



Strutture di Controllo in Matlab

Informatica (ICA) AA 2017 / 2018

Giacomo Boracchi

26 Settembre 2017

giacomo.boracchi@polimi.it



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

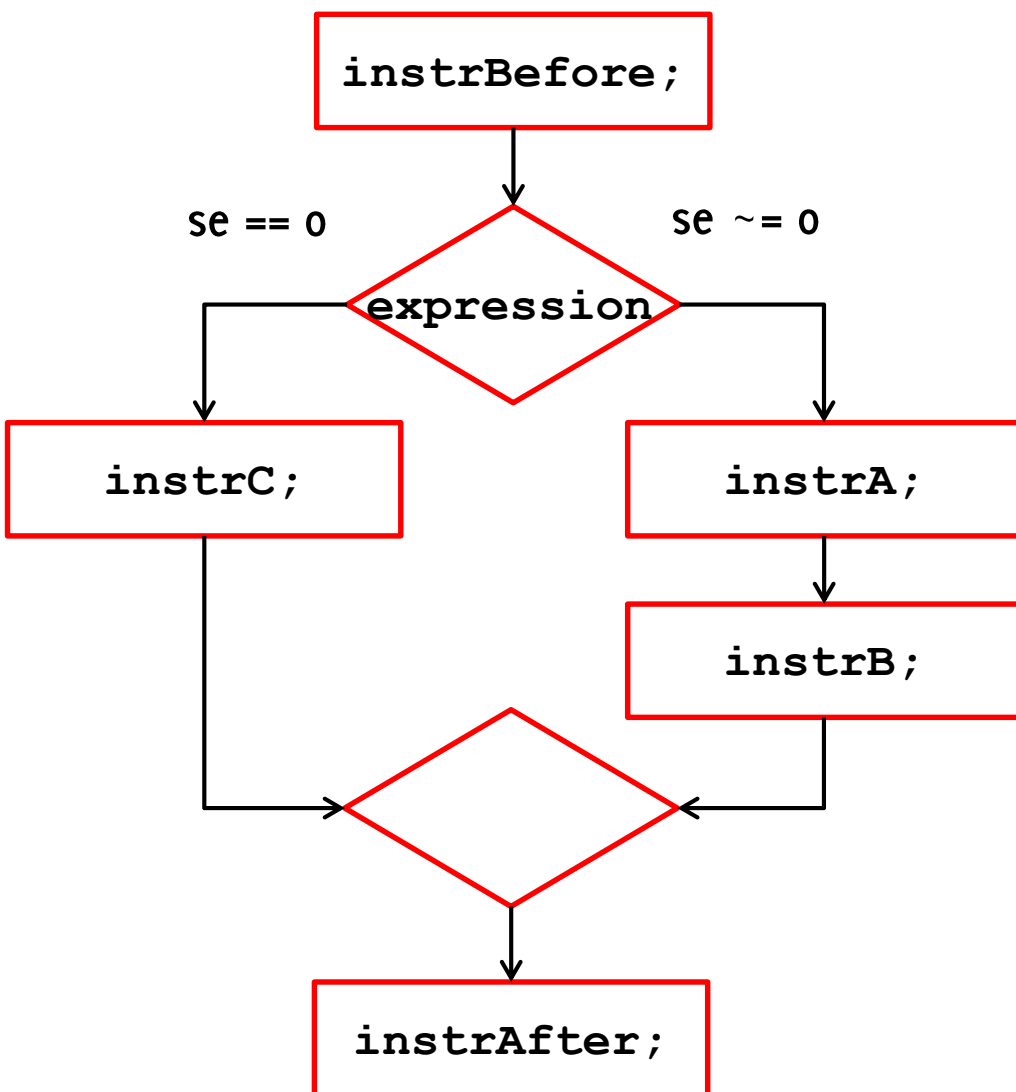
1. Terminata **instrBefore**,
valuto **expression**,
2. Se **expression** è vera ($\neq 0$),
allora eseguo **statement1**,
altrimenti eseguo **statement0**.
(se è presente **else**)
3. Terminato lo statement dell'**if**,
procedi con **instrAfter**, la
prima istruzione fuori dall'**if**

