** termina l'esecuzione di un costrutto iterativo

L'istruzione **`continue`** all'interno di un costrutto iterativo passa direttamente all'iterazione seguente, interrompendo quella corrente.

Cosa fa?

```
ii = 0;
while(ii < 10)
    x = input('\ninserire x:');
    if(x < 0)
        break;
    end
    fprintf('%d', x);
    ii = ii + 1
end
```

Cosa fa?

```
ii = 0;
while(ii < 10)
    x = input('\ninserire x:');
    if(x < 0)
        break;
    end
    fprintf('%d', x);
    ii = ii + 1
end
```

Richiede fino a 10 numeri e ne stampa il valore inserito. Le acquisizioni terminano anticipatamente se viene inserito un valore negativo.

Cosa fa?

```
ii = 0;
while(ii < 10)
    x = input('\ninserire x:');
    if(x < 0)
        continue;
    end
    fprintf('%d', x);
    ii = ii + 1
end
```

Cosa fa?

```
ii = 0;
while(ii < 10)
    x = input('\ninserire x:');
    if(x < 0)
        continue;
    end
    fprintf('%d', x);
    ii = ii + 1;
end
```

Richiede fino a 10 numeri e ne stampa il valore inserito. Le acquisizioni **non** terminano se viene inserito un valore negativo, però non viene stampato il valore inserito (il `continue` fa saltare alla successiva esecuzione)

Alternative a `break` e `continue`

Utilizzo di cicli con variabili **flag (o sentinella)** per terminare anticipatamente l'esecuzione del ciclo

Una variabile che assume un valore 0 / 1 a seconda che si verifichino o meno alcune condizioni durante l'esecuzione

Esempio: Alternativa e break e continue

Scrivere un ciclo che richiede una serie di valori interi e li associa alla variabile intera **a** e stampa a schermo

- non più di N numeri inseriti
- saltando i valori negativi inseriti (vengono calcolati per raggiungere N)
- interrompendo l'elaborazione al primo valore nullo incontrato

Esempi con continue e break

```
ii = 0;
while(ii < N)
    n = input('immetti un intero>0 ');
    ii = ii + 1;
    if (n < 0)
        continue;
    end
    if (n == 0)
        break;
    end
    fprintf('%d', n);
    % elabora i positivi
end
```

MOLTO IMPORTANTE: come farne a meno

```
ii = 0;
flag = 1; % diventa 0 quando inserisco un negativo
while(ii < N) && flag
    n = input('immetti un intero>0 ');
    ii = ii + 1;
    if (n == 0)
        flag = 0;
    elseif(n > 0)
        fprintf('%d',n);
        % elabora i positivi
    end
end
```

Importanza delle variabili di flag

Scrivere un ciclo con che richiede una serie di valori interi e li associa alla variabile intera `a` e stampa a schermo

- non più di 10 richieste
- saltando i valori negativi inseriti
- interrompendo l'elaborazione al primo valore nullo incontrato
- **Al termine, stampare un messaggio qualora fossero stati inseriti 10 numeri positivi**

MOLTO IMPORTANTE: come farne a meno

```
ii = 0;
flag = 1; % diventa 0 quando inserisco un negativo
while(ii < N) && flag
    n = input('immetti un intero>0 ');
    ii = ii + 1;
    if (n == 0)
        flag = 0;
    elseif(n > 0)
        fprintf('%d',n);
        % elabora i positivi
    end
end
if flag == 1
    fprintf('tutti i numeri sono non nulli')
end
```

MOLTO IMPORTANTE: come farne a meno

```
ii = 0;
flag = 1; % diventa 0 quando inserisco un negativo
while(ii < N) && flag
    n = input('immetti un intero>0 ');
    ii = ii + 1;
    if (n == 0)
        flag = 0;
    elseif(n > 0)
        fprintf('%d',n);
        % elabora i positivi
    end
end
if flag == 1
    fprintf('tutti i numeri sono non nulli')
end
```

se flag è rimasto uno vuol dire che nel ciclo sopra non è mai stato inserito un valore nullo, altrimenti sarebbe diventato o

