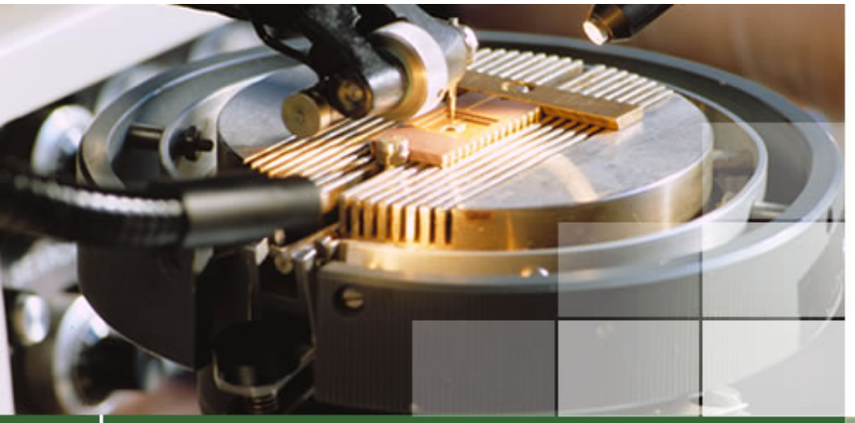


POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

# Memorie e MATLAB, di tutto un po'!

Matteo Ferroni

matteo.ferroni@polimi.it

21/12/2018

# Gerarchia di memoria: definizioni (ripasso)

- **Hit (successo)**: dati presenti in un blocco del livello superiore (esempio: Blocco X)
  - Hit **Rate** (**"%"** di successo): numero di accessi a memoria che trovano il dato nel livello superiore sul numero totale di accessi
  - Hit **Time** (**tempo** di successo): tempo per accedere al dato nel livello superiore della gerarchia

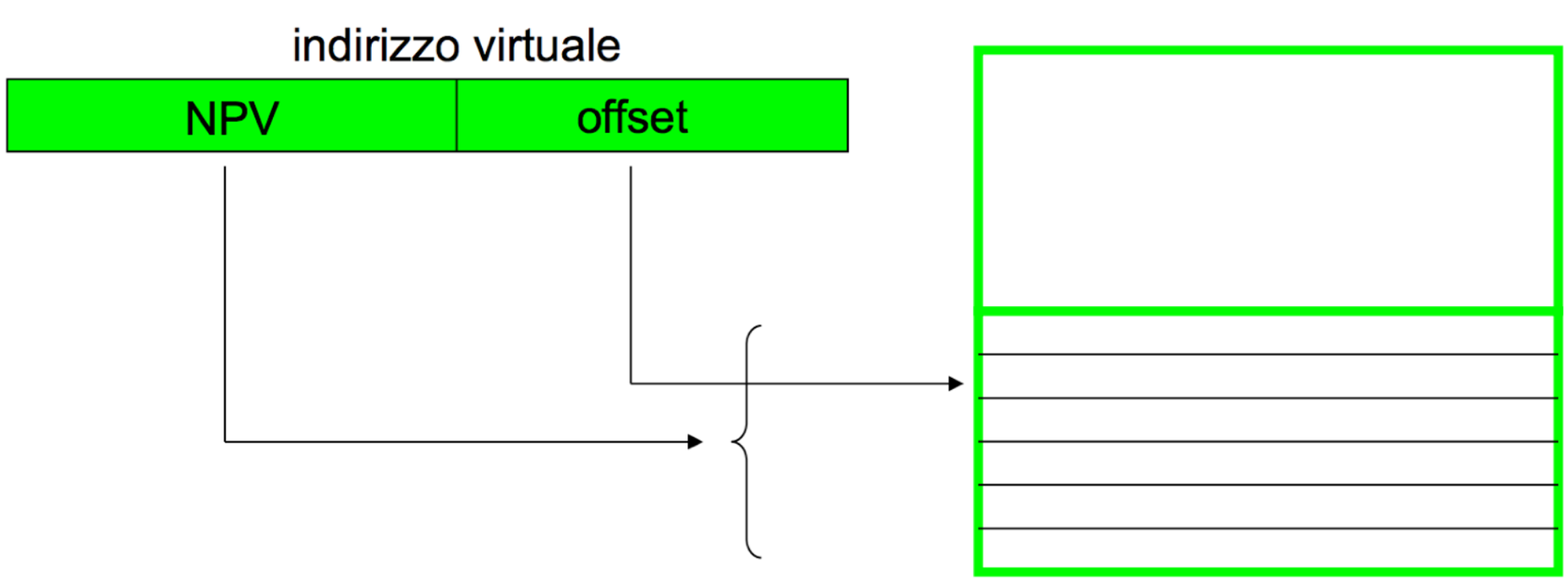
- **Miss (fallimento)**: i dati devono essere recuperati dal livello inferiore della memoria (Blocco Y)
  - Miss **Rate** (**"%"** di fallimento) = **1 - (Hit Rate)**
  - Miss **Penalty** (**tempo** di fallimento): tempo per determinare il MISS + tempo necessario a sosituire un blocco nel livello superiore + tempo per trasferire il blocco al processore

- **Tempo medio** di accesso in presenza di memoria cache:

**HitTime*HitRate + MissRate*MissPenalty**

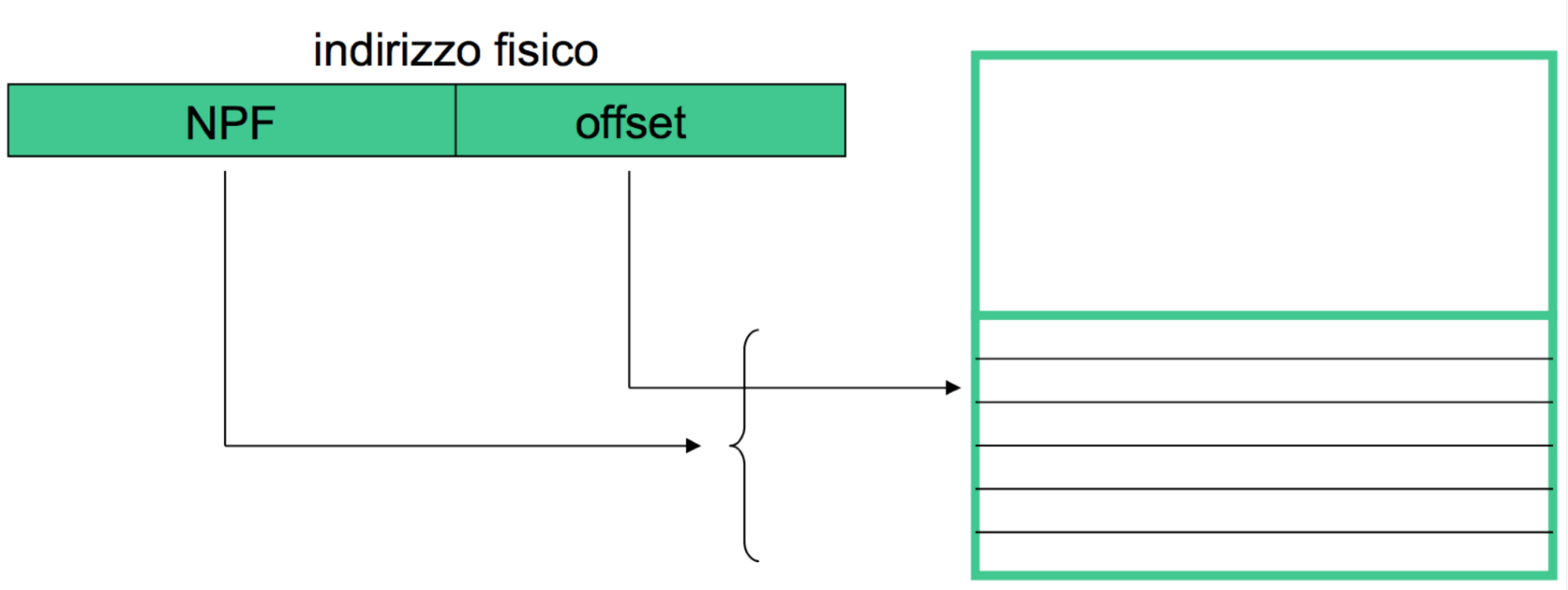
# Memoria fisica e memoria virtuale (ripasso)

- Un indirizzo virtuale è costituito da un numero di pagina virtuale (NPV) e da uno spiazzamento (offset) all'interno della pagina

