



# Introduzione al Linguaggio Matlab

Informatica AA 2020 / 2021

Giacomo Boracchi

18 Settembre 2020

[giacomo.boracchi@polimi.it](mailto:giacomo.boracchi@polimi.it)



Il Matlab

MATrix LABoratory



- Cos'è Matlab (MATrix LABoratory):
  - Un linguaggio di programmazione per calcolo numerico
  - Un Ambiente di sviluppo
- È pensato (e ottimizzato) per operare su matrici (ma include generiche funzionalità matematiche)

Lo utilizzerete nei successivi corsi di calcolo numerico

MATLAB è uno strumento commerciale, su licenza NON gratuita,

- Student edition fornita dal Politecnico (maggiori dettagli a laboratorio)

# Screenshot interfaccia MATLAB

The screenshot shows the MATLAB R2015b interface with several components and annotations:

- Toolbar:** Located at the top, containing icons for file operations (New, Open, Save, Find Files, Compare, Go To, Find, Print), editing (Insert, Comment, Indent), and execution (Run, Run and Advance, Run Section, Advance, Run and Time). An arrow points to it with the text "Lancia i tool di MATLAB ed altro...".
- Current Folder:** A file explorer on the left showing a directory with files like `calcolaSommaDivisori.m`, `cerca.m`, `inserisciRilievo.m`, `rilevamenti.m`, `scacchiera.m`, `sommaCompresi.m`, and `strLenRic.m`. An arrow points to it with the text "Contenuto della directory corrente".
- Editor:** The central window showing MATLAB code. An arrow points to the code with the text "Codice nell'Editor". The code includes:

```
1 n = input('inserire nr rilievi');
2
3 for ii = [1 : n]
4     rilievo(ii) = inserisciRilievo();
5 end
6
7 altezza media rilievi con
8 t tra 20 e 60 e long tra 30 e 40
9
10 altezze = [];
11 for ii = [1 : n]
12     if (rilievo(ii).lat > 20 && rilievo(ii).
13         && rilievo(ii).lon > 30 && rilie
14         altezze = [altezze, rilievo(ii).alt]
15     end
16 end
17
18 altezzaMedia = mean(altezze)
19
```
- Command Window:** On the right, showing the execution of `a = 1321` and the resulting value `a = 1321`. An arrow points to it with the text "Finestra dei comandi".
- Workspace:** At the bottom left, showing a table of variables. An arrow points to it with the text "Mostra le variabili nel workspace".

Name	Value
a	1321



## Installazione di Matlab

Potete installare Matlab dal sito Polimi seguendo le istruzioni

[https://boracchi.faculty.polimi.it/teaching/InfoB/matlab\\_install.pdf](https://boracchi.faculty.polimi.it/teaching/InfoB/matlab_install.pdf)

Installate Matlab il prima possibile, e certamente **prima del primo laboratorio**

Se avrete problemi di installazione l'ing Marelli vi assisterà

(Credits Diego Carrera)



## Caratteristiche del linguaggio di Matlab

Linguaggio di alto livello

- simile a linguaggi di programmazione C, Java, Pascal
- possiede comandi sintetici per effettuare complesse elaborazioni numeriche

Linguaggio interpretato, i comandi e istruzioni

- NON sono tradotti in codice eseguibile dall'hardware
- Ma invia istruzioni ad un altro programma, **l'interprete**, che li analizza ed esegue azioni da essi descritte



## Matlab è un Linguaggio Interpretato

### Linguaggi Interpretati:

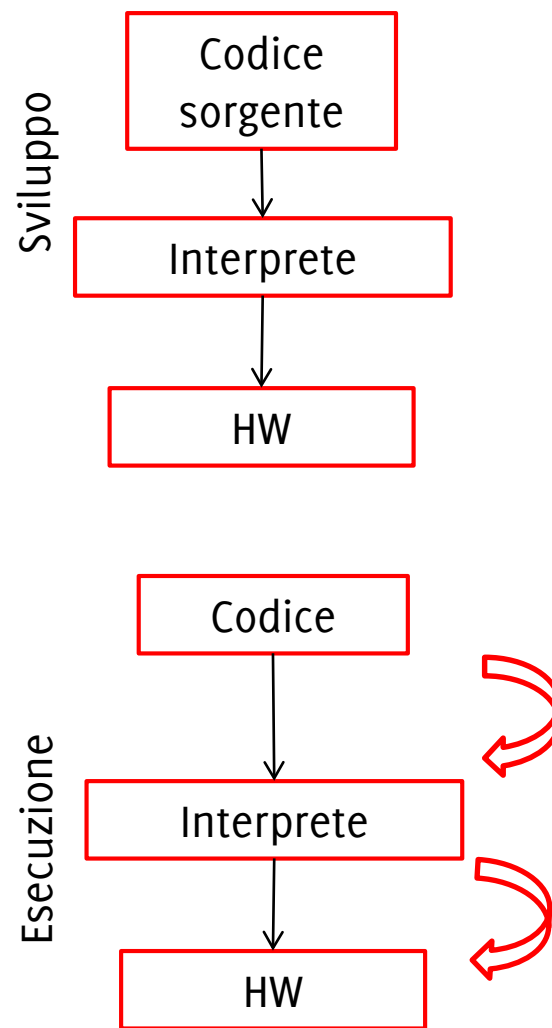
L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente

L'esecuzione del programma richiede la presenza del codice (talvolta il sorgente) e dell'interprete.

I programmi sono meno efficienti di quelli compilati

Portabilità meno pratica

Sviluppo più facile: è possibile eseguire le istruzioni mentre si scrive il codice sorgente





## Linguaggi Compilati vs Interpretati

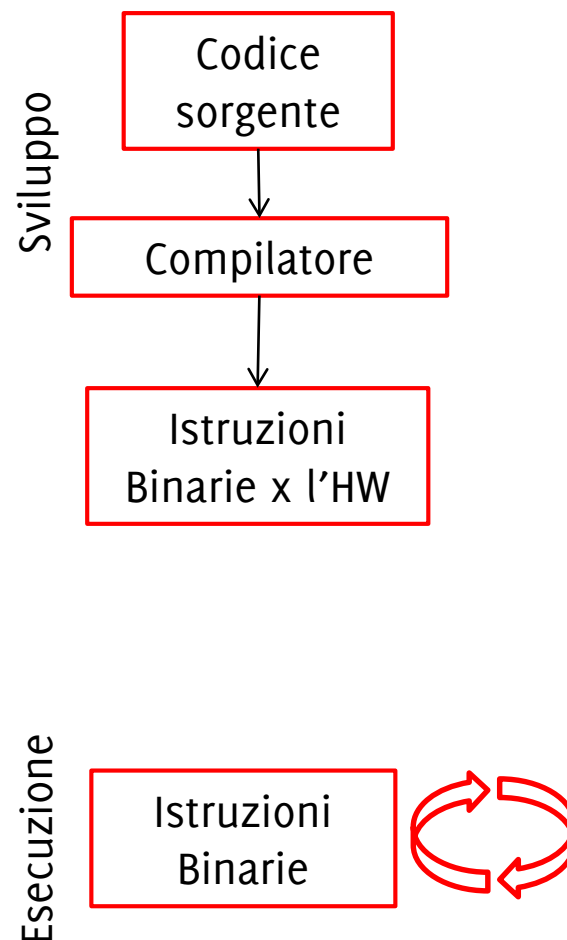
### Linguaggi **Compilati**:

Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)

L'esecuzione del programma non richiede la presenza del codice sorgente, né del compilatore.