MOLTO IMPORTANTE: come farne a meno

```
ii = 0;
flag = 1; % diventa 0 quando inserisco un negativo
while(ii < N) && flag
    n = input('immetti un intero>0 ');
    ii = ii + 1;
    if (n == 0)
        flag = 0;
    elseif(n > 0)
        fprintf('%d',n);
        % elabora i positivi
    end
end
if flag == 1
    fprintf('tutti i numeri sono non nulli')
end
```

Se avessi usato il break al posto della variabile di flag non avrei potuto determinare così facilmente se il ciclo sopra si fosse interrotto per via del break o se fosse terminato normalmente

Esercizio

```
% Scrivere un programma che determina se un numero n
inserito da utente è primo
```

Esercizio

```
% Scrivere un programma che richiede un intero all'utente un
intero M e stampa i primi M numeri primi
```

TODO

```
% Scrivere un programma che simula il lancio di un dado
10.000 volte e si mostri il numero di occorrenze di 1, 2, ..
,6 per fare vedere che il dado non è truccato

%hint: si usi randi(6 --oppure floor e rand(1)-- per
generare il lancio di un dado e quindi lo switch case e
diverse variabili contatori per conteggiare quante volte
esce ogni numero
```

Note

Confronto e Assegnamento

L'operatore di confronto **==** non va confuso con l'operatore di assegnamento **=**

Le loro sintassi sono simili

```
nomeVariabile == Espressione;
```

```
nomeVariabile = Espressione;
```

in entrambi i casi **Espressione** è una variabile/una costante/un valore fissato o un'espressione che coinvolge gli elementi sopra.

Il risultato del confronto
nomeVariabile == Espressione è **1**
se **nomeVariabile** ed **Espressione** coincidono.

Errori Frequenti

Confondere l'assegnamento con il confronto

```
a = 10;
if(a = 7)
    fprintf('Vero');
else
    fprintf('Falso');
end
```

```
if(a = 7)
    |
```

Error: The expression to the left of the equals sign is not a valid target for an assignment.

Il risultato di un assegnamento è un logical

```
b = '2';
a = b == '0';
fprintf('%d' , a);
```

In questo esempio **a** è una variabile di tipo logicals

Associa ad **a** il valore **1** se **b** è **0**, **1** altrimenti.

Viene letto

```
a = (b =='0');
```

Se `b = '2';` Stampa 0

Se `b = '0';` Stampa 1

Se `b = 0;` Stampa 0

&& 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

Esempio: a/b>10

- se b è 0 non voglio eseguire la divisione
- (b~=0) **&&** (a/b>10) è la soluzione corretta: && controlla prima b~=0 e se questo è falso non valuta il secondo termine. Invece (b~=0)&(a/b>10) potrerebbe ad una divisione per 0 quando b == 0

Scripts

Vantaggi/Svantaggi

Uno script può

- essere ri-eseguito
- essere facilmente modificato
- essere facilmente inviato

Uno script NON

- accetta variabili di input
- genera variabili di output

Uno script opera sulle variabili del workspace, che può essere arricchito introducendone di nuove durante l'esecuzione dello script stesso

Commenti

Il simbolo di commento può essere messo in qualsiasi punto della linea.

MATLAB ignorerà tutto quello che viene scritto alla destra del simbolo `%` .

Per esempio:

```
>> % This is a comment.
>> x = 2+3 % So is this.
x =
     5
```

Come Creare uno Script

Può essere creato utilizzando un qualsiasi editor di testo

- Ricordarsi di salvare il file come “solo testo” e di dare l’estensione .m
- Il file di **script** deve essere **presente nella directory corrente** o il **folder** contenente lo script deve **comparire nel path** di Matlab

Nomi degli Script

Il nome del file deve **iniziare con una lettera** e può contenere cifre e il carattere underscore, fino a 31 caratteri

Non dare lo stesso nome al file di script e a una variabile

Non chiamare uno script con lo stesso nome di un comando o funzione MATLAB.