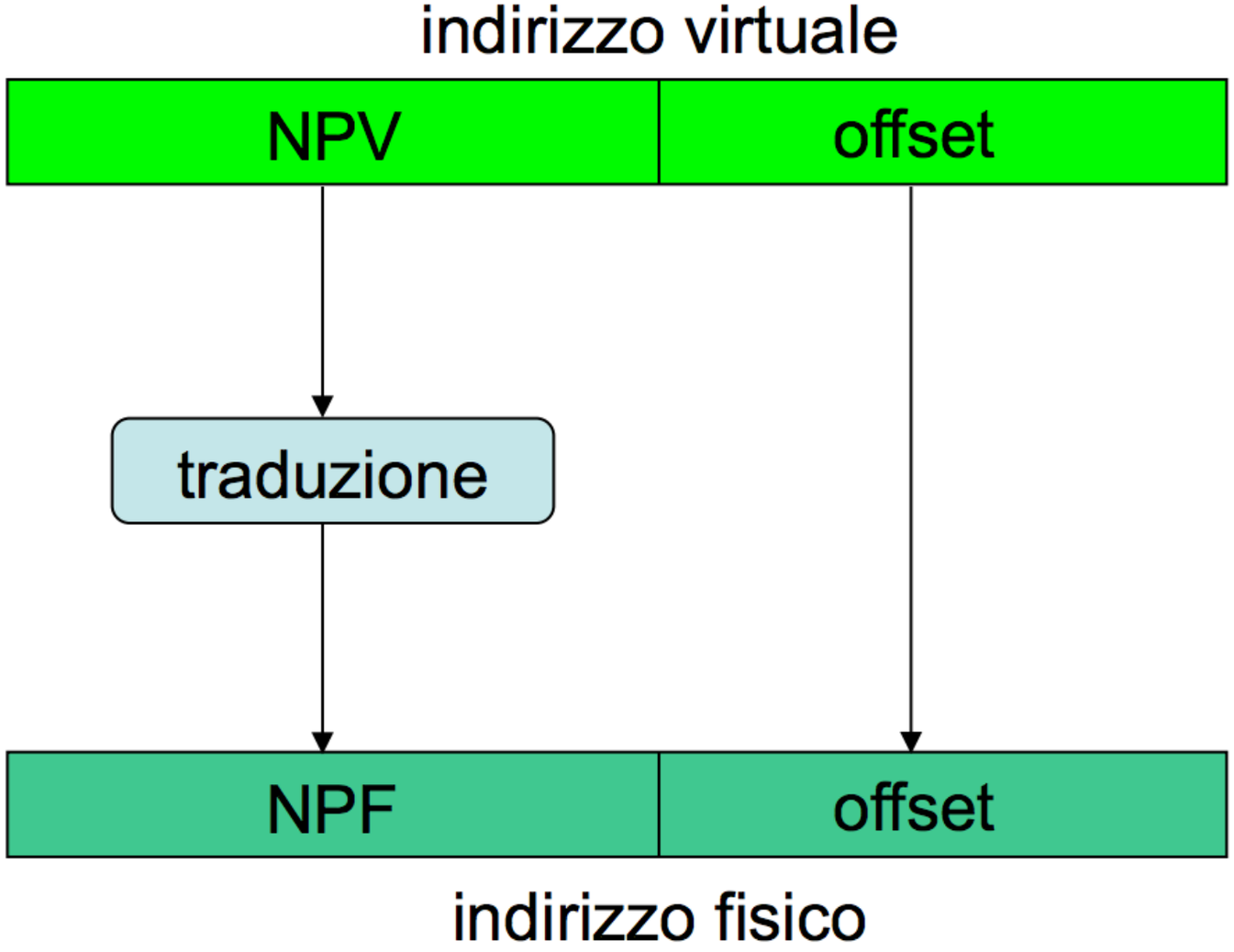


# Memoria fisica e memoria virtuale (ripasso)

- E' del tutto analoga: si hanno un numero di pagina fisica (NPF) e da uno spiazzamento (offset) all'interno della pagina



# Memoria fisica e memoria virtuale (ripasso)




le pagine virtuali e quelle fisiche hanno la stessa dimensione, quindi l'offset è lo stesso

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con hit rate di 0.8, hit time di 50 ns e miss penalty di 150 ns

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con hit rate di 0.9, hit time di 40 ns e miss penalty di 250 ns

Domande:

1) Quali dei due dispositivi ha il maggior numero di pagine di memoria fisica?

2) Si può dire quale dei due dispositivi ha maggiore memoria fisica? E quale dei due avrà maggiore memoria virtuale?

3) In quale dei due dispositivi l'accesso alla memoria è più rapido?

4) Si consideri ora il dispositivo più lento: qual è l'hit rate minimo che dovrebbe avere per essere rapido almeno quanto l'altro?

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con hit rate di 0.8, hit time di 50 ns e miss penalty di 150 ns

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con hit rate di 0.9, hit time di 40 ns e miss penalty di 250 ns

Domande:

*1) Quali dei due dispositivi ha il maggior numero di pagine di memoria fisica?*

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con hit rate di 0.8, hit time di 50 ns e miss penalty di 150 ns

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con hit rate di 0.9, hit time di 40 ns e miss penalty di 250 ns

Domande:

*1) Quali dei due dispositivi ha il maggior numero di pagine di memoria fisica?*

**Nessuno** dei due dispositivi:

A) 1 Mbyte memoria fisica => **indirizzo di 20 bit** (1 Mbyte = 2^**20** bytes)

Pagine di 4 Kbyte => **offset di 12 bit** (4 Kbyte = 2^**12** byte)

Bit di indirizzo disponibili => **20 - 12 = 8 bit** (2^**8** pagine = 256 pagine fisiche)

B) Indirizzo di **24 bit** => 2^**24** bytes = 16 Mbyte di memoria fisica

Pagine di 64 Kbyte => **offset di 16 bit** (64 Kbyte = 2^**16** byte)

Offset di 16 bit => **24 - 16 = 8 bit** (2^**8** pagine = 256 pagine fisiche)

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con hit rate di 0.8, hit time di 50 ns e miss penalty di 150 ns

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con hit rate di 0.9, hit time di 40 ns e miss penalty di 250 ns

Domande:

*2) Si può dire quale dei due dispositivi ha maggiore memoria fisica? E quale dei due avrà maggiore memoria virtuale?*

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con hit rate di 0.8, hit time di 50 ns e miss penalty di 150 ns

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con hit rate di 0.9, hit time di 40 ns e miss penalty di 250 ns

Domande:

*2) Si può dire quale dei due dispositivi ha maggiore memoria fisica? E quale dei due avrà maggiore memoria virtuale?*

A) 1 Mbyte memoria fisica => **indirizzo di 20 bit** (1 Mbyte = 2^**20** bytes)

Bit di indirizzo disponibili => **20 - 12 = 8 bit** (2^**8** pagine = 256 pagine fisiche)

B) Indirizzo di **24 bit** => 2^**24** bytes = 16 Mbyte di memoria fisica

Offset di 16 bit => **24 - 16 = 8 bit** (2^**8** pagine = 256 pagine fisiche)

***Non è possibile dire nulla della memoria virtuale: le informazioni fornite non ci permettono di risalire al numero di pagine virtuali ne di A ne di B***