I programmi sono efficienti

Il programma è facilmente portabile su piattaforme analoghe







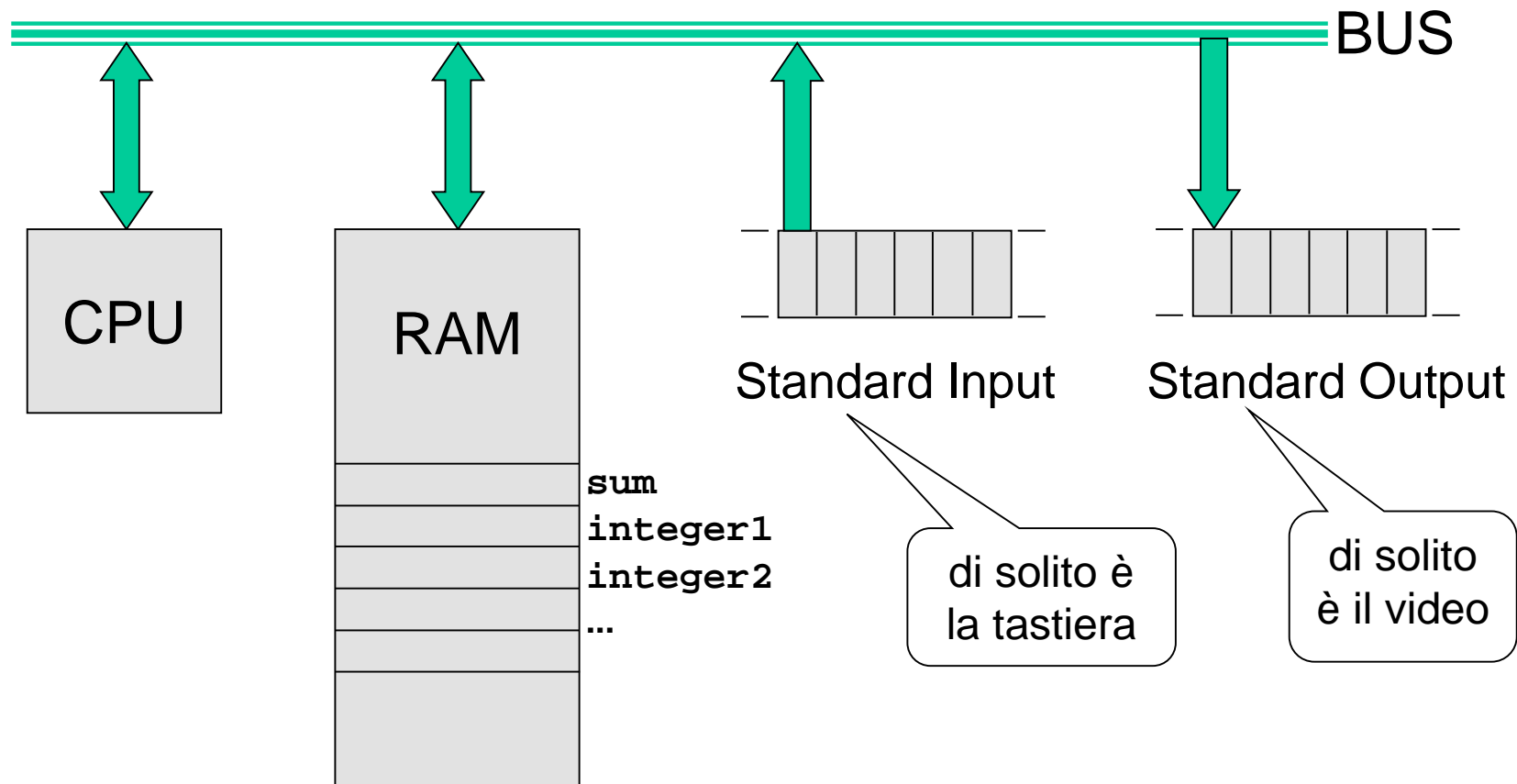
## Il nostro esecutore

Una panoramica sull'architettura dal calcolatore



# La macchina astratta C... ma andrà bene anche a noi

Algoritmi e programmi sono definiti in funzione del loro esecutore  
L'esecutore dei programmi C è una macchina astratta





# Le Istruzioni



## Le Istruzioni e la Command Window

Le **istruzioni** possono essere **inviate** direttamente **all'interprete** se scritte nella command window (dopo il simbolo `>>` )

- La command window è come una «super calcolatrice»
- La command window ha un'interfaccia testuale che inizia con `>>`

# Screenshot interfaccia MATLAB

The screenshot displays the MATLAB R2015b interface. The main window is the Editor, showing a script named 'rilevamenti.m'. The script contains the following code:

```
1 n = input('inserire nr rilievi');
2
3 for ii = [1 : n]
4     rilievo(ii) = inserisciRilievo();
5 end
6
7 % atezza media rilievi con
8 % lat tra 20 e 60 e long tra 30 e 40
9
10 altezze = [];
11 for ii = [1 : n]
12     if (rilievo(ii).lat > 20 && rilievo(ii).
13         && rilievo(ii).lon > 30 && rilie
14         altezze = [altezze, rilievo(ii).alt]
15     end
16 end
17
18 altezzaMedia = mean(altezze)
19
20 %%
21 LAT = [rilievo.lat];
22 LON = [rilievo.lon];
23 ALT = [rilievo.alt];
24
25 indx = find(LAT > 20 & LAT < 60 & LON > 30 &
```

The Command Window on the right shows the following output:

```
-----
Your MATLAB license will expire in 24 days
Please contact your system administrator o
MathWorks to renew this license.
-----

Academic License

>> a = 1321
a =
    1321
fx >>
```

An arrow points from a red-bordered box labeled "Finestra dei comandi" to the Command Window.

The Workspace window at the bottom left shows a table with the following data:

Name	Value
a	1321

The File Browser on the left shows the following files:

- calcolaSommaDivisori.m
- cerca.m
- inserisciRilievo.m
- rilevamenti.m
- scacchiera.m
- sommaCompresi.m
- strLenRic.m

The Windows taskbar at the bottom shows the system tray with the date and time: 00:24 lunedì 06/11/2017.



## Esempio: le operazioni aritmetiche

Nella command window è possibile eseguire qualsiasi operazione aritmetica

```
>> 5 + 7
```

```
ans =
```

```
12
```

```
>> 5 / 7
```

```
ans =
```

```
0.7143
```

**ans** è una variabile «di default»  
che contiene il risultato di  
un'istruzione che sia un  
assegnamento



## Esempio: le operazioni aritmetiche

Nella command window è possibile eseguire qualsiasi operazione aritmetica

```
>> 5 + 7      >> 5 * 7      >> 5 ^ 7
ans =
    12          ans =
    35          ans =
    78125
```

Elevamento a potenza

```
>> 5 / 7      >> 'a' + 2
ans =
    0.7143     ans =
    99
```

I caratteri alfanumerici si indicano con l'apice singolo: sono sempre legati agli interi mediante la tabella ASCII



## Istruzioni e Codice Sorgente

Le istruzioni possono essere contenute in un **file sorgente**, in particolare:

- uno script
- una funzione

e quindi eseguite in maniera sequenziale.

L'esecuzione di uno codice sorgente può essere visto come l'inserimento delle varie istruzioni in sequenza nella command window.



# Screenshot interfaccia MATLAB

The screenshot displays the MATLAB R2015b interface. The main window is the Editor, showing a script named 'rilevamenti.m'. The script contains the following code:

```
1 n = input('inserire nr rilievi');
2
3 for ii = [1 : n]
4     rilievo(ii) = inserisciRilievo();
5 end
6
7 % atezza media rilievi con
8 % lat tra 20 e 60 e long tra 30 e 40
9
10 altezze = []
11 for ii = 1:n
12     if (rilievo(ii).lat > 20 & rilievo(ii).lat < 60 &
13         rilievo(ii).lon > 30 & rilievo(ii).lon < 40)
14         altezze = [altezze, rilievo(ii).alt]
15     end
16 end
17
18 altezzaMedia = mean(altezze)
19
20 %%
21 LAT = [rilievo.lat];
22 LON = [rilievo.lon];
23 ALT = [rilievo.alt];
24
25 indx = find(LAT > 20 & LAT < 60 & LON > 30 &
```

The Command Window on the right shows the following output:

```
-----
Your MATLAB license will expire in 24 days
Please contact your system administrator or
MathWorks to renew this license.
-----

Academic License

>> a = 1321
a =
    1321

fx >>
```

The Workspace on the left shows a variable 'a' with a value of 1321.

A red box highlights the text "Editor, permette di scrivere funzioni e script" with an arrow pointing to the code in the Editor window.



# Le Variabili



### Variabile $\Leftrightarrow$ cella di memoria

Le variabili hanno un **nome**: un **identificatore simbolico** formato da successione di **lettere, cifre e carattere \_** con al primo posto una lettera.

