

FONDAMENTI DI INFORMATICA

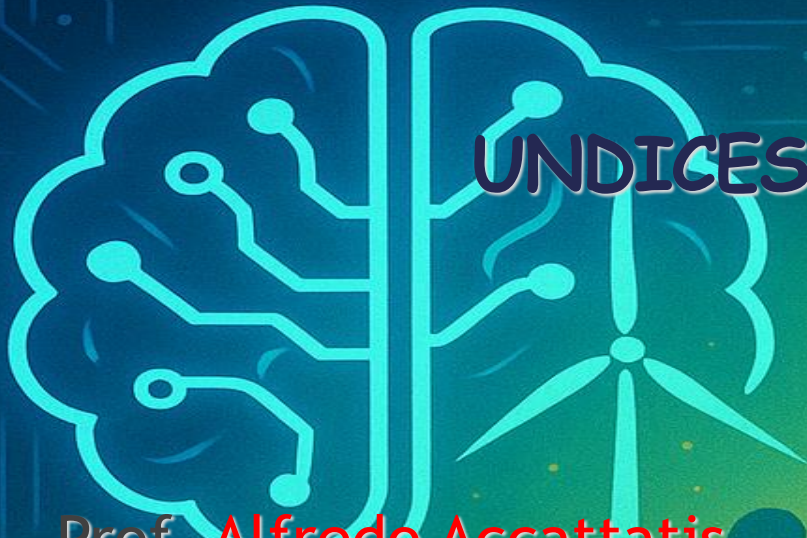
UNDICESIMA LEZIONE

Prof. **Alfredo Accattatis**

(accattatis@ing.uniroma2.it)

Tutor: Prof. **Vittorio Calafiore**

(vcalafio@gmail.com)



Definizione di Informatica (Computer Science)

- Le definizioni sono varie...ma simili....prendiamone una autorevole

Treccani:

- *Scienza che studia l'elaborazione delle informazioni e le sue applicazioni; più precisamente l'Informatica si occupa della rappresentazione, dell'organizzazione e del trattamento automatico della informazione*

"Computer science is no more about computers than astronomy is about telescopes." (L'informatica non riguarda i computer più di quanto l'astronomia riguardi i telescopi)

E. Dijkstra

Punto della situazione

- Informazione
- Rappresentazione (dell'informazione)
- **Organizzazione** (dell'informazione)
- Trattamento automatico (dell'informazione)
 - Algoritmi automatizzati

Organizzazione

-> Strutture dati

- Raccolta di dati organizzati per essere utilizzati da un computer
- Array:
 - Matrici
 - Vettori
 - CELL ARRAY

Cell Array: creazione

- Sintassi *simile* a quella usata per vettori e matrici:
 - uso di , e di ; per distinguere tra righe e colonne
 - uso di () e di { } per **cell-indexing** e **content-indexing**
 - (cella /contenuto) <=> (posizione del valore / valore)
 - { } e () al posto di []

```
>> Celle = {54, 'a'; [2.13 33.1 0.73 12.0], 'buongiorno'}
```

```
Celle =
```

```
2×2 cell array
```

```
{ [          54] }      { 'a' }  
{ [2.1300 33.1000 0.7300 12] }  { 'buongiorno' }
```

AIUTO!!! come digito le parentesi graffe?

AltGr Shift [**→** **{**
AltGr Shift] **→** **}**

Cell Array: creazione (2)

- È possibile inserire un elemento alla volta: poco efficiente
- L'utilizzo della funzione built-in `cell` è raccomandato, se si conoscono a priori le dimensioni

```
>> Raccolta = cell(3,3)
```

```
Raccolta =
```

```
3×3 cell array
```

```
{0×0 double}    {0×0 double}    {0×0 double}  
{0×0 double}    {0×0 double}    {0×0 double}  
{0×0 double}    {0×0 double}    {0×0 double}
```

