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 determina se un numero inserito dall'utente è un intero positivo

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

Esempio

```
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
```

Esempio

```
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

~(A && B) = ~A || ~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 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 richiede all'utente di scegliere un numero da 1 a 6, simula il lancio di un dato 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)

Esempio

```
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, 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 (ture/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

Algebra di Boole

Operazioni Logiche

Non solo operazioni aritmetiche

- Nei linguaggi di programmazione occorre spesso valutare delle condizioni logiche

Esempio: algoritmo per andare in Università

- Mi alzo quando suona la sveglia
- Mi dirigo verso la cucina
- Mangio e bevo un caffè
- Mi lavo e mi vesto
- **Se** devo mangiare in fuori
  - prendo il pranzo

  Altrimenti
  - prendo uno snack per metà mattina

Non solo operazioni aritmetiche

- **Espressione booleana**: espressione con valore **vero (1) o falso (0)**, determinata durante l'esecuzione del programma
- Nei linguaggi di programmazione le espressioni booleane si ottengono spesso mediante **operatori relazionali**

  (ad esempio, in Matlab **`==`**, **`~=`**, **`>`**, **`<`**, **`>=`**, **`<=`**)

  Es: **`(a > 7)`** , **`(b % 2 == 0)`**, **`(x <= w)`**

- Si possono considerare espressioni booleane anche frasi del linguaggio corrente, quali

  «Michele è biondo», «Giovanni studia molto»

  Che possono essere vere o false (0/1)

Espressioni booleane composte

Le **espressioni booleane composte** espressioni con valore 0/1 ottenute concatenando **espressioni booleane** con **operatori logici.**

Gli **operatori logici** sono:

NOT : ~

AND : **&&**

OR : **||**

Es: **(a > 7) && (b == 0)**

« **(**Michele è biondo) e (ha 4 anni)»

« **(**Giovanni mangia la pasta) o (mangia il riso)»

Operatori Logici

- **Operatori** che si applicano a **espressioni booleane** per costruire **condizioni composte**.
  - `~` : negazione (NOT),
  - `&&` : congiunzione (AND)
  - `||` : disgiunzione (OR),

- Nel seguito indichiamo le espr. booleane con lettere
  Es: `(a > 9) && (w %5 == 0)`

- In generale le **espressioni booleane** sono espressioni che possono essere **vere o false**

Es A = «Giovanni è più grande di Michele»
B = «Michele è biondo»

Tabelle di Verità

- Rappresenta tutti i possibili modi di valutare un' espressione booleana composta
- Una riga per ogni possibile assegnamento di valori logici alle variabili:
  - n variabili logiche (espressioni booleane) $\rightarrow 2^n$ possibili assegnamenti, quindi 2^n righe.
- Una colonna per ogni espressione che compone l'espressione data (inclusa la formula stessa)

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti

- Il NOT è un operatore **unario**, che prende in ingresso **una** sola espressione.

- `~A` è l'opposto di `A`

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti

- Il NOT è un operatore **unario**, che prende in ingresso **una** sola espressione.

- **~A** è l'opposto di **A**

negazione (NOT)	
A	***~A***
0	1
1	0

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti

- L'operatore AND è **binario**, prende in ingresso **due** espressioni.

- **`A && B`** è vero se e solo se sia **`A`** che **`B`** sono vere.

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti

- L'operatore AND è **binario**, prende in ingresso **due** espressioni.