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con **hit rate di 0.8**, hit time di **50 ns** e miss penalty di **150 ns**

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con **hit rate di 0.9**, hit time di **40 ns** e miss penalty di **250 ns**

Domande:

*3) In quale dei due dispositivi l'accesso alla memoria è più rapido?*

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con **hit rate di 0.8**, hit time di **50 ns** e miss penalty di **150 ns**

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con **hit rate di 0.9**, hit time di **40 ns** e miss penalty di **250 ns**

Domande:

*3) In quale dei due dispositivi l'accesso alla memoria è più rapido?*

***B è più veloce**:*

A. tempo medio di accesso alla memoria: **0.8 * 50 ns** + **(1-0.8) * 150 ns** = 70 ns

B. tempo medio di accesso alla memoria: **0.9 * 40 ns** + **(1-0.9) * 250 ns** = 61 ns

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con **hit rate di 0.8**, hit time di **50 ns** e miss penalty di **150 ns**

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con **hit rate di 0.9**, hit time di **40 ns** e miss penalty di **250 ns**

Domande:

*4) Si consideri ora il dispositivo più lento: qual è l'hit rate minimo che dovrebbe avere per essere rapido almeno quanto l'altro?*

# Memorie e tempi di accesso (TE 05/02/15)

Si considerino due dispositivi aventi le seguenti configurazioni:

**Configurazione A**

- 1 Mbyte di memoria fisica e pagine di memoria da 4 Kbyte
- una memoria cache con **hit rate di 0.8**, hit time di **50 ns** e miss penalty di **150 ns**

**Configurazione B**

- indirizzo di memoria fisica a 24 bit e pagine di memoria da 64 Kbyte
- una memoria cache con **hit rate di 0.9**, hit time di **40 ns** e miss penalty di **250 ns**

Domande:

*4) Si consideri ora il dispositivo più lento: qual è l'hit rate minimo che dovrebbe avere per essere rapido almeno quanto l'altro?*

A. tempo medio di accesso alla memoria: **0.8 * 50 ns** + **(1-0.8) * 150 ns** = 70 ns

*Posto:* ***hitRate = X****, dovrà valere:*

**X * 50 ns** + **(1- X) * 150 ns** <= 61ns

- 100 * **X** ns <= - 89 ns

**X >= 0.89** , quindi hit rate minimo = 0.89

# Iterazione vs. Ricorsione (Es.3 - TE 19/02/2015)

- Scrivere in MATLAB la funzione **ricorsiva** che, preso in ingresso un vettore P di K elementi interi ed un numero h, restituisca al chiamante il vettore *"ruotato a destra"* di h posizioni.
- Si scriva quindi una seconda funzione che risolve lo stesso problema in maniera **iterativa**.

Esempio:

**Input**: P = [1, 2, 3, 4, 5, 6] , h = 3

**Output**: P = [4, 5, 6, 1, 2, 3]

# Iterazione (Es.3 - TE 19/02/2015)

```
function out = g(P, h)
    for

    end
```

# Iterazione (Es.3 - TE 19/02/2015)

```
function out = g(P, h)
    for H=1:h

    end
```

# Iterazione (Es.3 - TE 19/02/2015)

```
function out = g(P, h)
        for H=1:h
            P = [P(end) P(1:end-1)];
        end
```

# Iterazione (Es.3 - TE 19/02/2015)

```
function out = g(P, h)
        for H=1:h
            P = [P(end) P(1:end-1)];
        end
        out = P;
end
```

# Iterazione (Es.3 - TE 19/02/2015)

```
function out = g(P, h)
        if (h<1)
            return
        end
        for H=1:h
            P = [P(end) P(1:end-1)];
        end
        out = P;
end
```

**H** = 0, **P** = [1, 2, 3, 4, 5, 6]

**H** = 1, **P** = [**6**, 1, 2, 3, 4, 5]

**H** = 2, **P** = [**5**, **6**, 1, 2, 3, 4]

**H** = 3, **P** = [**4**, **5**, **6**, 1, 2, 3]

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
```

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if


    else


end
```

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if


    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if (h==0)

    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if (h==0)
        out = P;
    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

```
rec( [1, 2, 3, 4, 5, 6], 3)
    ...
    out = rec( [6, 1, 2, 3, 4, 5], 2)
            ...
            out = rec( [5, 6, 1, 2, 3, 4], 1)
                    ...
                    out = rec( [4, 5, 6, 1, 2, 3], 0)
                            ...
                            out = [4, 5, 6, 1, 2, 3]
```

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if (h==0)
        out = P;
    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

```
rec( [1, 2, 3, 4, 5, 6], 3)
    …
    out = rec( [6, 1, 2, 3, 4, 5], 2)
            …
            out = rec( [5, 6, 1, 2, 3, 4], 1)
                    …
                    out = [4, 5, 6, 1, 2, 3]
                            …
                            out = [4, 5, 6, 1, 2, 3]
```

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if (h==0)
        out = P;
    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

rec( [1, 2, 3, 4, 5, 6], 3)
…
out = rec( [6, 1, 2, 3, 4, 5], 2)
…
out = **[4, 5, 6, 1, 2, 3]**
…
**out** = [4, 5, 6, 1, 2, 3]
…
out = [4, 5, 6, 1, 2, 3]