Cell Array: assegnazione e modifica

- Content-indexing - si fa riferimento al **valore** della cella; è supportato l'indirizzamento lineare per array multidimensionali (columnwise)

```
>> Raccolta{2,1} = 'Salve'
      Raccolta =
```

3×3 cell array

{0×0 double}	{0×0 double}	{0×0 double}
{ 'Salve' }	{0×0 double}	{0×0 double}
{0×0 double}	{0×0 double}	{0×0 double}

```
>> Raccolta{2} = 'Ciao'
```

```
Raccolta =
```

3×3 cell array

{0×0 double}	{0×0 double}	{0×0 double}
{ 'Ciao' }	{0×0 double}	{0×0 double}
{0×0 double}	{0×0 double}	{0×0 double}

Cell Array: assegnazione e modifica (2)

- Cell-indexing - si fa riferimento alla cella (**contenitore o posizione**)

```
>> Raccolta (0,3) = Raccolta(2)
      Raccolta =
      3x3 cell array
```

```
{0x0 double}      {0x0 double}      {'Ciao'      }
{'Ciao'          }      {0x0 double}      {0x0 double}
{0x0 double}      {0x0 double}      {0x0 double}
```

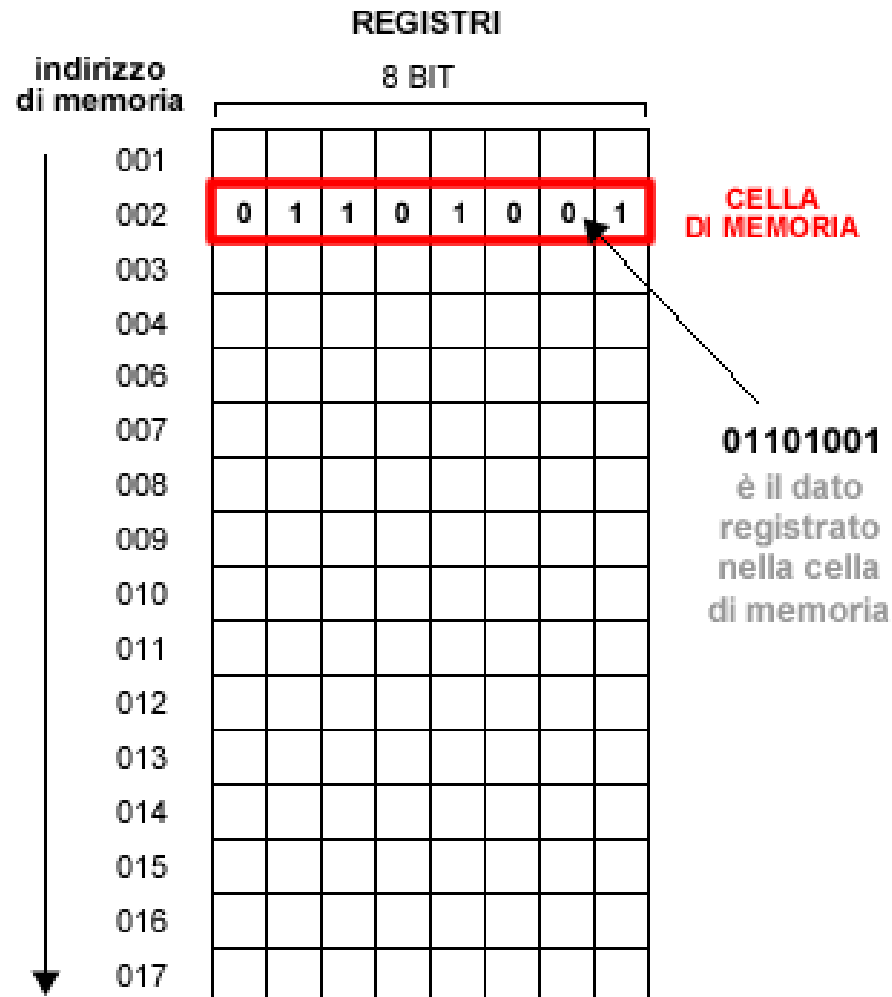
```
>> Raccolta(7) = 'Addio'
```

Conversion to cell from char is not possible.

```
>> Raccolta(7) = {'Addio'}
      3x3 cell array
```

```
{0x0 double}      {0x0 double}      {'Addio'      }
{'Ciao'          }      {0x0 double}      {0x0 double}
{0x0 double}      {0x0 double}      {0x0 double}
```