- **`A && B`** è vero se e solo se sia **`A`** che **`B`** sono vere.

congiunzione (AND)		
A	**B**	**A && B**
0	0	0
0	1	0
1	0	0
1	1	1

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore AND è **binario**, prende in ingresso **due** espressioni.
- **`A && B`** è vero se e solo se sia **`A`** che **`B`** sono vere.

congiunzione (AND)		
A	**B**	**A && B**
0	0	0
0	1	0
1	0	0
1	1	1

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore OR è **binario**, prende in ingresso **due** espressioni.
- **`A || B`** è vero se almeno una delle due è vera.

disgiunzione (OR)		
A	**B**	**A || B**
0	0	0
0	1	1
1	0	1
1	1	1

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore OR è **binario**, prende in ingresso **due** espressioni.
- **`A || B`** è vero se almeno una delle due è vera.
- **NB:** non è un OR esclusivo, come spesso accade nella lingua parlata

disgiunzione (OR)		
`A`	**`B`**	**`A || B`**
0	0	0
0	1	1
1	0	1
1	1	1

Aritmetica Operatori Logici

- Ordine Operatori Logici in assenza di parentesi (elementi a priorità maggiore in alto):

1. negazione (NOT) ~
2. operatori di relazione **<**, **>**, **<=**, **>=**
3. uguaglianza **==**, disuguaglianza **~=**,
4. congiunzione (AND) **&&**
5. disgiunzione (OR) **||**

Esempio

- `x > 0 || y == 3 && ~(z > 2)`
- `(x > 0) || ((y == 3) && ~(z > 2))`

Aritmetica degli Operatori Logici

- Gli operatori `&&` e `||` sono commutativi
  - `(a && b) == (b && a)`
  - `(a || b) == (b || a)`
- Le doppie negazioni si elidono: `~~a == a`

Tabelle di Verità

- Rappresenta tutti i possibili modi di valutare un' espressione booleana composta
- Una riga per ogni possibile assegnamento di valori logici alle variabili:
  - n variabili logiche (espressioni booleane) $\rightarrow 2^n$ possibili assegnamenti, quindi 2^n righe.
- Una colonna per ogni espressione che compone l'espressione data (inclusa la formula stessa)

Esempio Tabella di Verità

- `A && ~B || C`

Esempio Tabella di Verità

- `A && ~B || C`

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Esempio Tabella di Verità

- `A && ~B || C`

A	B	C	~B	A && ~B	A && ~B || C
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Esempio Tabella di Verità

- `A && ~B || C`

A	B	C	~B	A && ~B	A && ~B || C
0	0	0	1		
0	0	1	1		
0	1	0	0		
0	1	1	0		
1	0	0	1		
1	0	1	1		
1	1	0	0		
1	1	1	0		

Esempio Tabella di Verità

- `A && ~B || C`

A	B	C	~B	A && ~B	A && ~B || C
0	0	0	1	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	1	0	0	
1	0	0	1	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	1	0	0	

Esempio Tabella di Verità

- `A && ~B || C`

A	B	C	~B	A && ~B	A && ~B || C
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	1

Altro Esempio di Tabella di Verità

- `A && (~B || C)`

Altro Esempio di Tabella di Verità

- `A && (~B || C)`

A	B	C	~B	~B || C	A && (~B||C)
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Operatori Logici: Leggi di de Morgan

- Leggi di De Morgan: illustrano come distribuire la negazione rispetto a `||` e `&&`
  1. `~(a && b) == ~a || ~b`
  2. `~(a || b) == ~a && ~b`

- Es: `~((a >= 5) && (a <= 10))` -> [De Morgan]

  `~(a >= 5)||~(a <= 10)` -> [proprietà `>=` e `<=`]

  `~~(a < 5)|| ~~(a > 10)` -> [doppia negazione] `((a < 5) || (a > 10))`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- Due possibili soluzioni:
  - Applicando le leggi di De Morgan cerco di passare da una all'altra
  - Calcolo entrambe le tabella di verità e mostro che coincidono

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~`
- `~(I`
- `(~I`
- `A`
- `A`
- `A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~(B || ~C) || ~~A)`
- `~(I`
- `(~I`
- `A`
- `A`
- `A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~(B || ~C) || ~~A)`
- `~(B || ~C) || A`
- `(~I`
- `A`
- `A`
- `A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~(B || ~C) || ~~A)`
- `~(B || ~C) || A`
- `(~B && C) || A`
- `A`
- `A`
- `A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~(B || ~C) || ~~A)`
- `~(B || ~C) || A`
- `(~B && C) || A`
- `A || (~B && C)`
- `A`
- `A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~(B || ~C) || ~~A)`
- `~(B || ~C) || A`
- `(~B && C) || A`
- `A || (~B && C)`
- `A || (C && ~B)`
- `A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && ~B`
  - `~((B || ~C) && ~A)`