- 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;
```



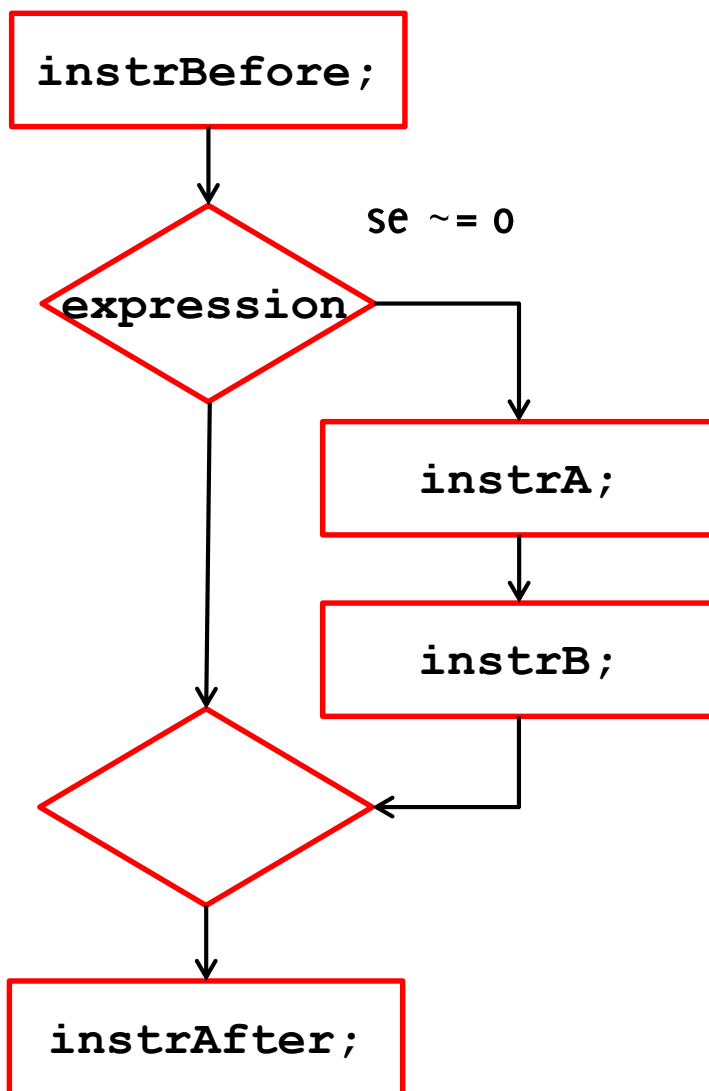
Costrutto Condizionale: `if`, l'esecuzione



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



Costrutto Condizionale: **if**, l'esecuzione



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



Esempio

%N.B: incolonnamento codice irrilevante!

```
if (mod(x,7) == 0)
    fprintf('%d multiplo di 7\n' , x) ;
else
    fprintf('%d non multiplo di 7\n' , x) ;
end
```



Esempio

%N.B: incolonnamento codice irrilevante!

```
if (mod(x,7) == 0)
    fprintf('%d multiplo di 7\n' , x) ;
else
    fprintf('%d non multiplo di 7\n' , x) ;
end
% posso fare senza else?
```




Esempio

%N.B: incolonnamento codice irrilevante!