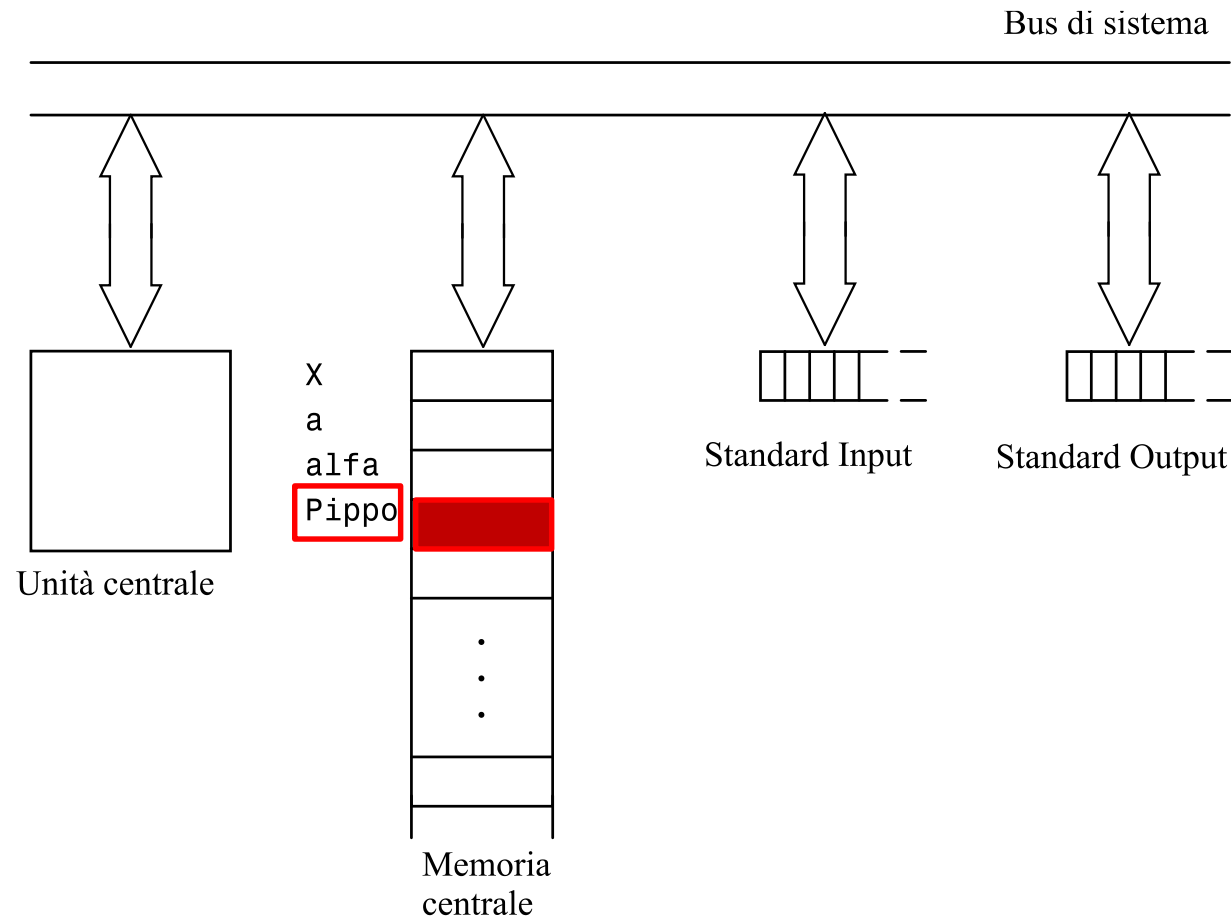
- Es. di identificatori: **a, x, alfa, pippo, a1, Giuseppe, DopoDomani, velocita\_massima**
- **NB**: maiuscole distinte dalle minuscole (**Alfa, alfa** e **ALPHA** sono tre diversi identificatori).

Si parla di identificatore simbolico perché permettono di accedere ad una cella di memoria (dov'è contenuto il valore della variabile) senza sapere dove sia questa cella (indirizzo, dimensioni in memoria).



## Le Variabili, identificatori simbolici

Per accedere (in lettura o scrittura) alla cella in rosso mi basta far riferimento alla variabile **Pippo** nel codice.





## le Variabili (cnt)

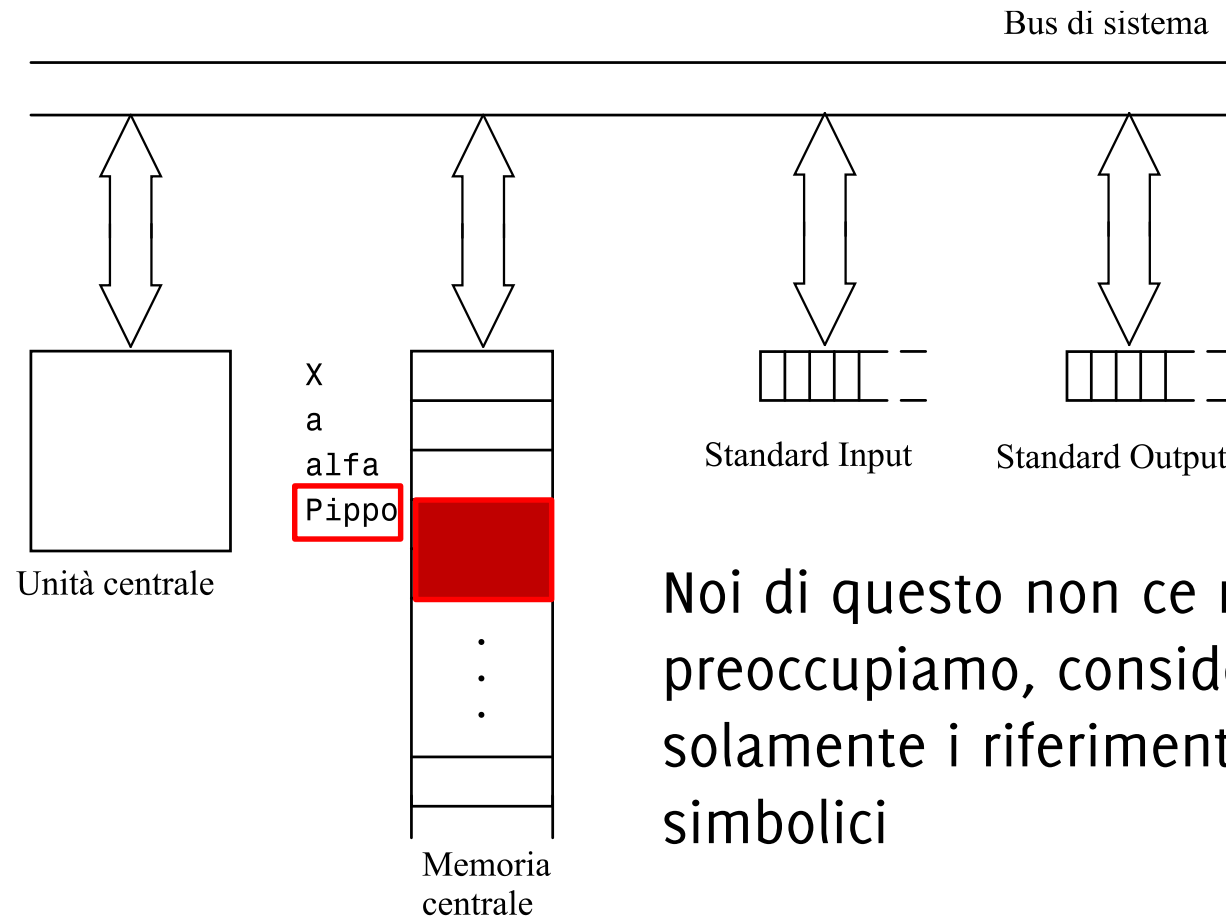
**Tutte le variabili** hanno un **tipo**, che:

- caratterizza i valori scrivibili nella cella
- le operazioni sulla variabile
- la dimensione della cella in memoria (di questo non ce ne preoccupiamo)



## Le Variabili, identificatori simbolici

A seconda del tipo è possibile che più celle facciano riferimento alla stessa variabile (**Pippo**)



Noi di questo non ce ne preoccupiamo, consideriamo solamente i riferimenti simbolici



## Il Valore delle Variabili

Per visualizzare il valore delle variabili:

- Basta inserire il nome della variabile nella command window

```
>> a
```

```
    a =
```

```
     7
```

- Il nome della variabile non deve essere seguito da ; altrimenti la visualizzazione viene soppressa

```
>> a;
```

```
>>
```



## Il Workspace

Tutte le variabili vengono salvate nel workspace, che corrisponde allo spazio di memoria del programma

E' possibile visualizzare le variabili ed il workspace:

- Il comando **whos** (visualizza tutte le variabili)
- Il comando **whos nomeVariabile** (visualizza solo **nomeVariabile**)
- Il pannello del Workspace

Per pulire il workspace e rimuovere tutte le variabili presenti si usa il comando: **clear**

```
>> clear
```

```
>> whos
```

```
>>
```

Nessuna variabile è più presente nel workspace





# Istruzioni di Assegnamento

E dichiarazione delle variabili



### Sintassi

**nomeVariabile = espressione**

Vengono eseguite le seguenti operazioni:

1. Viene valutato il valore di **espressione**
2. Il valore di **espressione** (a destra dell'uguale) viene copiato nella variabile **nomeVariabile** (a sinistra dell'uguale)



### Sintassi

**nomeVariabile = espressione**

Vengono eseguite le seguenti operazioni:

1. Viene valutato il valore di **espressione**
2. Il valore di **espressione** (a destra dell'uguale) viene copiato nella variabile **nomeVariabile** (a sinistra dell'uguale)

L'assegnamento è diverso dall'uguaglianza perché ha un verso ben definito: prende i valori da destra dell'uguale e li copia nella variabile a sinistra dell'uguale