- `~((B || ~C) && ~A)`
- `(~(B || ~C) || ~~A)`
- `~(B || ~C) || A`
- `(~B && C) || A`
- `A || (~B && C)`
- `A || (C && ~B)`
- `A || C && ~B`

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 (dimensioni compatibili)
- Valori numerici di a, b vengono interpretati come logici:
  - 0 come falso
  - tutti i numeri diversi da 0 come vero

Differenza tra `&&` e `&` (e tra || e |)

`&&` è l'operatore short-circuit per logicals scalari

- Quando calcola `a && b` non valuta `b` se `a` è falso

|| è l'operatore short-circuit per logicals scalari

- Quando calcola `a || b` non valuta `b` se `a` è vero

`&` e || si applicano anche a vettori di logicals (vedremo poi)

`&` può essere conveniente. Si pensi di dover valutare se la seguente espressione è vera:

«ho meno di 15 anni» `&` «I giochi olimpici invernali del 1924 si sono tenuti in Francia»

Usare `&&` e `&` permette di evitare di andare a cercare su Wikipedia …

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

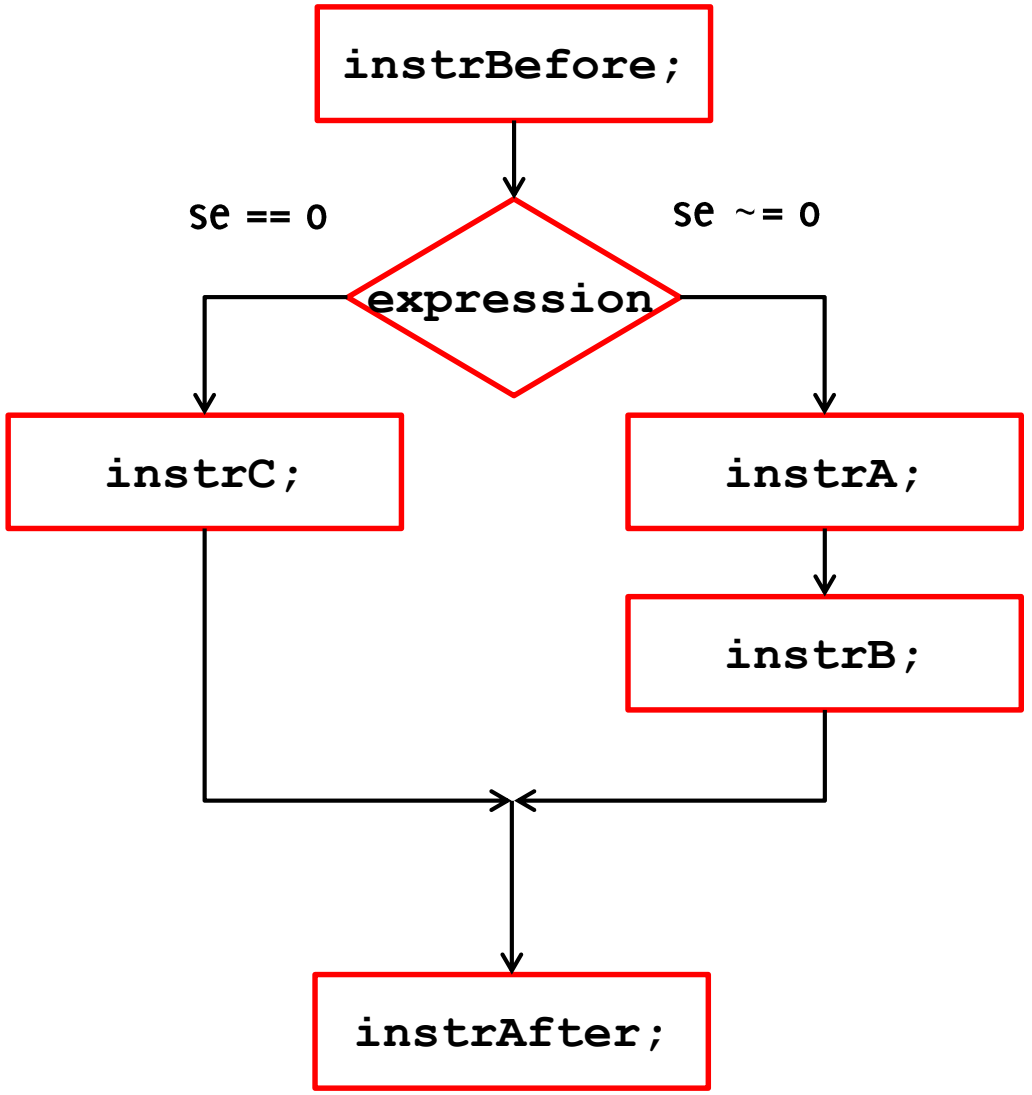
1. Terminata **`instrBefore`**, valuto **`expression`**,
2. Se **`expression`** è vera (**`!=0`**), allora eseguo **`statement1`**, altrimenti eseguo **`statement0`**. (se è presente **`else`**)
3. Terminato lo statement dell'**`if`** , procedi con **`instrAfter`**, la prima istruzione fuori dall'**`if`**