```
if (mod(x,7) == 0)
    fprintf('%d multiplo di 7\n' , x) ;
else
    fprintf('%d non multiplo di 7\n' , x) ;
end

%senza else.
fprintf('%d ' , x) ;
if (mod(x, 7) ~= 0)
    fprintf('non ');
end
fprintf(' multiplo di 7\n');
```



if Annidati

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

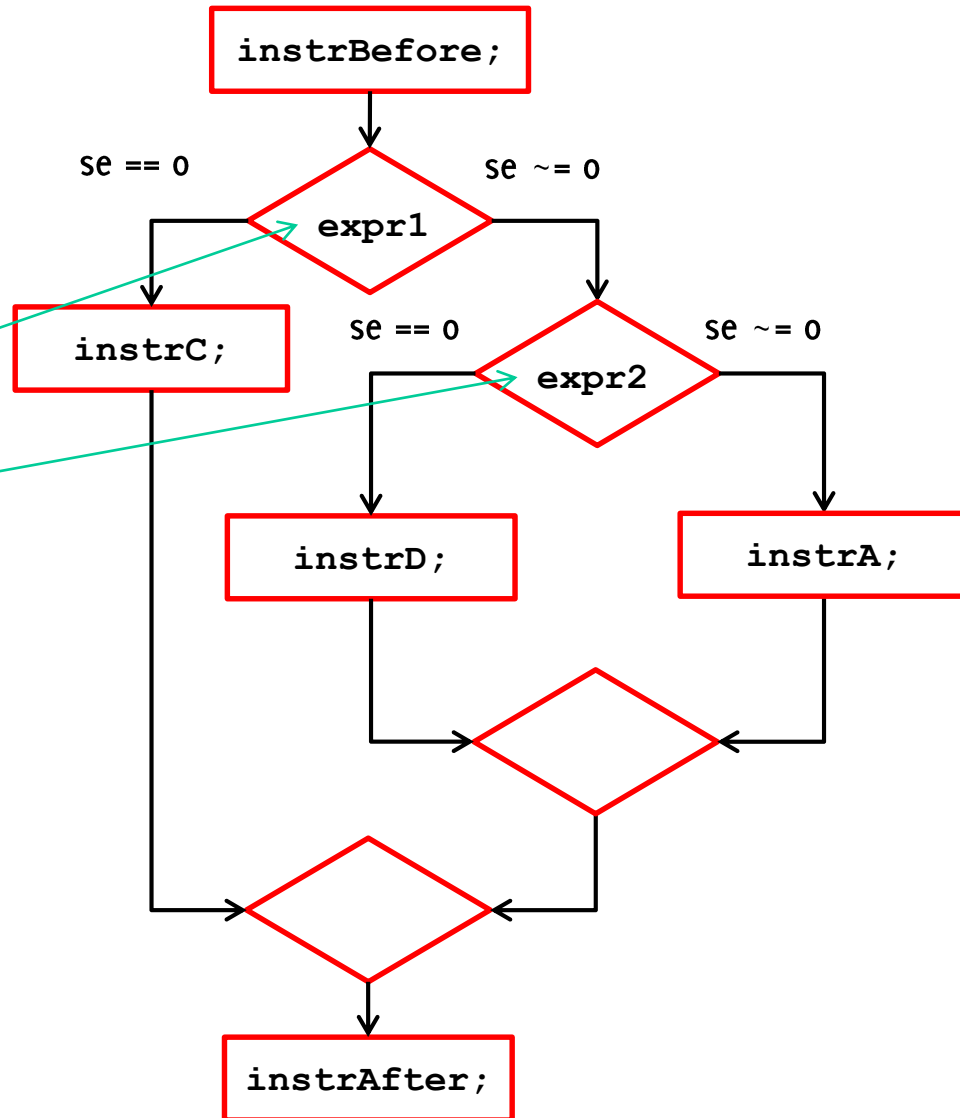
```
instrBefore;  
if (expr1)  
    if (expr2)  
        instrA;  
    else  
        instrD;  
    end  
else  
    instrC;  
end  
instrAfter;
```



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

.....

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

elseif espressione2

istr_2a

istr_2b

.....

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

else

istr_ka

istr_kb

.....

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

end

elseif e **else** non sono obbligatori!



Esempio

Scrivere un programma che determina se un numero inserito dall'utente è un intero positivo

(il comando **floor(n)** restituisce la parte intera di **n**)