Puntatori: cell indexing = indirizzo



Cell Array: assegnazione e modifica (3)

- ESEMPIO : assegno i valori agli elementi

```
[r,c]=size(Raccolta);
```

```
k = 1;
```

```
>> for i=1:r for j=1:c Raccolta{i,j}=k; k=k+1; end; end
```

```
>> Raccolta =
```

```
3×3 cell array
```

```
{[1]}      {[2]}      {[3]}
```

```
{[4]}      {[5]}      {[6]}
```

```
{[7]}      {[8]}      {[9]}
```

```
>> Raccolta{1} = 'Primo'; Raccolta{9} = 'Ultimo';
```

```
3×3 cell array
```

```
{'Primo'}      {[2]}      {[          3]}
```

```
{[          4]}      {[5]}      {[          6]}
```

```
{[          7]}      {[8]}      {'Ultimo'}
```

Cell Array: assegnazione e modifica (4)

- Torniamo all'esempio iniziale:

```
>> Celle = {54, 'a'; [2.13 33.1 0.73 12.0], 'buongiorno'}
```

```
2x2 cell array
```

```
{ [          54] }      { 'a'          }  
{ [2.1300 33.1000 0.7300 12] }    { 'buongiorno' }
```

- Come accedo alle componenti (singole) del vettore?

```
>> Celle{2}
```

```
ans =
```

```
2.13 33.1 0.73 12.0
```

Cell Array: assegnazione e modifica (5)

```
{54, 'a'; [2.13 33.1 0.73 12.0], 'buongiorno'}
```

```
>> Cella{2}(1)
```

```
ans =
```

```
2.13
```

```
>> Cella{2}(2)
```

```
ans =
```

```
33.1
```

Cell Array: ispezione

- Contenuti - elenca i contenuti columnwise

```
>> celldisp(Raccolta)
```

```
Raccolta{1,1} =
```

```
Primo
```

```
Raccolta{2,1} =
```

```
4
```

```
Raccolta{3,1} =
```

```
7
```

```
Raccolta{1,2} =
```

```
2
```

```
.....
```