- **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;
```

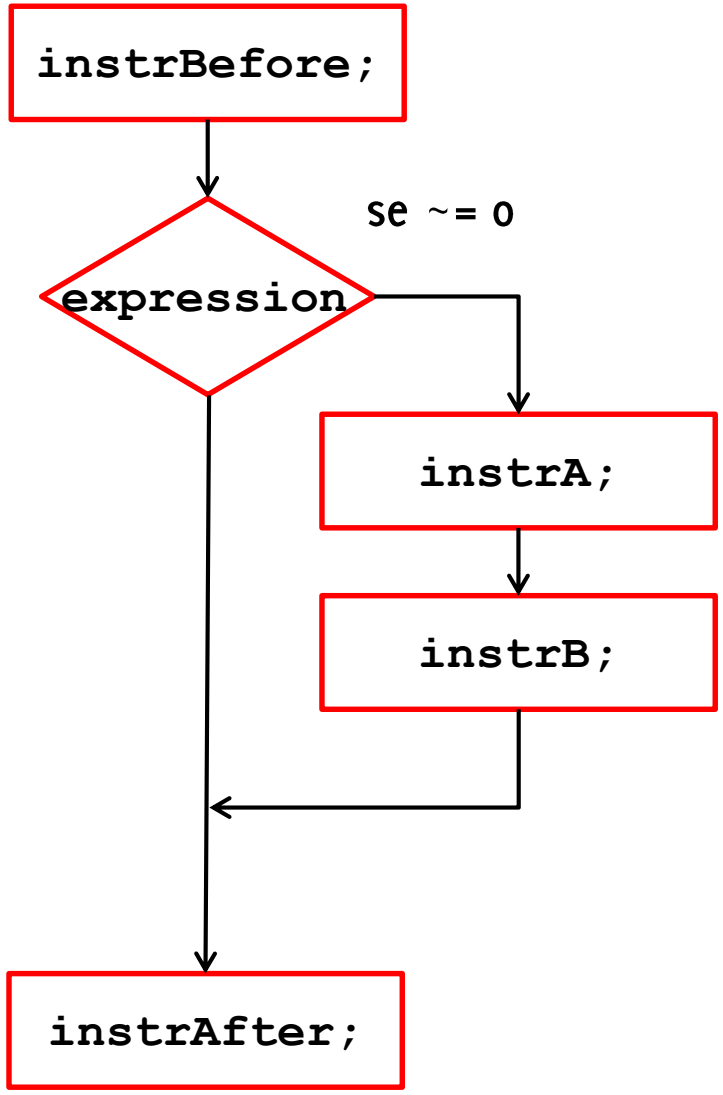
Costrutto Condizionale: `if`, l'esecuzione

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