```
x = input('inserisci x: ');
flag = 1; % questa variabile registra se x va bene

% x non va bene quando se x non è intero e positivo
if ~(x == floor(x) && x >= 0)
    flag = 0;
end

if flag == 1
    fprintf('\n %d intero positivo\n', x);
else
    fprintf('\n %d NON intero positivo\n', x);
end
```



```
x = input('inserisci x: ');
flag = 1; % questa variabile registra se x va bene

% x non va bene quando una di queste condizioni si
verifica
if x ~= floor(x) || x < 0
    flag = 0;
end
```

Questa condizione è equivalente alla precedente
grazie alle leggi di de Morgan

$$\sim (A \ \&\& \ B) = \sim A \ || \ \sim B$$

```
if flag == 1
    fprintf('\n %d intero positivo\n', x);
else
    fprintf('\n %d NON intero positivo\n', x);
end
```



Esempio

Scrivere un programma che richiede all'utente di scegliere un numero da 1 a 6, simula il lancio di un dado e quindi comunica se l'utente ha indovinato la puntata.

Per generare numeri casuali:

- `rand(1)` per generare un numero qualunque tra 0 ed 1 (tutti i numeri generati sono equiprobabili).
- Oppure, `randi(6)` genera un intero qualunque tra 1 e 6 (tutti i numeri generati sono equiprobabili)



```
x = input('scegli il nr 1-6 ');

if x ~= floor(x) || x <= 0 || x > 6
    fprintf('\ninserire un numero 1-6\n')
else
    % lancia il dado d
    d = randi(6);

    if(x == d)
        flag = true;
    else
        flag = false;
    end

    if flag
        fprintf('\nCOMPLIMENTI hai detto %d ed è uscito %d', x, d);
    else
        fprintf('\nPECCATO hai detto %d ed è uscito %d', x, d);
    end
end
```



Esempio

Scrivere un programma che determina il massimo tra tre numeri inseriti da tastiera



Soluzione 1: if Annidati

```
a = input('inserire numero :');  
b = input('inserire numero :');  
c = input('inserire numero :');
```

```
if (a > b)  
    if (a > c)  
        max = a;  
    else  
        max = c;  
    end  
else  
    if (b > c)  
        max = b;  
    else  
        max = c;  
    end  
end
```

```
fprintf('max(%d,%d,%d) = %d\n', a,b,c,massimo);
```

Osservazioni:

1. Il numero di indentazioni è n , pari a quanti numeri occorre controllare
2. Le parentesi negli if non sono necessarie qua



Soluzione 2: Condizioni composte

```
a = input('inserire numero :');  
b = input('inserire numero :');  
c = input('inserire numero :');
```

```
if (a>=b) && (a>=c)  
    massimo = a;  
end
```

```
if (b>=a) && (b>=c)  
    massimo = b;  
end
```

```
if (c>=a) && (c>=b)  
    massimo = c;  
end
```

```
fprintf('max(%d,%d,%d) = %d\n', a,b,c,massimo);
```

Osservazioni:

1. Condizioni composte si allungano quando si aggiungono numeri da controllare
2. Il numero di condizioni da valutare per n numeri è n
3. If usati in sequenza
4. E' necessario mettere $\geq$ altrimenti non gestisce correttamente il caso in cui almeno due numeri sono uguali



Soluzione 3: if in sequenza

```
a = input('inserire numero :');  
b = input('inserire numero :');  
c = input('inserire numero :');
```