- **Rappresentazione grafica**

- `>> cellplot(Raccolta)`

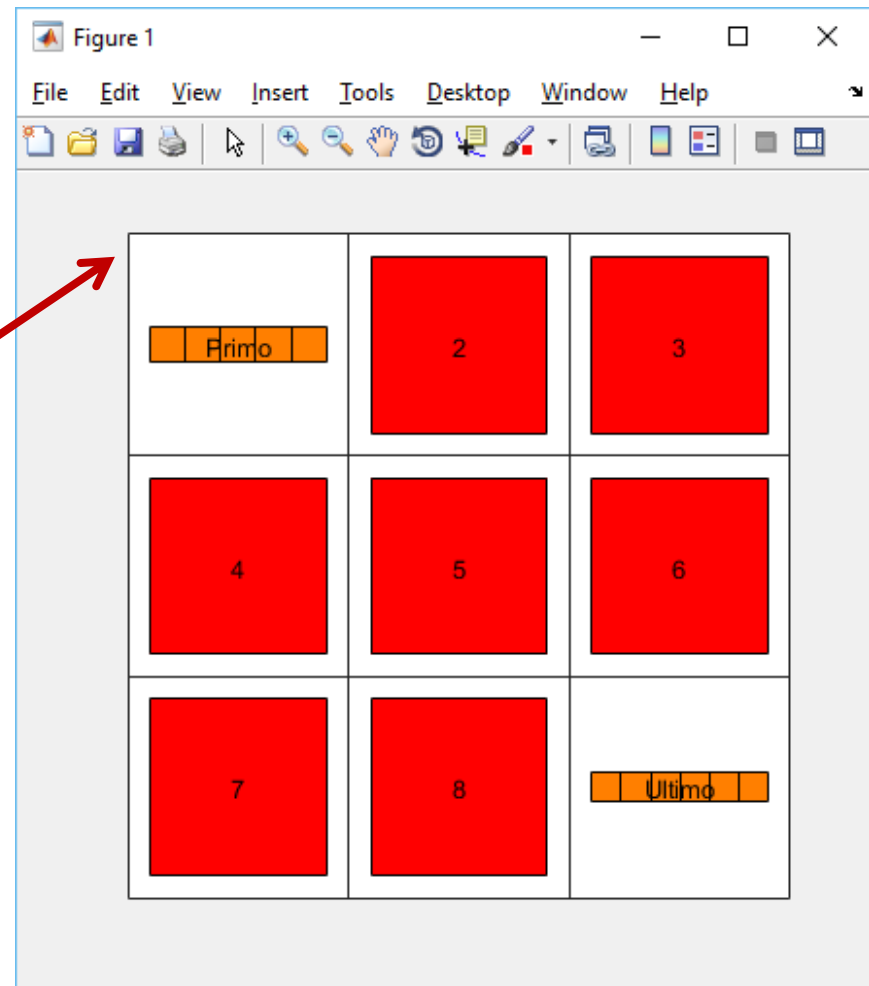
- **Tipo di dato**

- `>> class(Raccolta)`

- `ans = cell`

- `>> class(Raccolta{2})`

- `ans = double`



Cell Array: cancellazione

- Cell array monodimensionale (vettore)

```
>> rigaRaccolta = Raccolta(1,:)
      rigaRaccolta =
          1×3 cell array

      {'Primo'}      {[2]}      {[3]}
>> rigaRaccolta(1)=[]
      rigaRaccolta =
          1×2 cell array

      {[2]}      {[3]}
```

Cella array bidimensionale (matrice)

```
>> Raccolta(1, :) = []

      Raccolta =
          { [4] } { [5] } { [6] }
          { [7] } { [8] } { ['Ultimo'] }
```

Cell Array: utilizzi e cellstr

- Collezione di stringhe di lunghezza diversa: sono tipi di dato differenti (array di char).

- ESEMPIO : ordinare alfabeticamente una raccolta di 5 parole

```
>> stringa1 = 'Cane'; stringa2 = 'Gatto'; stringa3 = 'Oca';
```

```
>> whos stringa1 stringa2 stringa3
```

Name	Size	Bytes	Class
stringa1	1x4	8	char
stringa2	1x5	10	char
stringa3	1x3	6	char

```
>> parole = cell(5,1);
```

```
>> parole(1)=cellstr(stringa3); parole(2)={'Ciao'};
```

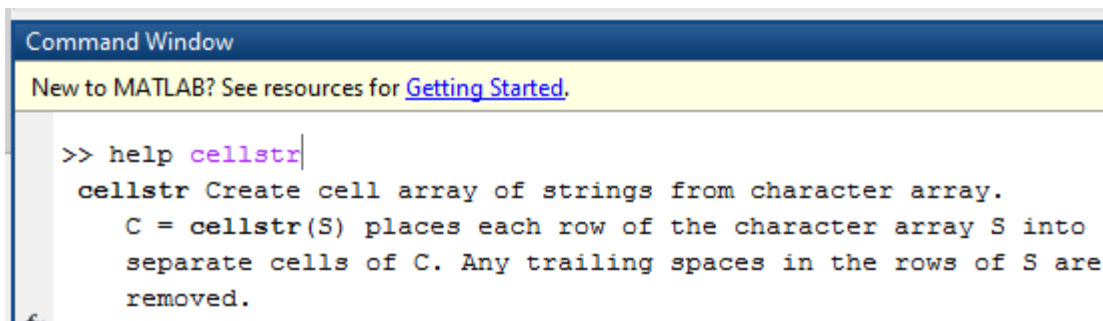
```
>> parole(3)=cellstr(stringa1); parole{4}='Saluti';
```

```
>> parole(5)=cellstr(stringa2);
```

```
>> paroleOrdinate = sort(parole)
```

```
5x1 cell array
```

```
{ 'Cane' }
{ 'Ciao' }
{ 'Gatto' }
{ 'Oca' }
{ 'Saluti' }
```