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

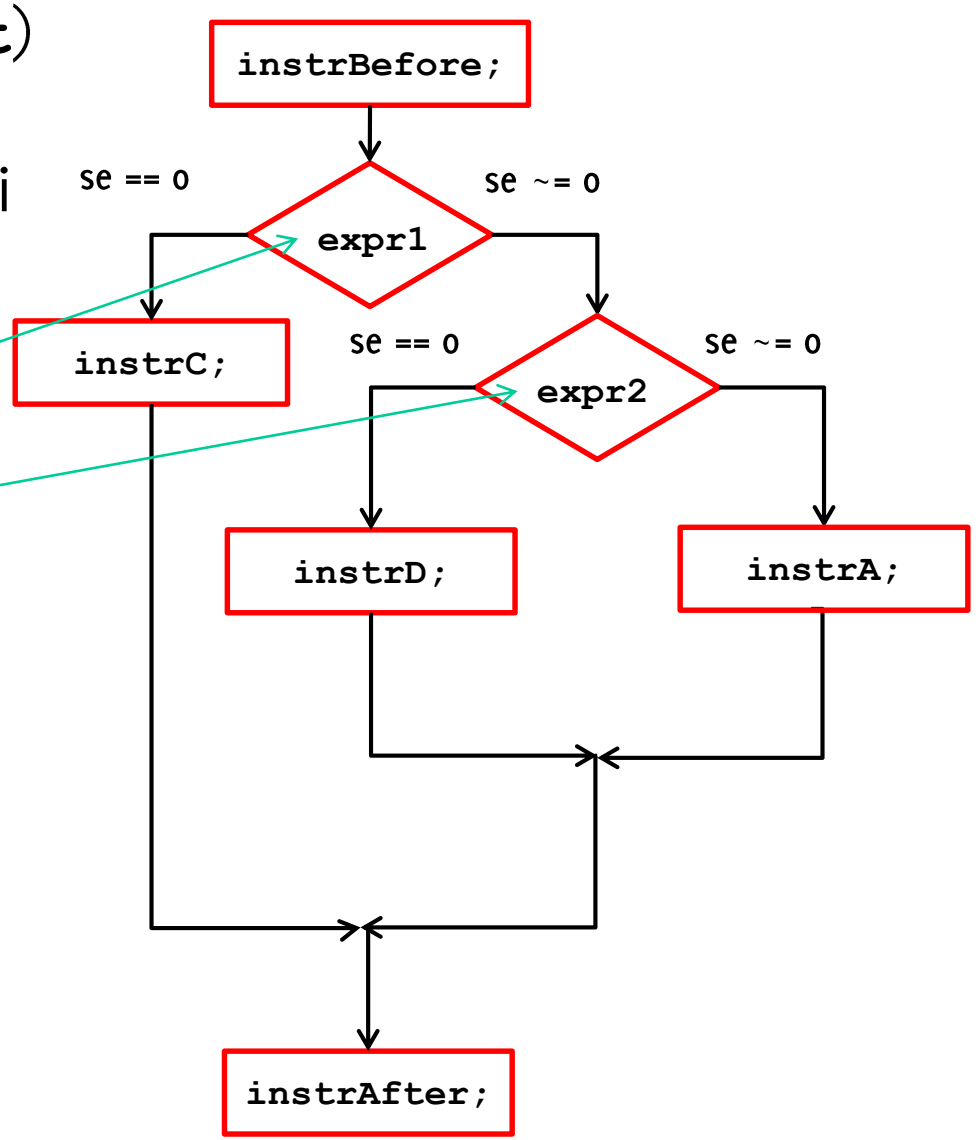
Il corpo di un **`if`** (cioè uno **`statement`**)
può a sua volta contenere costrutti **`if:`**
si realizzano quindi istruzioni condizionali
annidate