Per verificare se esiste già qualcosa che ha un certo nome si può utilizzare la funzione exist.

Strutturare e Documentare uno Script

1. Sezione dei commenti:
   - Il nome del programma e le parole chiave, nella prima riga
   - La data di creazione e i nomi degli autori nella seconda riga
   - La definizione dei nomi delle variabili per ogni variabile di input e di output
   - Il nome di ogni funzione creata dall'utente che viene usata nel programma
   - Il comando help visualizza tutta la sezione dei commenti all'inizio dello script
2. Sezione di Input: inserimento dei dati in input e/o uso di funzioni di input
3. Sezione di calcolo
4. Sezione di output: uso di funzioni per visualizzare i risultati del programma

Dati su cui Opera Uno Script

Gli script non accettano argomenti d'entrata e d'uscita

Usano

- variabili già presenti nel workspace
- variabili acquisite da tastiera o file
- nuove variabili introdotte nello script

Le variabili interne allo script diventano variabili del workspace

- Permangono dopo l'esecuzione dello script

Sezione di Calcolo

Calcoli matematici

Assegnamenti

Strutture di controllo

- Condizioni
- Cicli

Comandi per la costruzione di grafici

Chiamate a funzioni

Comandi in Matlab

Esempio di alcuni comandi (analizzeremo quelli più importanti)

- Il prompt accetta i comandi del sistema operativo (DOS, UNIX…)
  - Esempio: in ambiente dos, dir mostra il contenuto della directory corrente
- help richiama la guida in linea
- diary può essere utilizzato per salvare la sessione di lavoro
- who, whos e workspace mostrano l'elenco delle variabili definite
- save permette di salvare in un file le variabili definite. Load le ricarica
- clear cancella tutte le variabili
- close chiude tutte le figure

Gli Array

Informatica (ICMR) AA 2020 / 2021

Giacomo Boracchi

9 Ottobre 2020

giacomo.boracchi@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

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 i un foglietto `soldi`, una **sequenze di foglietti**, ed ciascun foglietto tiene traccia dei valori inseriti

Quindi sequenze di variabili: gli array

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,.. etc.

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:

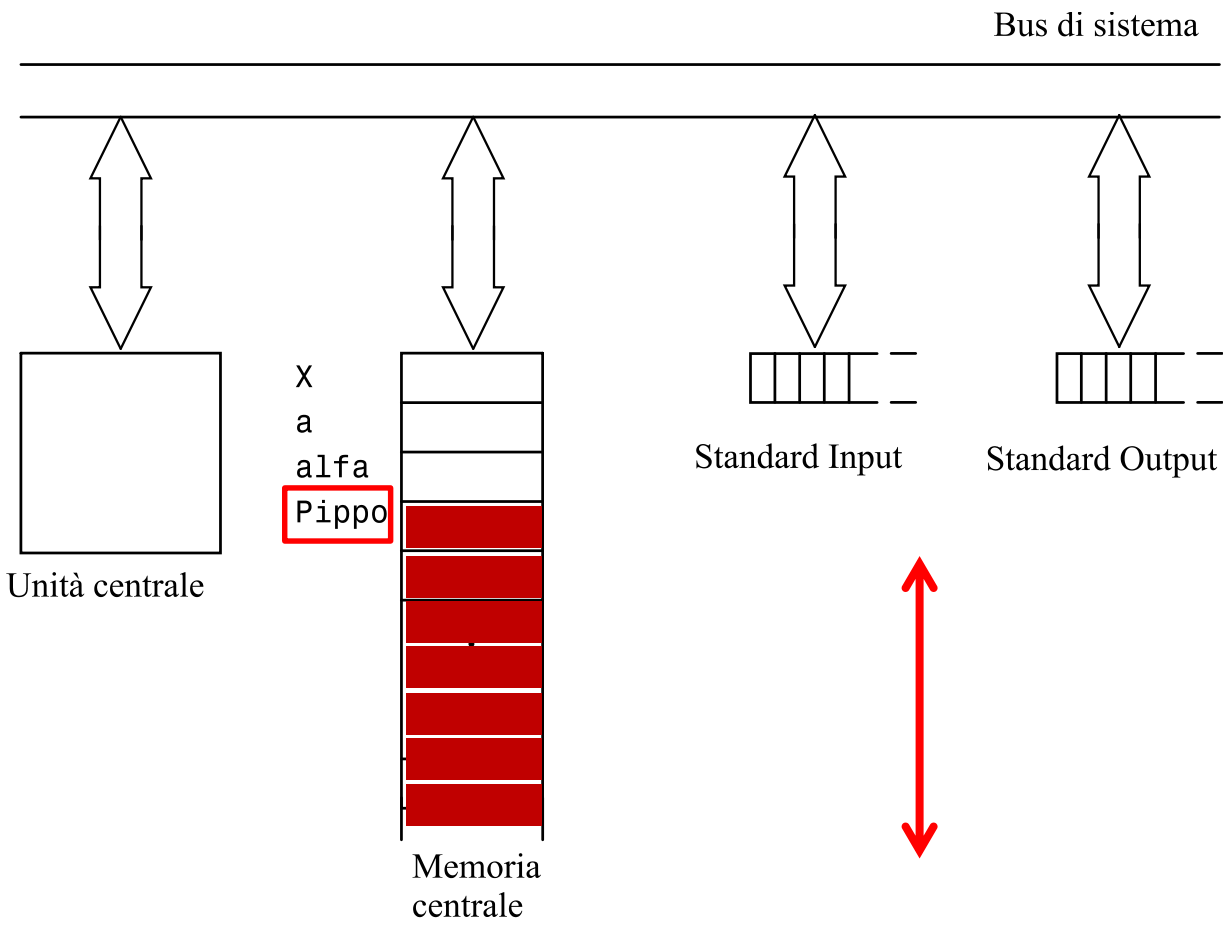
`vet(7)` conterrà un valore intero

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;
```

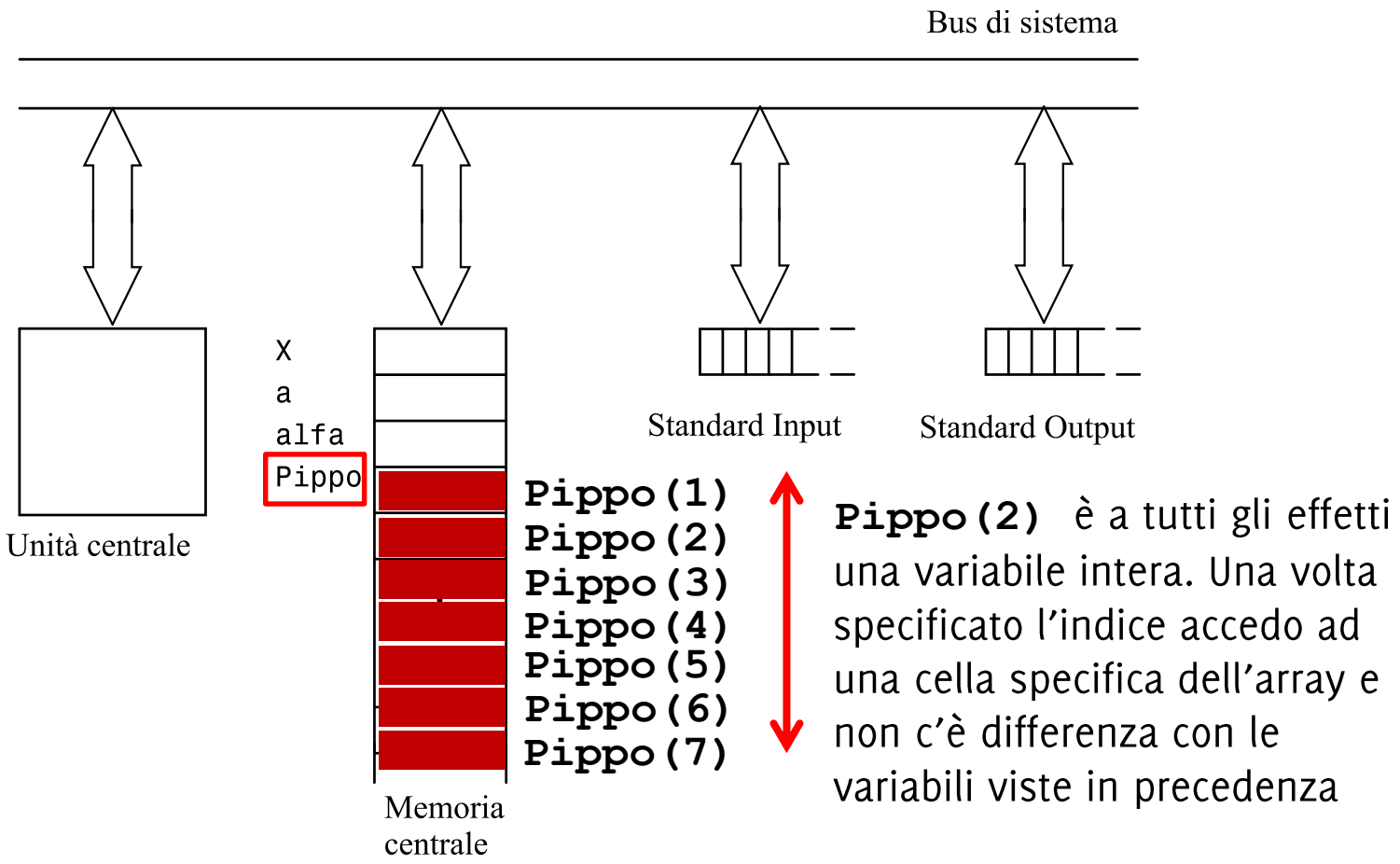
Lo spazio allocato per gli array

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



Lo spazio allocato per gli array

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



Pippo(2) è a tutti gli effetti una variabile intera. Una volta specificato l'indice accedo ad una cella specifica dell'array e non c'è differenza con le variabili viste in precedenza

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 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)

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 traminte 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** ~= 