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if (h==0)
        out = P;
    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

rec( [1, 2, 3, 4, 5, 6], 3)
…
out = **[4, 5, 6, 1, 2, 3]**
…
**out** = [4, 5, 6, 1, 2, 3]
…
out = [4, 5, 6, 1, 2, 3]
…
out = [4, 5, 6, 1, 2, 3]

# Ricorsione (Es.3 - TE 19/02/2015)

```
function out = rec(P, h)
    if (h==0)
        out = P;
    else
        out = rec([P(end) P(1:end-1)],h-1);
end
```

rec( [1, 2, 3, 4, 5, 6], 3) ⟶ **[4, 5, 6, 1, 2, 3]**

…

**out** = [4, 5, 6, 1, 2, 3]

…

out = [4, 5, 6, 1, 2, 3]

…

out = [4, 5, 6, 1, 2, 3]

…

out = [4, 5, 6, 1, 2, 3]

# Ordinamento: “Bubble Sort”

- Scrivere una funzione ***ordina*** che ordini un vettore di interi, una funzione ***confronta*** che confronti due numeri ed una funzione ***scambiaPosizioni*** che inverta le posizioni di due elementi di un vettore.

- La funzione ***confronta*** può confrontare i numeri come preferisce, fintanto che il valore restituito (0 o 1) rappresenti la *relazione di maggiore e minore* generalizzato (ossia la funzione sia un ordine parziale).

- La funzione ***ordina*** ordina il suo vettore di dati in ingresso *nel senso* della funzione ***confronta***, implementando l’algoritmo “Bubble Sort” che ***scambia*** le ***posizioni*** degli elementi adiacenti.

# Bubble Sort - Esempio

6 5 3 1 8 7 2 4

# Funzione *ordina*

```
function [v] = ordina(v)

        for i = 1:length(v)-1

            end
    end
end
```

# Funzione *ordina*

```
function [v] = ordina(v)

        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));
        end
    end
end
```

# Funzione *ordina*

```
function [v] = ordina(v)

        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));

                if (confronto)


                end

            end
        end
end
```

# Funzione *ordina*

```
function [v] = ordina(v)

        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));

            if (confronto)
                v = scambiaPosizioni(v, i);

            end

        end
    end
end
```

# Funzione *ordina*

```
function [v] = ordina(v)
    continua = 1;
    while continua


        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));

            if (confronto)
                v = scambiaPosizioni(v, i);

            end

        end
    end
end
```

# Funzione *ordina*

```
function [v] = ordina(v)
    continua = 1;
    while continua



        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));

            if (confronto)
                v = scambiaPosizioni(v, i);
                continua = 1
            end


        end
    end
end
```

# Funzione *ordina*

```
function [v] = ordina(v)
    continua = 1;
    while continua
        continua = 0;

        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));
            if (confronto)
                v = scambiaPosizioni(v, i);
                continua = 1
            end

        end
    end
end
```

6 5 3 1 8 7 2 4

# Funzione *scambiaPosizioni*

```
function [ v ] = scambiaPosizioni( v, pos )

end
```

# Funzione *scambiaPosizioni*

```
function [ v ] = scambiaPosizioni( v, pos )
    a = v(pos);

end
```

# Funzione *scambiaPosizioni*

```
function [ v ] = scambiaPosizioni( v, pos )
    a = v(pos);
    v(pos+1) = a;
end
```

# Funzione *scambiaPosizioni*

```
function [ v ] = scambiaPosizioni( v, pos )
    a = v(pos);
    b = v(pos+1);
    v(pos+1) = a;
end
```

# Funzione *scambiaPosizioni*

```
function [ v ] = scambiaPosizioni( v, pos )
    a = v(pos);
    b = v(pos+1);
    v(pos) = b;
    v(pos+1) = a;
end
```

# Funzione *confronta*

```
function [response] = confronta(a,b)



end
```

# Funzione *confronta*

```
function [response] = confronta(a,b)
    % torna vero se a è MINORE di b,

    response = a < b ;
end
```

# Funzione *confronta*

```
function [response] = confronta(a,b)
    % torna vero se a è MINORE di b,
    % provocando ordinamento DECRESCENTE
    response = a < b ;
end
```

# Funzione *confronta*

```
function [response] = confronta(a,b)
    % torna vero se a è MINORE di b,
    % provocando ordinamento DECRESCENTE
    response = a < b ;
end
```

```
function [response] = confronta(a,b)
    % torna vero se a è MAGGIORE di B
    % provocando ordinamento CRESCENTE
    response = a > b ;
end
```

# Funzione *ordina*

```
function [v] = ordina(v)
    continua = 1;
    while continua
        continua = 0;

        for i = 1:length(v)-1
            confronto = confronta(v(i), v(i+1));
            if (confronto)
                v = scambiaPosizioni(v, i);
                continua = 1
            end

        end
    end
end
```

6 5 3 1 8 7 2 4

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

- In matematica, la **congettura di Goldbach** è uno dei più vecchi problemi irrisolti nella teoria dei numeri. Essa afferma che *ogni numero pari maggiore di 2 può essere scritto come somma di due numeri primi* (che possono essere anche uguali).