Command Window

New to MATLAB? See resources for [Getting Started](#).

```
>> help cellstr
cellstr Create cell array of strings from character array.
C = cellstr(S) places each row of the character array S into
separate cells of C. Any trailing spaces in the rows of S are
removed.
```

Structure

- Struttura di dati MATLAB organizzata come raggruppamento di valori caratterizzati da un legame logico.
- Può contenere tipi di dato differenti: ciascun elemento è memorizzato in un campo (*field*).
- Il campo può essere di qualunque tipo supportato da MATLAB: p.e. carattere, stringa, numero, vettore, matrice, cell array, ecc.
- La *structure* MATLAB non è un array
 - Non si possono utilizzare indici numerici per manipolare i suoi elementi
 - Non si può utilizzare codice “vettorializzato” (cicli iterativi)

Structure: creazione

```
S = struct('campo1',Valore1, 'campo2',Valore2,...)
```

La struttura è definita da:

1. Nome del field **campoN**
2. Valore del field **ValoreN**
3. Qualunque tipo di dato (classe) per **ValoreN**

```
>> Studente = struct('Nome', 'Mario', 'Cognome', 'Rossi', 'eta', 20)
```

```
Studente =
```

```
    Nome: 'Mario'
```

```
  Cognome: 'Rossi'
```

```
      eta: 20
```

Structure: manipolazione

- **Modifica valore - dot operator :**

```
>> Studente.Nome='Giorgio'
Studente =
    Nome: 'Giorgio'
    Cognome: 'Rossi'
    eta: 20
```

- **Copia struttura - crea una nuova struttura identica:**

```
>> AltroStudente=Studente
AltroStudente =
    Nome: 'Giorgio'
    Cognome: 'Rossi'
    eta: 20
```

Structure: manipolazione (2)

- Visualizzazione contenuti

```
>> disp(Studente)
```

```
Nome: 'Giorgio'
```

```
Cognome: 'Rossi'
```

```
eta: 20
```

```
>> fprintf('%s %s %d\n',Studente.Nome,Studente.Cognome,Studente.eta)
```

```
Giorgio Rossi 20
```

- Estrazione dei nomi dei campi - crea un cell array:

```
>> campi=fieldnames(Studente);
```

```
>> campi
```

```
campi =
```

```
    'Nome'
```

```
    'Cognome'
```

```
    'eta'
```

```
>> class (campi)
```

```
ans =
```

```
cell
```

Structure: manipolazione (3)

- Rimozione campo - fornisce una nuova struttura:

```
>> rmfield(AltroStudiante, 'eta')
```

```
ans =
```

```
Nome: 'Giorgio'
```

```
Cognome: 'Bianchi'
```

```
>> AltroStudiante
```

```
AltroStudiante =
```

```
Nome: 'Giorgio'
```

```
Cognome: 'Bianchi'
```

```
eta: 20
```

Se si vuole rimuovere un campo da una struttura:

```
>> AltroStudiante = rmfield(AltroStudiante, 'eta')
```

Structure: manipolazione (4)

- Aggiunta campo:

```
>> Studente.voto=30
```

```
Studente =
```

```
    Nome: 'Giorgio'
```

```
  Cognome: 'Rossi'
```

```
    eta: 20
```

```
    voto: 30
```

Structure: vettori di strutture (struct array)

- Permette di memorizzare dati eterogenei strutturati e di utilizzarli come elementi indicizzati (cicli iterativi)
- Creazione tramite estensione di struttura già esistente:

```
>> Studente(2) = struct('Nome', 'Marco', ...  
                        'Cognome', 'Verdi', 'eta', 21);
```