**nomeVariabile = espressione**





## Dichiarazione di Variabili Mediante Assegnamento

Quando assegno un valore ad una variabile che non è stata inizializzata (e.g., **a**), la **variabile viene creata**

```
>> a = 7
```

```
a =
```

```
7
```

Posso quindi modificare il valore di **a** mediante successivo assegnamento

```
>> a = 15
```

```
a =
```

```
15
```

Il nome “variabile” deriva dal fatto che queste possono cambiare valore



In

**nomeVariabile = espressione**

**espressione** potrebbe essere:

- un'operazione algebrica, ad esempio

```
>> a = 15 / 3 + 2
```

```
a =
```

```
7
```

- un'espressione che coinvolge una variabile

```
>> b = a - 1
```

```
b =
```

```
6
```



## Note sull'Assegnamento (cnt)

Attenzione:

```
>> a = a + 1
```

Cosa fa? Due opzioni (una è molto sbagliata)

- È un'equazione, senza soluzione?  
Non esiste valore di **a** che coincide ad **a+1**
- È un'operazione di assegnamento quindi
  1. Legge il valore di **a** a dx dell'uguale
  2. Somma a questo valore 1, il risultato è il valore di **espressione**
  3. Assegna il valore di **espressione** ad **a**



## Note sull'Assegnamento (cnt)

Attenzione:

```
>> a = a + 1
```

Cosa fa? Due opzioni (una è molto sbagliata)

- È un'equazione, senza soluzione?  
Non esiste valore di ~~a~~ che coincide ad **a+1**
- È un'operazione di assegnamento quindi
  1. Legge il valore di **a** a dx dell'uguale
  2. Somma a questo valore 1, il risultato è il valore di **espressione**
  3. Assegna il valore di **espressione** ad **a**



## Note sull'Assegnamento (cnt)

Quindi,

```
>> a = 7;
```

```
>> a = a + 1
```

```
    a =
```

```
     8
```





## Note sull'Assegnamento (cnt)

Ovviamente non è possibile assegnare ad una variabile, il valore di una variabile che non esiste:

```
>> a = v
```

**Undefined function or variable 'v'.** (messaggio di errore dell'interprete Matlab)

Non è richiesto il ; al termine dell'istruzione

Il risultato di un'operazione che non comporta un assegnamento viene assegnato alla variabile **ans**



## Le Istruzioni di Assegnamento: i caratteri

I caratteri alfanumerici vanno racchiusi tra apici singoli: “

Assegnamenti di un carattere ad una variabile

```
a = 'A';
```

```
a = 'z';
```

```
a = '1';
```

**N.B:** l'ultima istruzione assegna alla variabile a il valore corrispondente al carattere 1 che nella tabella ASCII corrisponde al numero 49



I Tipi



## Tipo Double

Di default, valori numerici danno luogo a variabili di tipo **double**: un double contiene uno **scalare** espresso con doppia precisione (**64 bit**)

È possibile vedere il tipo delle variabili mediante **whos**

```
whos nomeVariabile
```

```
>> a = 7;
```

```
>> whos a
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>	<b>Attributes</b>
a	1x1	8	double	



## Tipo Char

Una variabile di tipo char contiene uno **scalare** a 16 bit (8 bit in Octave), ciascuno dei quali rappresenta un carattere

- Es: **Iniziale = 'G';**

Nome della variabile

Carattere 'G' corrisponde  
nella tabella ASCII al numer

```
whos Iniziale
```

Name	Size	Bytes	Class	Attributes
Iniziale	1x1	2	char	



## Tipo Double (2)

In Matlab è possibile rappresentare anche numeri **complessi**

parti reali e immaginarie possono essere positive o negative nell'intervallo di valori  $[10^{-308}, 10^{308}]$

```
>> a = [sqrt(-1)]
```

```
a =  
    0 + 1.0000i
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	16	double

Attributes complex
-----------------------



## Gestione Dinamica delle Variabili

I tipi delle variabili possono cambiare:

- mediante conversione esplicita
- mediante assegnamento: **il tipo** di una variabile è **definito dal valore** contenuto

```
>> a = 1
```

```
a =
```

```
    1.0000
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x1	8	<b>double</b>	

```
>> a = 'c';
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x1	2	<b>char</b>	



## Tipo Char

Ogni carattere corrisponde ad un numero intero.

Il mapping è dato dalla tabella ASCII (es 'A' è 65, 'a' è 97)

È quindi possibile sommare numeri ai caratteri, però attenzione che questo modifica il tipo della variabile

```
>> iniziale_nome = 'G'
```

```
iniziale_nome =
```

```
G
```

```
>> whos iniziale_nome
```

Name	Size	Bytes	Class	Attributes
iniziale_nome	1x1	2	char	





## Tipo Char

Ogni carattere corrisponde ad un numero intero.

Il mapping è dato dalla tabella ASCII (es 'A' è 65, 'a' è 97)

È quindi possibile sommare numeri ai caratteri, però attenzione che questo modifica il tipo della variabile

```
>> iniziale_nome = iniziale_nome + 1
iniziale_nome =
```

72

```
>> whos iniziale_nome
```

Name	Size	Bytes	Class	Attributes
iniziale_nome	1x1	8	double	





## Rappresentazione dei Caratteri

Ogni carattere viene mappato in un numero intero (che è espresso da sequenza di bit) utilizzando dei codici

Il codice più usato è l'*ASCII (American Standard Code for Information Interchange)* a 7 o 8 bit che contiene:

- Caratteri alfanumerici
- Caratteri simbolici (es. punteggiatura, @&%\$ etc..)
- Caratteri di comando (es. termina riga, vai a capo, tab)



## Non solo numeri codifica dei caratteri

Nei calcolatori i caratteri vengono codificati mediante sequenze di  $n \geq 1$  bit, ognuna rappresentante un carattere distinto

- Corrispondenza biunivoca tra numeri e caratteri

Codice ASCII (American Standard Computer Interchange Interface): utilizza  $n=7$  bit per 128 caratteri

Il codice ASCII a 7 bit è pensato per la lingua inglese.

**Codifica ASCII esteso a 8 bit (256 parole di codice). È la più usata.** Rappresenta il doppio dei caratteri

- Si aggiungono così, ad esempio, le lettere con i vari gradi di accento (come À, Á, Â, Ã, Ä, Å, ecc..), necessarie in molte lingue europee, e altri simboli speciali ancora



## La codifica ASCII (parziale)

DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR
48	0	65	A	75	K	97	a	107	k
49	1	66	B	76	L	98	b	108	l
50	2	67	C	77	M	99	c	109	m
51	3	68	D	78	N	100	d	110	n
52	4	69	E	79	O	101	e	111	o
53	5	70	F	80	P	102	f	112	p
54	6	71	G	81	Q	103	g	113	q
55	7	72	H	82	R	104	h	114	r
56	8	73	I	83	S	105	i	115	s
57	9	74	J	84	T	106	j	116	t
				85	U			117	u
				86	V			118	v
				87	W			119	w
				88	X			120	x
				89	Y			121	y
				90	Z			122	z

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>:</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>



# Gli Operatori Aritmetici



## Operatori Aritmetici

Vi sono i soliti operatori aritmetici  $+$ ,  $-$ ,  $*$ ,  $/$  e le **parentesi tonde** per definire operazioni tra i valori delle variabili

Input	Output	Commento
<code>1234/6</code>	<code>ans= 205.6667</code>	calcolo di un valore scalare
<code>a=1234/6</code>	<code>a = 205.6667</code>	assegnamento alla variabile a del risultato di 1234/6
<code>2/5</code>	<code>ans = 0.40000</code>	divisione
<code>5/0</code>	<code>ans = Inf</code>	divisione per zero
<code>5^2</code>	<code>ans = 25</code>	potenza





## Operatori Aritmetici: resto della divisione intera

Un nuovo operatore aritmetico: resto della divisione intera, o *modulo*

- es. `mod(17,5)` vale 2, `mod(15,5)` vale 0



## Operatori Aritmetici: resto della divisione intera

Un nuovo operatore aritmetico: resto della divisione intera, o *modulo*

- es. `mod(17,5)` vale 2, `mod(15,5)` vale 0

Esempi:

```
a = 11; b = 4;
```

```
c = mod(a, 2);
```



## Operatori Aritmetici: resto della divisione intera

Un nuovo operatore aritmetico: resto della divisione intera, o *modulo*

- es. `mod(17,5)` vale 2, `mod(15,5)` vale 0

Esempi:

```
a = 11; b = 4;
```

```
c = mod(a, 2); (viene scritto in c il valore 1)
```



## Operatori Aritmetici: resto della divisione intera

Un nuovo operatore aritmetico: resto della divisione intera, o *modulo*

- es. `mod(17,5)` vale 2, `mod(15,5)` vale 0

Esempi:

```
a = 11; b = 4;
```

```
c = mod(a, 2); (viene scritto in c il valore 1)
```

```
a = 70; b = 5;
```

```
c = mod(a, (b + 2));
```