```
massimo = a;  
if(massimo < b)  
    massimo = b;  
end
```

```
if(massimo < c)  
    massimo = c;  
end
```

```
fprintf('max(%d,%d,%d) = %d\n', a,b,c,massimo);
```

Osservazioni:

1. Questa è una sequenza di confronti semplici
2. Non ricorda niente questa soluzione?



Vi ricordate?

Algoritmo per ricercare il prodotto migliore

1. Prendi in mano il primo prodotto: assumi che sia il migliore
2. Procedi fino al prossimo prodotto
3. Confrontalo con quello che hai in mano
4. **Se** il prodotto davanti a te è migliore: abbandona il prodotto che hai in mano e prendi quello sullo scaffale
5. **Ripeti** i passi **2 - 4** **fino a** raggiungere la fine della corsia
6. Hai in mano il prodotto migliore.

Algoritmo per trovare il massimo di una sequenza numerica



Esempio

Scrivere un programma che, inserito un intero positivo, determina se corrisponde ad un anno bisestile

- 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);  
  
if bisestile  
    fprintf('%d è bisestile\n', n);  
else  
    fprintf('%d non è bisestile\n', n);  
end
```

Osservazione:

Le variabili **div_4**, **div_100**, **div_400**, **bisestile** sono dei logicals e contengono il risultato di un'operazione logica (true/false)



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



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



- **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



```
a = input('inserire primo operando: ');
b = input('inserire secondo operando: ');
OP = input('inserire operazione (+ - * / ^): ', 's');
switch OP
    case '+'
        res = a + b;
    case '-'
        res = a - b;
    case '*'
        res = a * b;
    case '/'
        if b == 0
            res = Inf;
            fprintf('\ndivisione per zero\n')
        else
            res = a / b;
        end
    otherwise
        fprintf('\nOPERATORE NON RICONOSCIUTO\n')
        res = [];
end
fprintf(' %d %c %d = %d\n', a, OP, b, res);
```




Matlab: Costrutti Iterativi

Istruzioni composte: **while**

Il costrutto **for** verrà presentato dopo gli array



Il Ciclo `while`

```
while expression  
    statement  
end
```

- **expression** assume valore **true** o **false**, può contenere con operatori relazionali (`==`, `<`, `>`, `<=`, `>=`, `~=`)
- **statement** rappresenta il corpo del ciclo, la sequenza di istruzioni da iterare
- **expression** rappresenta la condizione di permanenza nel ciclo: finchè è vera si esegue **statement**
- **expression** deve essere inizializzata (avere un valore) prima dell'inizio del ciclo
- Il valore di espressione deve cambiare nelle ripetizioni



Costrutto Iterativo: **while**, l'esecuzione

1. Terminata **instrBefore** viene valutata **expression**
2. Se **expression** è vera ($0 \neq 0$) viene eseguito **statement**
3. Al termine, viene valutata nuovamente **expression** e la procedura continua finché **expression** è falsa ($= 0$)
4. Uscito dal ciclo, eseguo **instrAfter**