- Creazione tramite replicazione:

- si crea la prima struttura

```
>> Studente = struct('Nome', 'Mario', ...  
                    'Cognome', 'Rossi', 'eta', 20);
```

- poi si replica con la dimensione necessaria - tutti elementi uguali

```
>> Studente = repmat(Studente, 1, 15);
```

- si modificano i valori

```
>> Studente(2) = struct('Nome', 'Giorgio', ...  
                        'Cognome', 'Bianchi', 'eta', 21);
```

.....

Structure: vettori di strutture (2)

- Creazione efficiente tramite pre-allocazione (si parte iniziando dall'ultimo elemento):

```
>> Studente(15) = struct('Nome', 'Enrico', ...  
                        'Cognome', 'Marchi', 'eta', 19);  
>> Studente(1) = struct('Nome', 'Mario', ...  
                        'Cognome', 'Rossi', 'eta', 20);  
>> Studente(2) = struct('Nome', 'Giorgio', ...  
                        'Cognome', 'Bianchi', 'eta', 21);
```

.....

Studente		
Nome	Cognome	Età
Mario	Rossi	20
Giorgio	Bianchi	21
...	...	...
Enrico	Marchi	19

Structure: vettori di strutture (3)

- Accesso ai dati

```
>> Studente
Studente =
1x15 struct array with fields:
    Nome
    Cognome
    eta
```

```
>> Studente(2)
ans =
    Nome: 'Giorgio'
    Cognome: 'Bianchi'
    eta: 20
```

```
>> Studente(1).Cognome
ans =
    Cognome: 'Rossi'
```

Structure: vettori di strutture (4)

- Possiamo "ciclare" tra gli elementi

```
function etaMedia(Studiante)
e = 0;
n=length(Studiante);
fprintf('\n%-10s %-10s %-10s', 'Cognome', 'Nome', 'Età')
for i=1:n
    %Incrementa l'età totale
    e = e + Studiante(i).eta;
    fprintf('\n%-10s %-10s %-10d', Studiante(i).Cognome, ...
        Studiante(i).Nome, Studiante(i).eta)
end
e=e/n;
fprintf('\nL' età media è di %3.1f anni.\n', e);
end
```

```
>> etaMedia(Studiante)
```

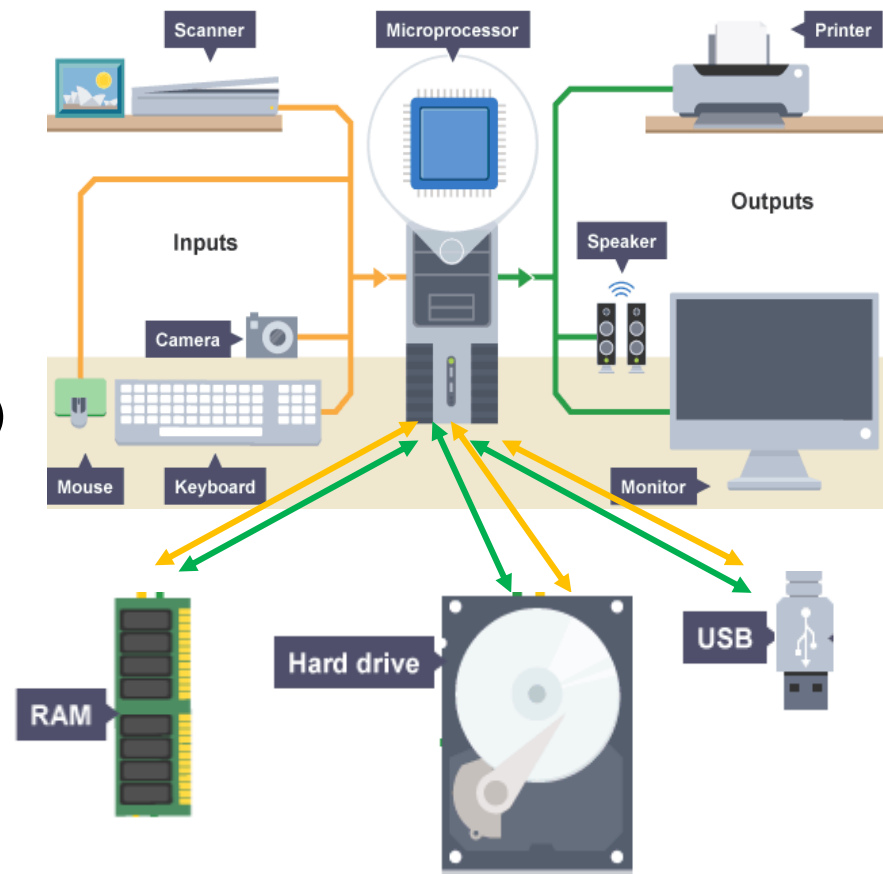
Cognome	Nome	Età
Bianchi	Giuseppe	23
Marchi	Gino	19
Dalidi	Enrico	20
Lurci	Andrea	24
Rossi	Mario	20