- Si scriva quindi una funzione in MATLAB (chiamata ***goldbach***) che, dato un numero pari, stampa a video i due numeri primi che sommati lo compongono. Nel realizzare la funzione goldbach, si consiglia di **utilizzare una seconda funzione**, chiamata ***primo***, che dato un numero x in ingresso dica se tale numero sia primo o meno. L'interfaccia della funzione primo è la seguente:

```
function [ris] = primo(x)
```

- Nota: questa esercizio richiede quindi la creazione di UNA SOLA funzione: goldbach

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end
```

```
    if ((x+y) == n)
        display([num2str(x) ' + '  num2str(y)])
        return
    end
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end
```

```
                if (primo(y))
                    if ((x+y) == n)
                        display([num2str(x) ' + '  num2str(y)])
                        return
                    end
                end
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end



        if (primo(x))
            if (primo(y))
                if ((x+y) == n)
                    display([num2str(x) ' + '  num2str(y)])
                    return
                end
            end

    end
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end

    for x=2:n
        if (primo(x))
            if (primo(y))
                if ((x+y) == n)
                    display([num2str(x) ' + '  num2str(y)])
                    return
                end
            end

    end
end
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end

    for x=2:n
        if (primo(x))
            for y=2:x
                if (primo(y))
                    if ((x+y) == n)
                        display([num2str(x) ' + '  num2str(y)])
                        return
                    end
                end
            end
        end
    end
end
```

# Congettura di Goldbach (Es.1 - TE 19/02/2015)

```
function [x,y] = goldbach(n)

    if (mod(n,2) ~= 0 || n<=2)
        return
    end

    for x=2:n
        if (primo(x))
            for y=2:x
                if (primo(y))
                    if ((x+y) == n)
                        display([num2str(x) ' + '  num2str(y)])
                        return
                    end
                end
            end
        end
    end

    display 'congettura errata *_*'
```

# Numero primo, in tutte le salse

- Dato un numero intero positivo inserito dall'utente, dire se tale numero è primo (stampa a video 1 se primo, 0 altrimenti).
- Un numero è primo se è divisibile solo per 1 e se stesso.

Esempio:

- 7 —> 1
- 9 —> 0
- 9871 —> ?

# Numero primo, soluzione *iterativa*

```
function [ris] = numeroPrimoIterativa(x)
```

# Numero primo, soluzione *iterativa*

```
function [ris] = numeroPrimoIterativa(x)
for y=2:sqrt(x)

end
```

# Numero primo, soluzione *iterativa*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)

end
```

un numero è **primo**
fino a **prova contraria!**

# Numero primo, soluzione *iterativa*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
end
```

un numero è **primo**
fino a **prova contraria!**

basta che sia **0** ad una iterazione qualsiasi
per aver trovato la **“prova contraria”**

# Numero primo, soluzione *ricorsiva*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
    end
```

```
function [ris] = numeroPrimoRicorsiva(x,y)
```

# Numero primo, soluzione *ricorsiva*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
    end
```

```
function [ris] = numeroPrimoRicorsiva(x,y)

        ris = (~(~rem(x,y)))*numeroPrimoRicorsiva(x,y+1);
```

# Numero primo, soluzione *ricorsiva*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
    end
```

```
function [ris] = numeroPrimoRicorsiva(x,y)
    ris = 1;
    ris = (~(~rem(x,y)))*numeroPrimoRicorsiva(x,y+1);
```

# Numero primo, soluzione *ricorsiva*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
    end
```

```
function [ris] = numeroPrimoRicorsiva(x,y)
    if y>sqrt(x)
        ris = 1;

        ris = (~(~rem(x,y)))*numeroPrimoRicorsiva(x,y+1);
```

# Numero primo, soluzione *ricorsiva*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
    end
```

```
function [ris] = numeroPrimoRicorsiva(x,y)
    if y>sqrt(x)
        ris = 1;
    else
        ris = (~(~rem(x,y)))*numeroPrimoRicorsiva(x,y+1);
    end
```

# Numero primo, soluzione *ricorsiva*

```
function [ris] = numeroPrimoIterativa(x)
    ris=1;
    for y=2:sqrt(x)
        ris = ris*(~(~rem(x,y)));
    end
```

```
function [ris] = numeroPrimoRicorsiva(x,y)
    if y>sqrt(x)
        ris = 1;
    else
        ris = (~(~rem(x,y)))*numeroPrimoRicorsiva(x,y+1);
    end
```

# Numero primo, soluzione *vettoriale*

```
function [ris] = numeroPrimoVettoriale(x)
```

# Numero primo, soluzione *vettoriale*

```
function [ris] = numeroPrimoVettoriale(x)
    divisori = 2:sqrt(x);
```

# Numero primo, soluzione *vettoriale*

```
function [ris] = numeroPrimoVettoriale(x)
    divisori = 2:sqrt(x);
    resti = rem(x,divisori);
    maschera = (resti == 0);
```

# Numero primo, soluzione *vettoriale*

```
function [ris] = numeroPrimoVettoriale(x)
    divisori = 2:sqrt(x);
    resti = rem(x,divisori);
    maschera = (resti == 0);
    ris = ~any(maschera);
```

# Numero primo, soluzione *vettoriale* (compatta)

```
function [ris] = numeroPrimoVettoriale(x)
    divisori = 2:sqrt(x);
    resti = rem(x,divisori);
    maschera = (resti == 0);
    ris = ~any(maschera);
```

**Equivalente a**

```
function [ris] = numeroPrimoVettoriale(x)
    ris = ~any(rem(x,2:sqrt(x)) == 0);
```

# Esempio di invocazione

```
numeroDaControllare = 9871;
numeroPrimoIterativa(numeroDaControllare)
numeroPrimoRicorsiva(numeroDaControllare,2)
numeroPrimoVettoriale(numeroDaControllare)
```

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

- Rappresentiamo un **polinomio** con un **vettore** contenente i suoi **coefficienti**, dal termine di grado massimo a quello di grado minimo.