## Operatori Aritmetici: resto della divisione intera

Un nuovo operatore aritmetico: resto della divisione intera, o *modulo*

- es. `mod(17,5)` vale 2, `mod(15,5)` vale 0

Esempi:

```
a = 11; b = 4;
```

```
c = mod(a, 2); (viene scritto in c il valore 1)
```

```
a = 70; b = 5;
```

```
c = mod(a, (b + 2)); (viene scritto in c il valore 0)
```



## Operatori Aritmetici: resto della divisione intera

Un nuovo operatore aritmetico: resto della divisione intera, o *modulo*

- es. `mod(17,5)` vale 2, `mod(15,5)` vale 0

Esempi:

```
a = 11; b = 4;
```

```
c = mod(a, 2); (viene scritto in c il valore 1)
```

```
a = 70; b = 5;
```

```
c = mod(a, (b + 2)); (viene scritto in c il valore 0)
```

N.B: il valore di **b** non viene modificato, per modificare **b** :

```
b = b + 2; c = mod(a, b);
```



Sequenze di istruzioni Matlab



## Script (m-file)

Uno script è un **file di testo** contenente una **sequenza di comandi MATLAB**

- non deve contenere caratteri di formattazione (solo testo puro)
- viene salvato con estensione `.m`

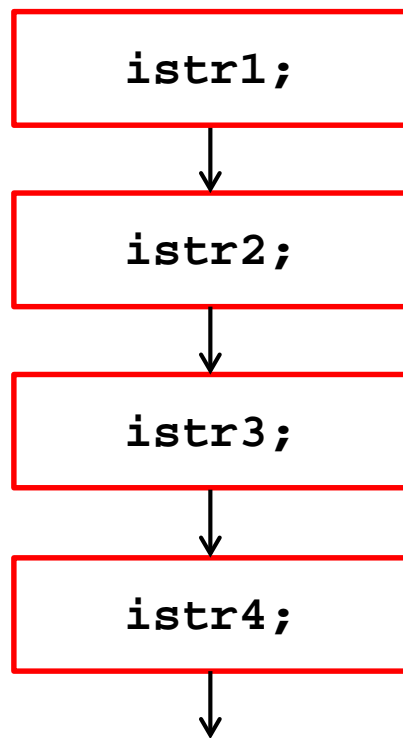
I comandi all'interno di uno script sono eseguiti **sequenzialmente**, come se fossero scritti nella finestra dei comandi

- Per eseguire il file (quindi la sequenza di istruzioni che contiene) si digita il suo nome (senza `.m`)
- I risultati appaiono nella finestra dei comandi (se non usiamo il `;`)





## Sequenze di istruzioni



`istr1;`

`istr2;`

`istr3;`

`istr4;`

`...`



## La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale**, dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

1. Istruzione1;
2. Istruzione2;
3. ...
4. IstruzioneN;

Es:

```
a = 45;
```

```
z = 5;
```

```
x = (a - z) / 10;
```

Stato della memoria

a ——— 45



## La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale**, dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

1. Istruzione1;
2. Istruzione2;
3. ...
4. IstruzioneN;

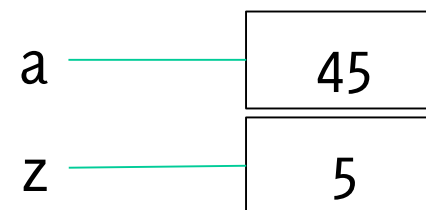
Es:

```
a = 45;
```

```
z = 5;
```

```
x = (a - z) / 10;
```


Stato della memoria





## La Sequenza di Istruzioni

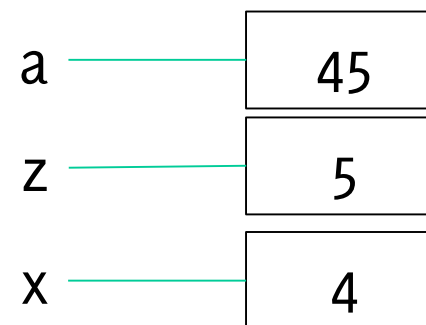
- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale**, dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

1. Istruzione1;
  2. Istruzione2;
  3. ...
  4. IstruzioneN;
- 

Es:

```
a = 45;  
z = 5;  
x = (a - z) / 10;
```

Stato della memoria





## Esecuzione di uno script

Uno script viene mandato in esecuzione (lanciato) inserendone il nome nella command window (senza estensione .m)

Occorre collocarsi nel folder contenente lo script prima di lanciarlo

È possibile mandare in esecuzione lo script anche premendo F5 dall'editor o utilizzando il bottone apposito nell'interfaccia grafica



## Istruzioni negli Script e ';'

Le istruzioni **possono terminare** con ';' ma non è obbligatorio

Di default, il risultato di ogni istruzione viene visualizzato nella command window.

Il ';' **blocca la visualizzazione** del risultato dell'istruzione

- Maggiore velocità di esecuzione
- Visualizzazione più compatta

Regola di buona programmazione

- Inserire sempre il ';', a meno che non si voglia ispezionare il valore di una variabile a scopo di debugging



# Input/output



## Acquisizione Dati da Tastiera (input)

Funzione input

```
valore = input(stringaDaVisualizzare);
```

Matlab stampa a video la **stringaDaVisualizzare** e attende un input in formato **Matlab**

- Un numero (i.e., uno scalare)
- Un carattere (delimitato da apici singoli)
- Array, se racchiuso tra [ e ], oppure
- Stringa, se racchiusa tra ' e ', oppure
- Una qualsiasi espressione Matlab



Vedremo  
nel seguito

Il dato inserito dall'utente viene memorizzato nella variabile **valore**

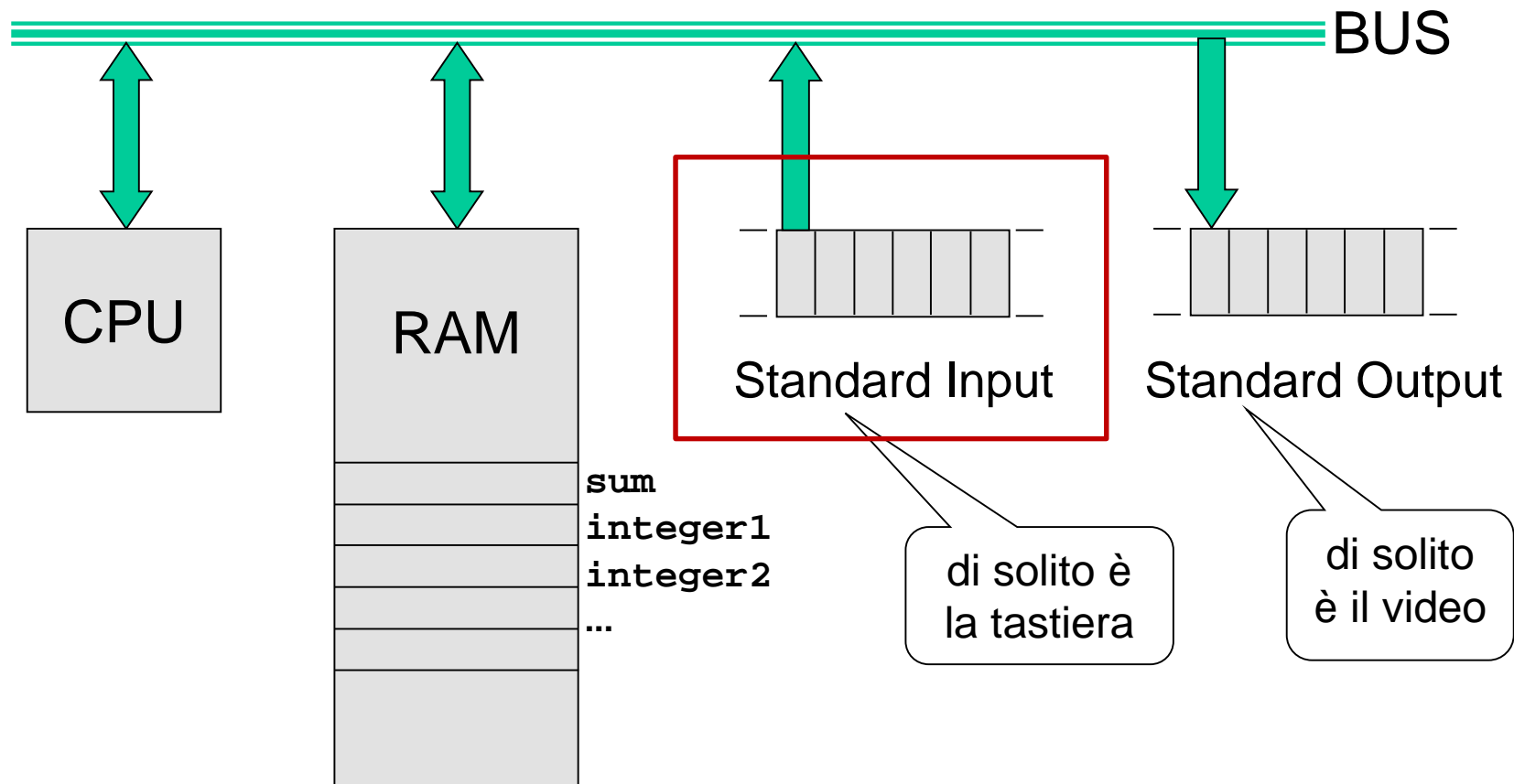
se **stringaDaVisualizzare** è una sequenza di caratteri, questa deve essere racchiusa tra apici singoli