L' età media è di 21.2 anni.

```
fx >>
```

I/O - Input & Output

- Input: dati in ingresso a CPU
- Output: dati in uscita da CPU
- I/O da periferiche in MATLAB:
 - `input()`
 - `disp()`, `fprintf()`, `plot()`
- I/O da/a un file contenente dati:
 - Input: leggere dal file
 - Output: scrivere nel file



I/O Basilare - load & save

- Input: comando `load`

```
>> load nomefile.ext
```

- Crea una matrice (NxM) di nome `nomefile`: ciascun elemento contiene un dato letto dal file.
- Il file deve essere strutturato in maniera **regolare**: tabella righe-colonne cioè stesso numero di dati su ciascuna riga.

- Output: comando `save`

```
>> save nomefile.ext matriceDati -ascii
```

- Crea un file di dati o testo in formato ASCII contenente i valori di `matriceDati`. Se il file già esiste viene **sovrascritto**.
- Per aggiungere i dati di `matriceDati` ad un file esistente si devono «appendere» questi dati a quelli già esistenti: si usa il qualificatore `-append`.

```
>> save nomefile.ext matriceDati -ascii -append
```

I/O Generalizzato

- Per poter accedere ai contenuti di un file con struttura generica (non regolare), p.e. file di testo o file con dati numerici e testo/caratteri, è necessario:
 1. **Aprire** il file - il file viene cercato (per lettura) o creato (per scrittura)
 2. **Leggere** dal file, **scrivere** nel file, **appendere** al file
 3. **Chiudere** il file
- MATLAB supporta svariati tipologie di comandi e funzioni per svolgere I/O da file. Consideriamo: `textscan()`, `fscanf()` e `fprintf()`.

I/O Generalizzato: aprire un file

- L'apertura del file (ricerca o creazione) si esegue mediante la funzione **fopen**

FID = fopen('nomefile.ext', 'permessi')

1. **FID** è numero intero che identifica il file. Se **FID = -1** allora il file non esiste.
2. **nomefile.ext** è il nome del file, relativo al percorso di ricerca configurato (in genere MATLABPATH)
3. **permessi** modalità di accesso al file,
 - 'r' per leggere
 - 'w' per scrivere
 - 'a' per appendere

I/O Generalizzato: chiudere un file

- La chiusura del file va eseguita se non si devono fare più accessi, e comunque prima che termini l'esecuzione del programma, utilizzando la funzione **fclose**

esitoChiusura = fclose(FID)

1. **esitoChiusura** è un valore pari a 0 se la chiusura è andata a buon fine, altrimenti pari a -1 , p.e. nel caso di file corrotto. Se si utilizza come parametro la stringa '**all**', tutti i file aperti verranno chiusi.
2. **FID** è l'identificativo del file assegnato in apertura.