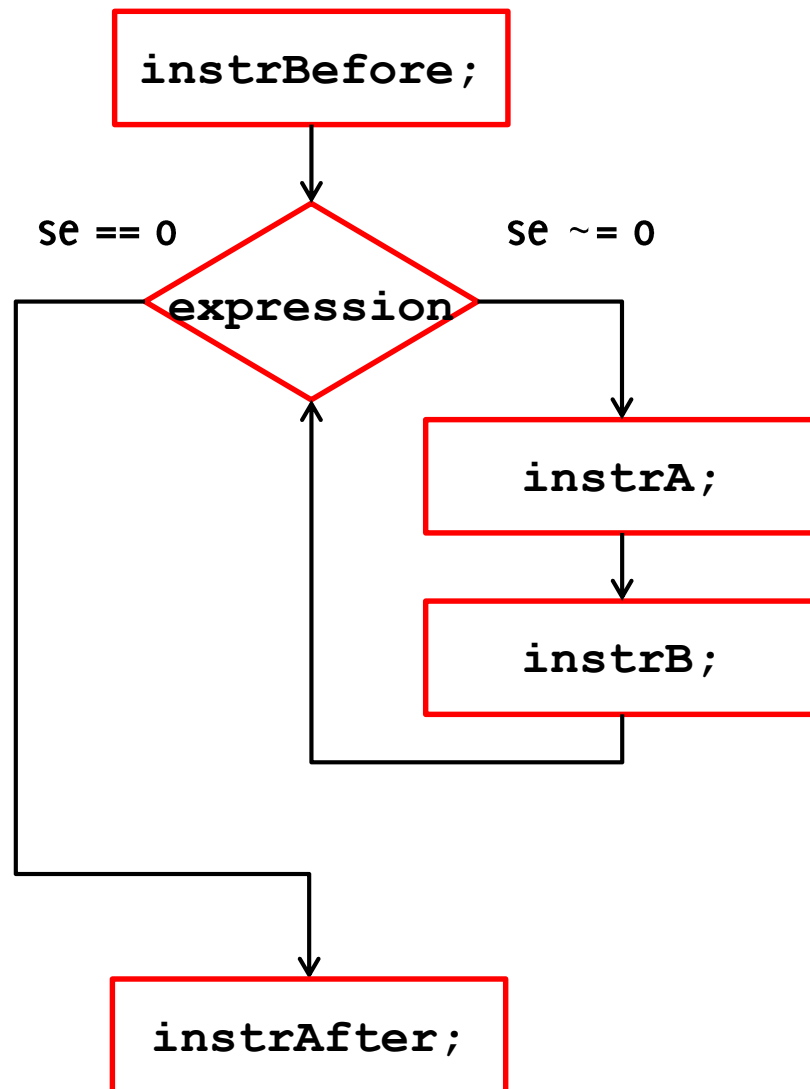
instrBefore;

while (**expression**)
statement;

end

instrAfter;

Costrutto Iterativo: **while**, l'esecuzione



```
instrBefore;  
while (expression)  
    instrA;  
    instrB;  
end  
instrAfter;
```



Esempio

```
% stampa i primi 100 numeri
```



Esempio

```
% stampa i primi 100 numeri  
n = 100;  
while (n > 0)  
    n = n + 1;  
    fprintf('%d, ', n);  
end
```



Esempio

```
% stampa i primi 100 numeri pari  
n = 100;  
while (n > 0)  
    n = n + 1;  
    fprintf('%d, ', 2*n);  
end
```

Manteniamo la variabile **n** come **contatore**, che tiene traccia del numero di iterazioni eseguite nel ciclo



Costrutto Iterativo: **while**, Avvertenze

Il corpo del **while** non viene mai eseguito quando **expression** risulta falsa al primo controllo

```
n = 100;  
while (n < 0)  
    fprintf( '%d, ', 2*n) ;  
end
```




Costrutto Iterativo: **while**, Avvertenze

Se **expression** è vera ed il corpo non ne modifica mai il valore, allora abbiamo un loop infinito (l'esecuzione del programma **non** termina)

```
n = 100;  
while (n > 0)  
    fprintf( '%d, ', 2*n) ;  
end
```



Costrutto Iterativo: `while`

```
% calcolare la somma di una sequenza di numeri  
inseriti dall'utente (continuare fino a quando  
l'utente inserisce 0)
```



Costrutto Iterativo: `while`

```
% calcolare la somma e la media di una sequenza di  
numeri inseriti dall'utente (continuare fino a quando  
l'utente inserisce 0)
```



Esempio

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



Esempio

Calcoliamo gli interessi 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('%g years: $%g\n', year,value)
```

```
end
```



Esempio

% il quadrato di N è uguale alla somma dei primi N numeri dispari,
calcolare il quadrato di un nr inserito da utente (<100)



Costrutti Annidati...

Ovviamente anche il corpo del **while** può contenere altri costrutti (**while** / **if** o altri che vedremo poi)



Esempio di **while** contenente **if**

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

N.B: un anno è bisestile se

- È multiplo di 4 ma non di 100

Oppure

- È multiplo di 400



Soluzione