# La macchina astratta C... ma andrà bene anche a noi

Algoritmi e programmi sono definiti in funzione del loro esecutore  
L'esecutore dei programmi C è una macchina astratta





## Scrittura a schermo con `fprintf`

Esempio:

```
fprintf( '\nInserire a: ' );
```

Sintassi:

```
fprintf ( stringaControllo );
```

- *stringaControllo* sequenze di caratteri (i.e., stringa) delimitata da doppi apici singoli `''`.
- Possono essere
  - **caratteri normali** (lettere, cifre, punteggiatura)
  - caratteri speciali (es, vai a capo)
  - Placeholders (e.g. `'%d'` per il contenuto di variabili)

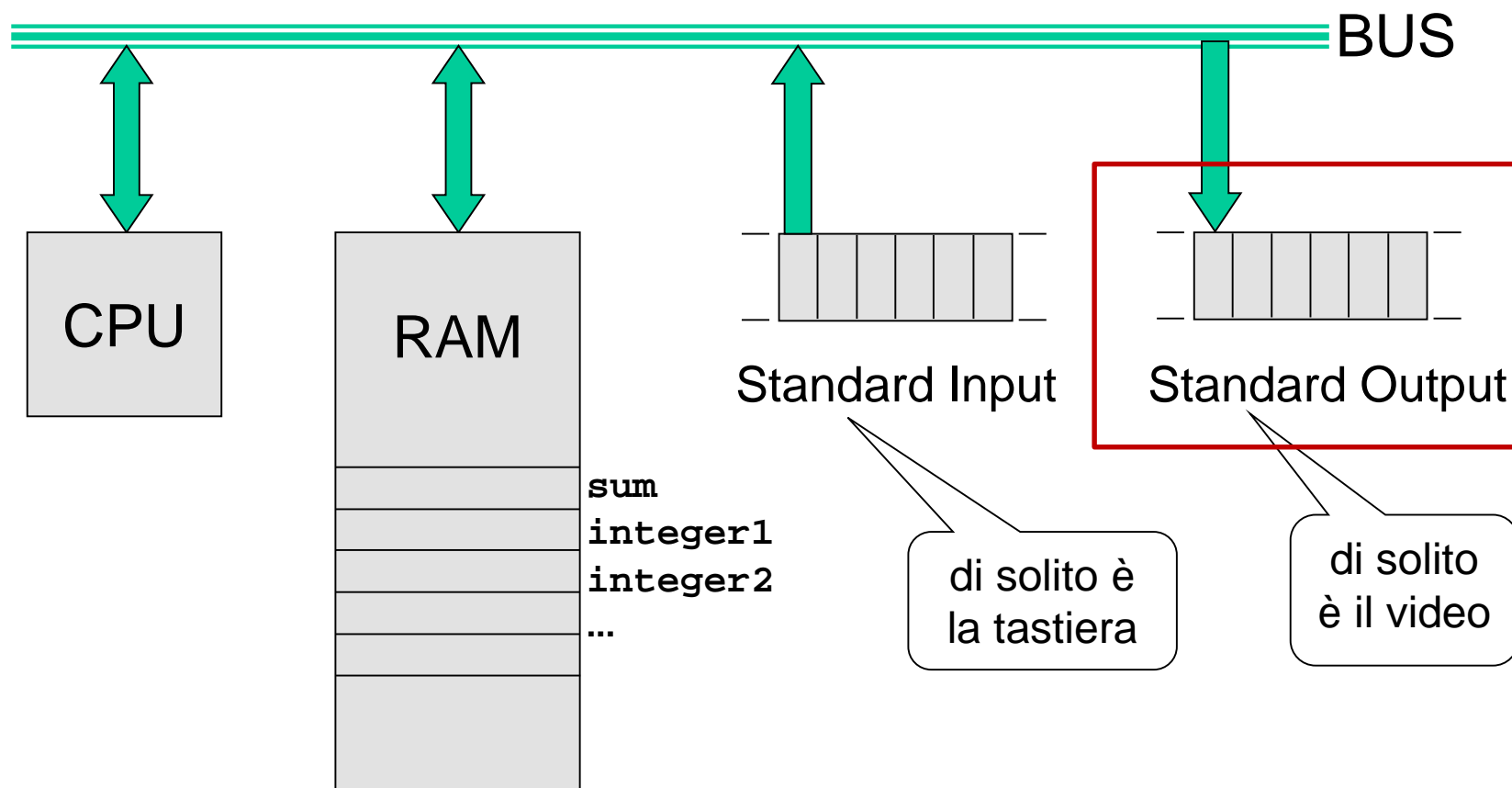
I caratteri nella *stringaControllo* vengono riportati a schermo.



# La macchina astratta C... ma andrà bene anche a noi

Algoritmi e programmi sono definiti in funzione del loro esecutore

L'esecutore dei programmi C è una macchina astratta





## Scrittura a schermo con `fprintf`

Usiamo il comando `fprintf`

```
fprintf('Il valore di pi e`%f \n', pi);
```

Stringhe di formato:

- `%d` (interi),
- `%e` (formato esponenziale),
- `%f` (formato decimale),
- `%g` (il più corto tra il formato esponenziale e decimale)

Per i caratteri si usa

- `%c` (un solo carattere)

Esempio:

```
fprintf( '\n %d + %d = %d', a, b, a+b );
```

Sintassi:

```
fprintf ( stringaControllo, elementiStampa );
```

- *stringaControllo* può contenere
  - caratteri di stampa (normali o speciali)
  - caratteri di conversione (segnaposto, convertono valori di variabili in caratteri per la stampa)
- *elementiStampa* elenco di **variabili, espressioni composte**, o costanti separati da virgole

Ogni elemento di *elementiStampa* viene convertito in caratteri e associato ai caratteri di conversione in *stringaControllo* in **nell'ordine con cui appare**.



## *stringaControllo:*

### Alcuni caratteri speciali per la stampa

- `'\n'` manda a capo
- `'\t'` spazia di un «tab»

### Alcuni caratteri di conversione

- `%d` intero decimale
- `%f` numero reale
- `%c` carattere
- `%s` sequenza di caratteri (stringa)



## Scrittura a schermo con `fprintf`

Esempi:

```
>> cat_dipend = 1;
```

```
>> stip_medio = 35623.5;
```

```
>> fprintf('Lo stipendio annuo dei dipendenti di categoria  
%d è pari a $%f\n', cat_dipend, stip_medio);
```



## Scrittura a schermo con `fprintf`

Esempi:

```
>> cat_dipend = 1;
```

```
>> stip_medio = 35623.5;
```

```
>> fprintf('Lo stipendio annuo dei dipendenti di categoria  
%d è pari a $%f\n', cat_dipend, stip_medio);
```

```
Lo stipendio annuo dei dipendenti di categoria 1 è pari a  
$35623.500000
```





## Scrittura a schermo con `fprintf`

Esempi:

```
>> cat_dipend = 1;
```

```
>> stip_medio = 35623.5;
```

```
>> fprintf('Lo stipendio annuo dei dipendenti di categoria  
%d è pari a $%f\n', cat_dipend, stip_medio);
```

Nella stampa `%d` verrà sostituito dal valore di `cat_dipend` mentre `%f` verrà sostituito dal valore di `stip_medio`.

L'abbinamento è dovuto **esclusivamente all'ordine** con cui appaiono i caratteri di conversione e le variabili (non al tipo).