I/O Verifiche apertura/chiusura

Buona norma è verificare che apertura e chiusura sono state completate correttamente, e nel caso contrario dare indicazioni all'utente.

```
fid = fopen('nomefile', 'r' ); % apro in lettura
if fid == -1
    disp('Apertura del file fallita!')
else
    % qui vanno le istruzioni per accedere al file
    % ed elaborare i dati (es. Con textscan o fscanf)
    % ...

esitoChiusura = fclose(fid);
if esitoChiusura ~= 0
    disp('Chiusura file fallita!')
end
end
```

I/O Generalizzato: lettura (1)

- La lettura del file può eseguire mediante la funzione **textscan**, che trasferisce tutti i dati del file in un *Cell Array* in ordine per colonna.

Lettura = textscan(FID, 'formato')

1. **Lettura** è il cell-array di destinazione. La lettura terminerà quando si raggiunge l'*end-of-file* (EOF).
2. **FID** è l'identificativo del file.
3. **formato** specifica la struttura di ciascuna riga del file: le righe devono essere formattate compatibilmente (%s, %d etc.)

I/O Generalizzato: lettura (2)

- La lettura del file può eseguire mediante la funzione **fscanf**, che trasferisce tutti i dati del file in un *Array* in ordine per colonna.

ArrayLettura = fscanf(FID, 'formato')

1. **Lettura** è l'array di destinazione (matrice o vettore). La lettura terminerà quando si raggiunge l'*end-of-file* (EOF) oppure dato di tipo non previsto.
2. **FID** è l'identificativo del file.
3. **formato** specifica la struttura di ciascuna riga del file: le righe devono essere formattate compatibilmente (%s, %d etc.)

Creazione file auto.txt

- Panda 20
- FerrariGTO 5
- Ibrida 30
- Cinquecento 22
- Shuttle 0.001

I/O Generalizzato: lettura (3)

- ESEMPIO: Elencare i comandi per leggere dal file 'auto.txt' i nomi di modelli di autovetture e consumi di carburante dichiarati per ciclo urbano (km/l). Calcolare il consumo medio.

```
>> IDFile = fopen('auto.txt')
      IDFile =
      5
>> dati = textscan(IDFile, '%s %f')
      dati =
      1x2 cell array
      {5x1 cell} {5x1 double }
>> consumo = dati{1, 2};
>> consumoMedio = mean(consumo)
consumoMedio =
      15.4002
>> fclose(IDFile)
ans = 0
```

- Panda 20
- FerrariGTO 5
- Ibrida 30
- Cinquecento 22
- Shuttle 0.001

I/O Generalizzato: scrittura (1)

- La scrittura del file si può eseguire mediante la funzione `fprintf`; **`fprintf`** restituisce il numero dei byte *effettivamente* trasferiti.

`esito= fprintf(FID, 'formato', variabili)`

1. **FID** è l'identificativo del file, che è stato aperto con permesso di scrittura. Per appendere dei dati è necessario aprire il file con permesso 'a'. Se viene omesso i dati sono trasferiti al monitor, essendo il dispositivo di output di default.
2. **formato** specifica la struttura di ciascuna riga del file.
3. **variabili** i dati da scrivere
4. **esito** il numero di byte scritti oppure cod. errore

I/O Generalizzato: scrittura (2)

ESEMPIO: Scrivere i comandi per scrivere il file 'alti_bassi.txt' contenente i nomi modelli delle autovetture del precedente esempio affiancati da 'A' se superano il consumo medio calcolato, 'B' se si trovano al di sotto.

I/O Generalizzato: scrittura (3)

```
>> IDFile = fopen('alti_bassi.txt', 'w')
        IDFile =
            7
>> modello = dati{1, 1};
>> for i = 1:length(modelli)
        if consumo[i] > consumoMedio
            codice = 'A';
        else
            codice = 'B';
        end;
        fprintf(IDFile, ' %s %c', modello(i), codice);
    end;
>> fclose(IDFile)
        ans = 0
```