```
cnt = 0;
bis = 1;
while(bis)
    x = input('inserire anno: ');
    if (mod(x,4)==0) && (mod(x,100) ~= 0) ||
        (mod(x,400) ==0)
        cnt = cnt + 1;
    else
        fprintf('%d non è bisestile', x);
        bis = 0;
    end
end
fprintf('hai inserito %d anni bisestili', cnt);
```



Soluzione

```
cnt = 0;
bis = 1;
while(bis)
    x = input('inserire anno: ');
    if (mod(x,4)==0) && (mod(x,100) ~= 0) ||
        (mod(x,400) ==0)
        cnt = cnt + 1;
    else
        fprintf('%d non è bisestile', x);
        bis = 0;
    end
end
fprintf('hai inserito %d anni bisestili', cnt);
```

Nota che `cnt` tiene traccia di quante volte viene eseguito il ciclo. È una variabile contatore ma non modifica la condizione di permanenza che dipende solo dai valori inseriti dall'utente



Esempio di `while` annidati

Scrivere un programma che stampa la tabella pitagorica

TABELLINE

x	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	12	14	16	18	20
3	0	3	6	9	12	15	18	21	24	27	30
4	0	4	8	12	16	20	24	28	32	36	40
5	0	5	10	15	20	25	30	35	40	45	50
6	0	6	12	18	24	30	36	42	48	54	60
7	0	7	14	21	28	35	42	49	56	63	70
8	0	8	16	24	32	40	48	56	64	72	80
9	0	9	18	27	36	45	54	63	72	81	90
10	0	10	20	30	40	50	60	70	80	90	100



Come stampo la tabellina del 7?

```
tab = 7;

ii = 0;
while(ii <=10)
    fprintf('%3d ', tab * ii);
    ii = ii + 1;
end
fprintf('\n');
```

La variabile `ii` conta quante volte viene eseguito il ciclo e stampa la tabellina del 7



Soluzione

```
tab = 0;
MAX = 10;
while (tab <= MAX)
    ii = 0;
    while (ii <= MAX)
        fprintf('%3d ', tab * ii);
        ii = ii + 1;
    end
    fprintf('\n');
    tab = tab + 1;
end
```



Soluzione

```
tab = 0;
MAX = 10;
while (tab <= MAX)
    ii = 0;
    while (ii <= MAX)
        fprintf('%3d ', tab * ii);
        ii = ii + 1;
    end
    fprintf('\n');
    tab = tab + 1;
end
```

Basta iterare tutte le istruzioni precedenti incrementando `tab` per avere la stampa delle tabelline
Se si modifica `MAX` si ottengono più tabelline e più lunghe



Esercizi (TODO)

- Scrivere un programma che richiede all'utente una sequenza di caratteri minuscoli e ne stampa il corrispettivo maiuscolo (fino a quando l'utente non inserisce #)
- Scrivere un programma che richieda all'utente di inserire due interi e ne calcola il massimo comune divisore.
 - **Modificarlo per fare meno prove del minimo tra gli input, utilizzando variabili di flag.**
- **Modificare esercizio tabelline per stampare solo la parte triangolare alta.**



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



Esercizi TODO:

Preparare un programma C per giocare a Carta / Sasso / Forbice, richiedendo all'utente di inserire i caratteri 'c', 's', 'f', controllando anche che il carattere inserito sia ammissibile.



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)
        continue;
    end
    fprintf('%d', x);
    ii = ii + 1
end
```



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
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
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 0



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
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



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 **compare 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



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



Stampa dei Risultati (2)

- **disp** vs. **fprintf**
 - **disp** è in grado di stampare anche valori complessi
 - $x = 2 \cdot (1 - 2i)^3$;
 - `str = ['disp: x = ' num2str(x)];`
 - `disp(str);`

disp: x = -22+4i

- **fprintf** ne stampa solo la parte reale
 - `fprintf('fprintf: x = %8.4f\n', x);`

fprintf: x = -22.0000