```
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
   ..........
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!

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

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

Soluzione

```
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
```

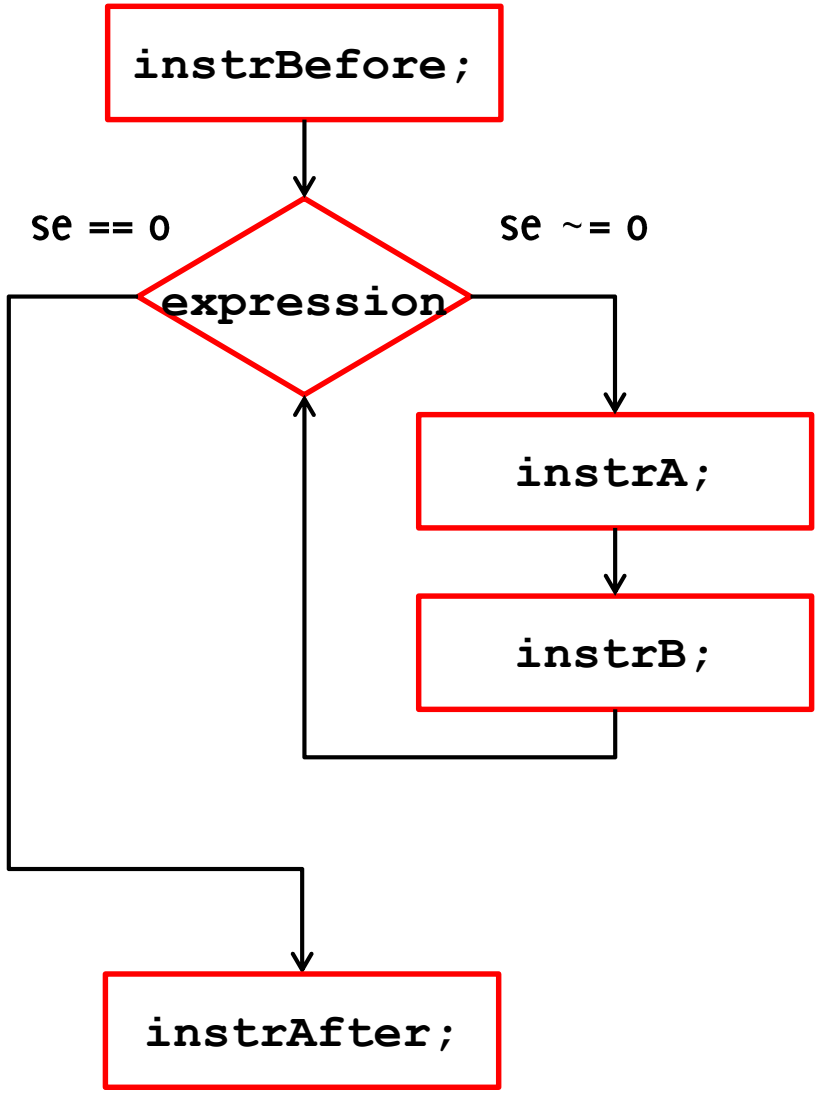
- **`expression`** assume valore **true** o **false**, può contenere con operatori relazionali (==, <, >, <=, >=, ~=)
- **`statement`** rappresenta il corpo del ciclo, la sequenza di istruzioni da iterare
- **`expression`** rappresenta la condizione di permanenza nel ciclo: finchè è vera si esegue **`statement`**
- **`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 ~= 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)

Esempio

```
% il quadrato di N è uguale alla somma dei primi N numeri dispari,
calcolare il quadrato di un nr inserito da utente (<100)
max = 100;
n = input('inserire un numero minore di 100 ');
if n < 100
    s  = 0;
    ii = 0;
    while(ii<n)
        s = s + 2 * ii + 1;
        ii = ii + 1;
    end
    fprintf('il quadrato di %d è %d', n, s);
else
    fprintf('errore, inserire numeri <= 100');
end
```

Non ci sarebbero problemi a prendere un valore maggiore di MAX

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