*Si noti che un polinomio di grado n corrisponde a un vettore di lunghezza n+1*

*Esempio: **3**x^4 + **5**x^2 + **2**x + **7** (grado 4) corrisponde al vettore [**3 0 5 2 7**] (lunghezza 5)*

- Scrivere una funzione ricorsiva di nome **derivata** che:
  1. Riceva in ingresso un vettore che rappresenta un polinomio e un valore n
  2. Restituisca un vettore che rappresenta la derivata n-esima del polinomio

- Per calcolare la **derivata prima** del polinomio applicare la comune regola di derivazione per i polinomi. **Caso particolare**
- Calcolare la **derivata n-esima** come la **derivata prima della derivata (n-1)-esima** **Caso generale**

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
function[der] = derivata (pol, n)
```

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima

end
```

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima
    der = derivata(derivata(pol, n-1), 1)

end
```

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

    % Vettore con gli esponenti dei singoli monomi
    % Es. per 2x^3 + x^2 + 3 --> esp = [3 2 1]
    esp = [length(pol)-1 : -1 : 1]

else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima
    der = derivata(derivata(pol, n-1), 1)

end
```

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

    % Vettore con gli esponenti dei singoli monomi
    % Es. per 2x^3 + x^2 + 3 --> esp = [3 2 1]
    esp = [length(pol)-1 : -1 : 1]

    % Calcolo della derivata prima:
    % Es. per il polinomio precedente:
    % [3 2 1].*[2 1 0] = [6 2 0]
    der = esp.*pol(1:length(pol)-1)
```

**Caso particolare**

```
else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima
    der = derivata(derivata(pol, n-1), 1)
```

**Caso generale**

```
end
```

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

**5**x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

*corrisponde al vettore* [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = derivata(**derivata(pol, 2)**, 1)

**derivata(pol, 2)**

der = derivata(**derivata(pol, 1)**, 1)

**derivata(pol, 1)**

esp = [5 4 3 2 1]

der = **[25 16 9 4 1]**

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

**5**x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

*corrisponde al vettore* [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = derivata(**derivata(pol, 2)**, 1)

**derivata(pol, 2)**

**der = derivata([25 16 9 4 1], 1)**

derivata([25 16 9 4 1], 1)

esp = [4 3 2 1]

der = **[100 48 18 4]**

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

**5**x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

*corrisponde al vettore* [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = derivata(**derivata(pol, 2)**, 1)

**derivata(pol, 2)**

der = **[100 48 18 4]**

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

**5**x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

*corrisponde al vettore* [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = **derivata([100 48 18 4], 1)**

**derivata([100 48 18 4], 1)**

esp = [3 2 1]

der = **[300 96 18]**

# Esercizio: calcolo derivata (ricorsiva)

RIPASSO

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

**5**x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

*corrisponde al vettore* [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = **[300 96 18]**

# Esercizio 4: Metodo di Eratostene

Scrivere un programma che stampa i numeri primi minori di 100.
(Crivello di Eratostene)

***Metodo di Eratostene***

1. Si scrivono tutti i numeri naturali a partire da 2 fino ad N in un elenco detto “setaccio”.
2. Poi si cancellano (setacciano) tutti i multipli del primo numero del setaccio (escluso lui stesso).
3. Si prosegue così fino ad arrivare in fondo.
4. I numeri che restano sono i numeri primi minori od uguali a n.

# Esercizio 4: Metodo di Eratostene

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;
```

```c
    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Si procede al setaccio: ignoro i primi due elementi (numeri 0 e 1)
    for (i=2;i<100;i++) {
```

```c
    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Si procede al setaccio: ignoro i primi due elementi (numeri 0 e 1)
    for (i=2;i<100;i++) {
        // Se il numero è ancora marcato come primo (non è multiplo dei suoi precedenti)
        if(array[i]==1) {
```

```c
    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Si procede al setaccio: ignoro i primi due elementi (numeri 0 e 1)
    for (i=2;i<100;i++) {
        // Se il numero è ancora marcato come primo (non è multiplo dei suoi precedenti)
        if(array[i]==1) {
            // Marco come "non primi" tutti i suoi multipli
```

```c
    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Si procede al setaccio: ignoro i primi due elementi (numeri 0 e 1)
    for (i=2;i<100;i++) {
        // Se il numero è ancora marcato come primo (non è multiplo dei suoi precedenti)
        if(array[i]==1) {
            // Marco come "non primi" tutti i suoi multipli
            for (j=2; j<=(100/i); j++){
                array[i*j] = 0;
            }
        }
    }

    // Si stampano i primi 100 numeri primi
    printf("\nNumeri primi < 100: ");

    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Si procede al setaccio: ignoro i primi due elementi (numeri 0 e 1)
    for (i=2;i<100;i++) {
        // Se il numero è ancora marcato come primo (non è multiplo dei suoi precedenti)
        if(array[i]==1) {
            // Marco come "non primi" tutti i suoi multipli
            for (j=2; j<=(100/i); j++){
                array[i*j] = 0;
            }
        }
    }

    // Si stampano i primi 100 numeri primi
    printf("\nNumeri primi < 100: ");

    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Se il numero è ancora marcato come primo (non è multiplo dei suoi precedenti)
    if(array[i]==1) {
        // Marco come "non primi" tutti i suoi multipli
        for (j=2; j<=(100/i); j++){
            array[i*j] = 0;
        }
    }

    printf( "\nNumeri primi < 100: " );

    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

# Esercizio 4: Metodo di Eratostene