## Scrittura a schermo con `fprintf`

Esempi:

```
>> iniz_nome = 'G';
```

```
>> iniz_cognome = 'B';
```

```
>> fprintf('Questo programma è stato scritto da  
\n%c%c\n\nBuon lavoro!\n', iniz_nome, iniz_cognome);
```



## Scrittura a schermo con `fprintf`

Esempi:

```
>> iniz_nome = 'G';
```

```
>> iniz_cognome = 'B';
```

```
>> fprintf('Questo programma è stato scritto da  
\n%c%c\n\nBuon lavoro!\n', iniz_nome, iniz_cognome);
```

```
Questo programma è stato scritto da
```

```
GB
```

```
Buon lavoro!
```



## Scrittura a schermo con `fprintf`

Esempi:

```
>> iniz_nome = 'G';
```

```
>> iniz_cognome = 'B';
```

```
>> fprintf( '%s\n%c%c\n\n%s\n', 'Questo programma è stato  
scritto da', iniz_nome, iniz_cognome, "Buon lavoro!");
```

È possibile specificare anche le stringhe (sequenze di caratteri) al di fuori della *stringaControllo*, purchè a queste si faccia riferimento con un carattere di conversione `%s`

**N.B:** non esiste un tipo di variabile built-in per contenere una stringa, occorrerà usare gli array.



Un primo semplice programma



## Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato



## Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato

```
clear
clc
% richiedo all'utente anno di nascita
anno = input('Inserire anno di nascita');
% calcolo anno di laurea
annodiLaurea = anno + 30;
% visualizzo a schermo l'anno di laurea
fprintf('Sei nato nel %d laurerai nel %d', anno,
annodiLaurea);
```



## Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato

```
clear
```

Pulisce il workspace

```
clc
```

```
% richiedo all'utente anno di nascita
```

```
anno = input('Inserire anno di nascita');
```

```
% calcolo anno di laurea
```

```
annodiLaurea = anno + 30;
```

```
% visualizzo a schermo l'anno di laurea
```

```
fprintf('Sei nato nel %d laurerai nel %d', anno,  
annodiLaurea);
```





## Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato

```
clear
```

```
clc
```

Pulisce la command window

```
% richiedo all'utente anno di nascita
```

```
anno = input('Inserire anno di nascita');
```

```
% calcolo anno di laurea
```

```
annodiLaurea = anno + 30;
```

```
% visualizzo a schermo l'anno di laurea
```

```
fprintf('Sei nato nel %d laurerai nel %d', anno,  
annodiLaurea);
```



## Programma 2

Scrivere un programma che richiede all'utente due numeri e ne calcola la somma



E qualche programma più elaborato



## Esempio di programmi

Scrivere un programma che prende in ingresso una temperatura in Fahrenheit e la trasforma in Celsius

$$C = 5/9 * (F - 32)$$



Scrivere un programma che richiede due caratteri che vengono salvati in opportune variabili.

Il programma poi scambia i contenuti delle variabili e ne stampa i valori.



## Esempio

```
clear
clc
% acquisisco nome
nome = input('inserisci iniziale nome');
% acquisisco cognome
cognome = input('inserisci iniziale cognome ');
% copio in temp il valore di nome
fprintf('\n\niniziale nome %c\niniziale cognome
%c', nome, cognome);
temp = nome;
% copio cognome in nome
nome = cognome;
% copio temp in cognome
cognome = temp;
% stampa
fprintf('\n\niniziale nome %c\niniziale cognome
%c', nome, cognome);
```

Scrivere un programma che richiede due caratteri che vengono salvati in opportune variabili.

Il programma poi scambia i contenuti delle variabili e ne stampa i valori.



# Tipo di Dato Logico



## Tipo di Dato Logico

È un tipo di dato che può avere solo due valori

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

I valori di questo tipo possono essere generati

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

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





## Esempi

```
>> a = true;
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x1	1	logical	

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

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

```
a =
```

```
0
```



## Operatori Relazionali

Operano su tipi numerici o stringhe.

Possono essere usati per confrontare

- due scalari
- due vettori aventi la stessa dimensione

Forma generale:  $a \text{ OP } b$

- $a, b$  devono essere variabili (o espressioni) della stessa dimensione
- OP:  $==, \neq, >, \geq, <, \leq$

Esempi:

- $3 < 4$  true(1)
- $3 == 4$  false(0)
- $'A' < 'B'$  true(1),  $'Z' < 'b'$  true(1),



**Attenzione:** non confondere `==` e `=`

- `==` è un operatore di confronto
- `=` è un operatore di assegnamento (con una direzione ben precisa)

Es: dire quali di queste espressioni ha senso

- `x = 7`
- `7 = x`
  
- `x == 7`
- `7 == x`



**Attenzione:** non confondere `==` e `=`

- `==` è un operatore di confronto
- `=` è un operatore di assegnamento (con una direzione ben precisa)

Es: dire quali di queste espressioni ha senso

- `x = 7` (**Ok**, è un assegnamento, se serve crea x)
- `7 = x` (**No**, l'assegnamento ha un verso, non posso assegnare un valore a 7)
- `x == 7` (**Ok**, è un confronto)
- `7 == x` (**Ok**, è un confronto)



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

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

Per i numeri piccoli conviene usare una soglia

- `abs( sin(pi) - sin(0) ) <= eps`

`eps` è una variabile built-in in Matlab e vale  $2.2204e-16$

`pi` è una variabile built-in ed è una buona approssimazione di  $\pi$



# 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

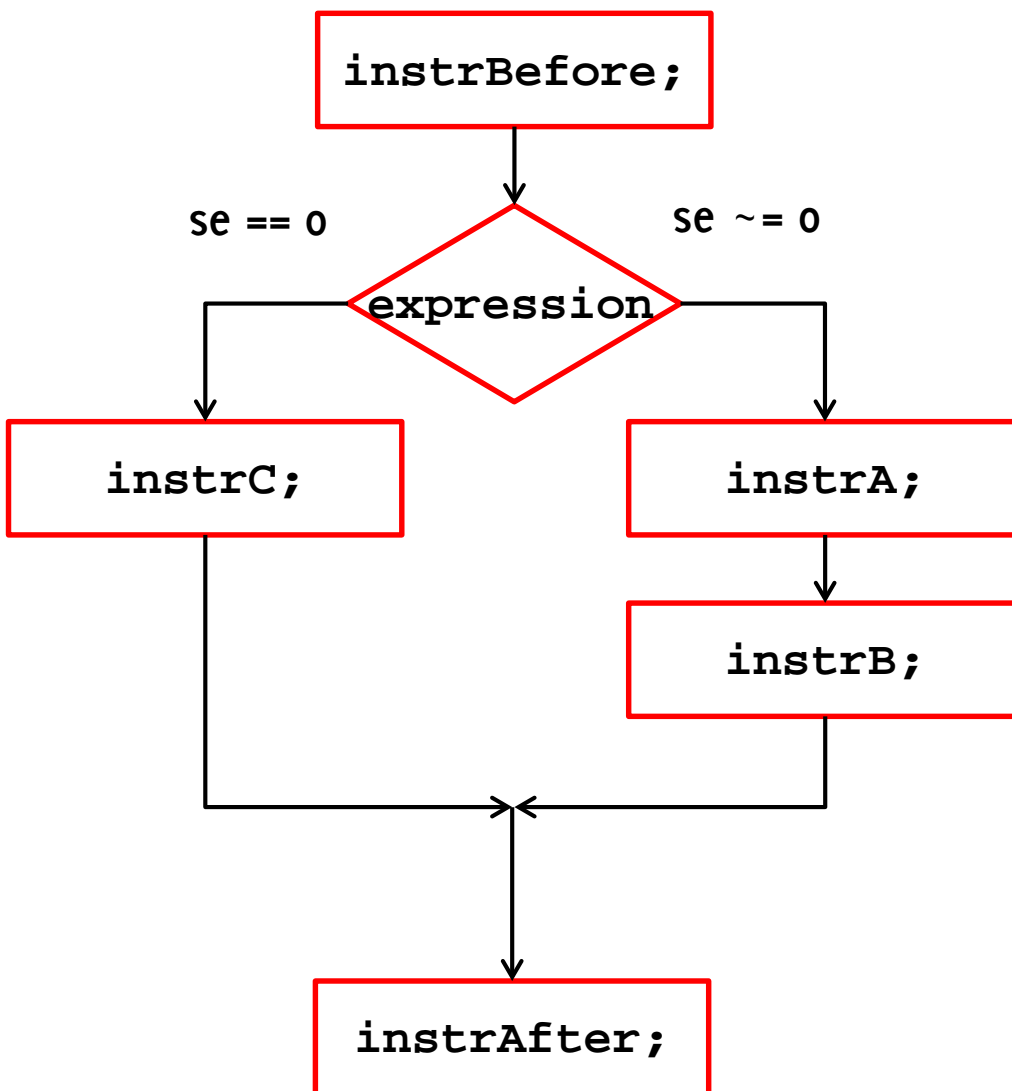
N.B **if(expression)** non richiede  
il ; perché l'istruzione non termina  
dopo )

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





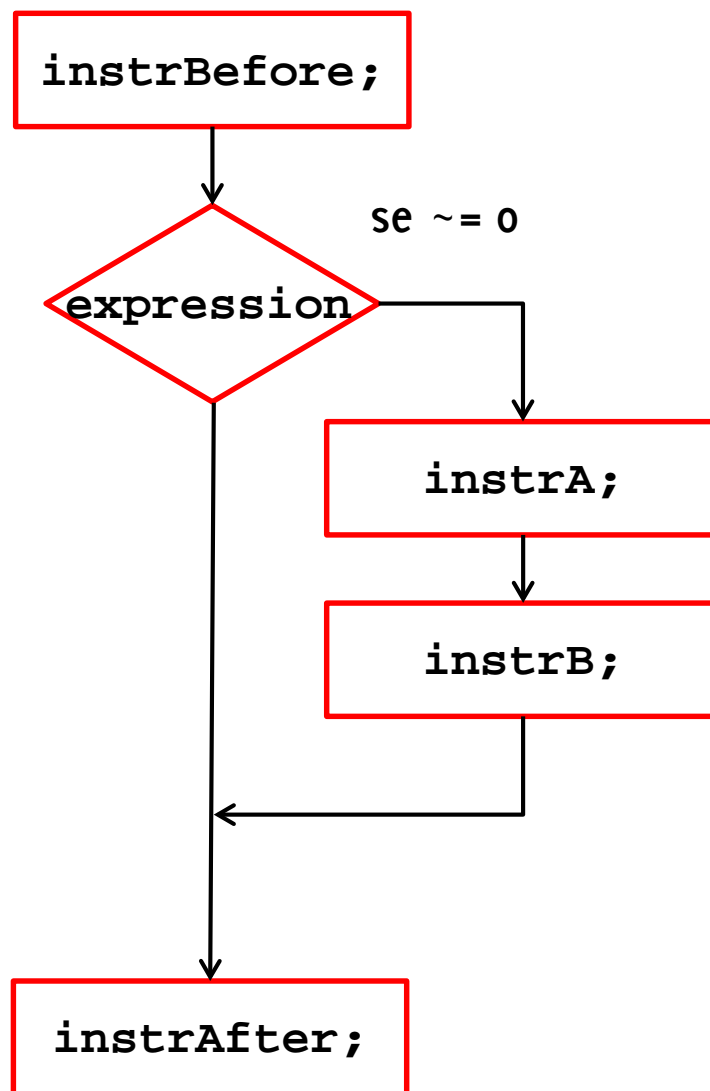
## Costrutto Condizionale: **if**, l'esecuzione



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



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



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



**%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
```