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 (**ugualiI**) 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 && ugualiI == true)
        if(v1(ii) ~= v2(ii))
            ugualiI = false;
        else
            ugualiI = 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

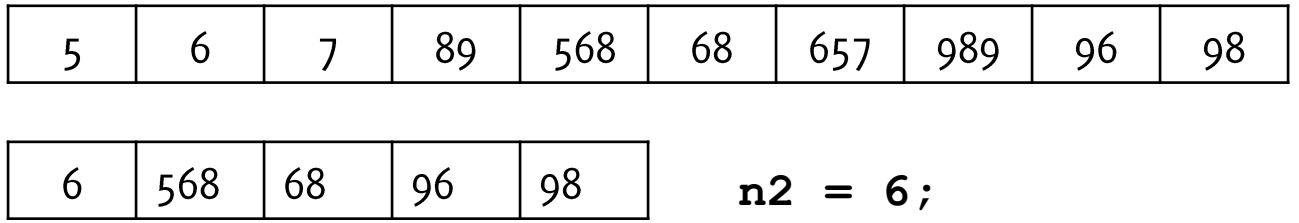
`v2`

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]
  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:

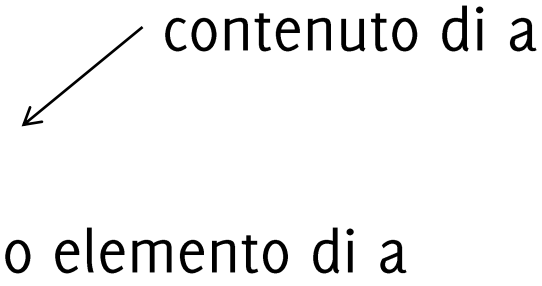
```
>> a = [1,2,3]
a =
   1     2     3
>> b = [a, a + 3 , a + 6]
b =
   1   2   3   4   5   6   7   8   9
>> b = [a, a +3]
b =
   1   2   3   1   2   3   3
```

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

Concatenare i Vettori

Esempi

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