```c
#include <stdio.h>
int main()
{
    int i,j;

    // Utilizzo un array di 100 elementi
    // Se il contenuto nella cella i-esima è 1, il numero è primo
    int array[100];

    // Inizializzazione dell'array: "tutti i numeri sono primi"
    for (i=0;i<100;i++)
        array[i]=1;

    // Se il numero è ancora marcato come primo (non è multiplo dei suoi precedenti)
    if(array[i]==1) {
        // Marco come "non primi" tutti i suoi multipli
        for (j=2; j<=(100/i); j++){
            array[i*j] = 0;
        }
    }

    printf( "\nNumeri primi < 100: " );

    for (i=1;i<100;i++){
        if(array[i]==1)
            printf("\t%d", i);
    }
    return 0;
}
```

meglio:

j<=(100/i) && j*i<100

# Metodo di Eratostene - In MATLAB?

# Metodo di Eratostene - In MATLAB?

```
rimasti = 2:100;
setacciati = [];
```

# Metodo di Eratostene - In MATLAB?

```
rimasti = 2:100;
setacciati = [];

while ~isemplty(rimasti)

end
```

# Metodo di Eratostene - In MATLAB?

```
rimasti = 2:100;
setacciati = [];

while ~isemplty(rimasti)
    candidato = rimasti(1);
    setacciati = [setacciati rimasti(1)];

end
```

# Metodo di Eratostene - In MATLAB?

```
rimasti = 2:100;
setacciati = [];

while ~isemplty(rimasti)
    candidato = rimasti(1);
    setacciati = [setacciati rimasti(1)];
                              mod(rimasti,candidato)));
end
```

# Metodo di Eratostene - In MATLAB?

```
rimasti = 2:100;
setacciati = [];

while ~isemplty(rimasti)
    candidato = rimasti(1);
    setacciati = [setacciati rimasti(1)];
    rimasti = rimasti(logical(mod(rimasti,candidato)));
end
```

# Esercizio: Vettori e maschere

```
% chiedere all'utente di inserire un vettore e un numero
%
% calcolare:
% # il numero di elementi del vettore uguali al numero insierito
% # il numero di elementi del vettore maggiori del numero insierito
% # il numero di elementi del vettore minori del numero insierito
%
% indicare poi il valore di tali elementi,
% la loro posizione del vettore
% il vettore binario per ogni operazione richiesta
```

# Esercizio: Vettori e maschere

```
%x = input('inserire un numero')
%vettore = input('inserisci un vettore')
```

# Esercizio: Vettori e maschere

```
%x = input('inserire un numero')
%vettore = input('inserisci un vettore')

x = 10
vettore = [10 20 30 50 60]

%vettore binario
Buguali=
Bmaggiori
Bminori=
```

# Esercizio: Vettori e maschere

```
%x = input('inserire un numero')
%vettore = input('inserisci un vettore')

x = 10
vettore = [10 20 30 50 60]

%vettore binario
Buguali= vettore==x
Bmaggiori= vettore>x
Bminori= vettore<x

%valore degli elementi
Vuguali=
Vmaggiori
Vminori=
```

# Esercizio: Vettori e maschere

```
%x = input('inserire un numero')
%vettore = input('inserisci un vettore')

x = 10
vettore = [10 20 30 50 60]

%vettore binario
 Buguali= vettore==x
 Bmaggiori= vettore>x
 Bminori= vettore<x

 %valore degli elementi
Vuguali= vettore(vettore==x)
Vmaggiori= vettore(vettore>x)
Vminori= vettore(vettore<x)
```

# Esercizio: Vettori e maschere

```
%numero di elementi
```

# Esercizio: Vettori e maschere

```
%numero di elementi
Nuguali=size( vettore (vettore==x),2)
Nmaggiori=size( vettore (vettore>x),2)
Nminori=size( vettore (vettore<x),2)

%alternativa:
% Nmaggiori = sum(vettore > x)

%posizione degli elementi
Puguali=
Pmaggiori
Pminori=
```

# Esercizio: Vettori e maschere

```
%numero di elementi
Nuguali=size( vettore (vettore==x),2)
Nmaggiori=size( vettore (vettore>x),2)
Nminori=size( vettore (vettore<x),2)

%alternativa:
% Nmaggiori = sum(vettore > x)

%posizione degli elementi
Puguali= find(vettore==x)
Pmaggiori= find(vettore>x)
Pminori= find(vettore<x)
```

*Grazie*