`%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?`



## if Annidati

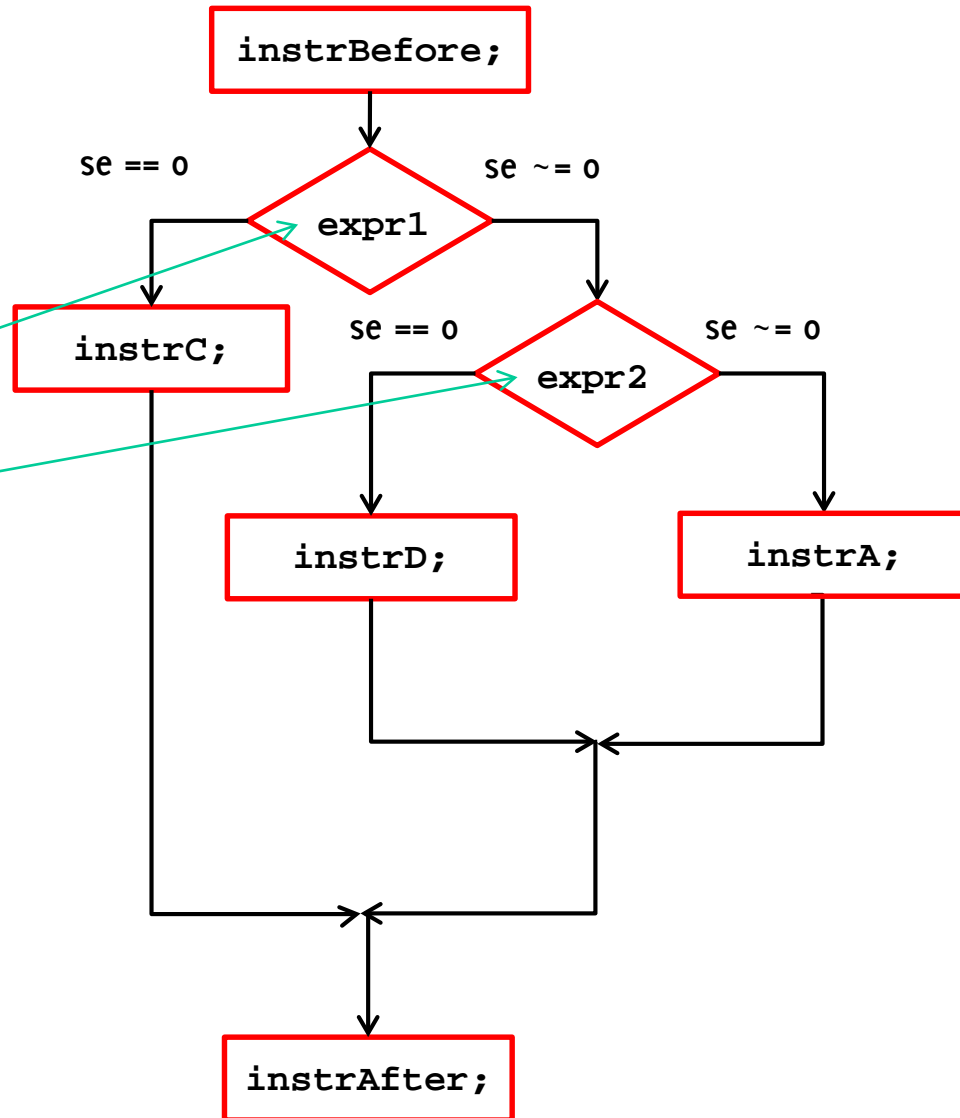
Il corpo di un **if** (cioè uno **statement**) può a sua volta contenere costrutti **if**: si realizzano quindi istruzioni condizionali **annidate**

```
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
```



## Valutare una condizione nell'else: `elseif`

`elseif` permette di valutare un'ulteriore condizione nel 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!



# 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)
```

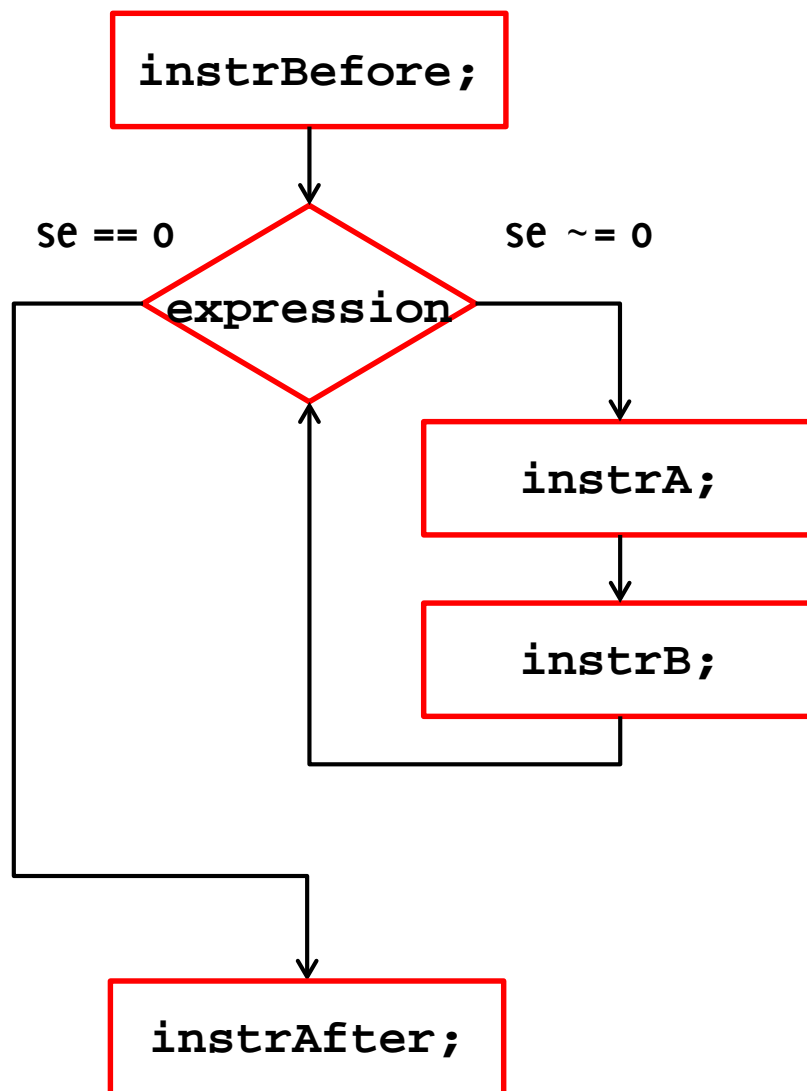
```
    statement;
```

```
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

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