

Intro alla

Programmazione

Davy

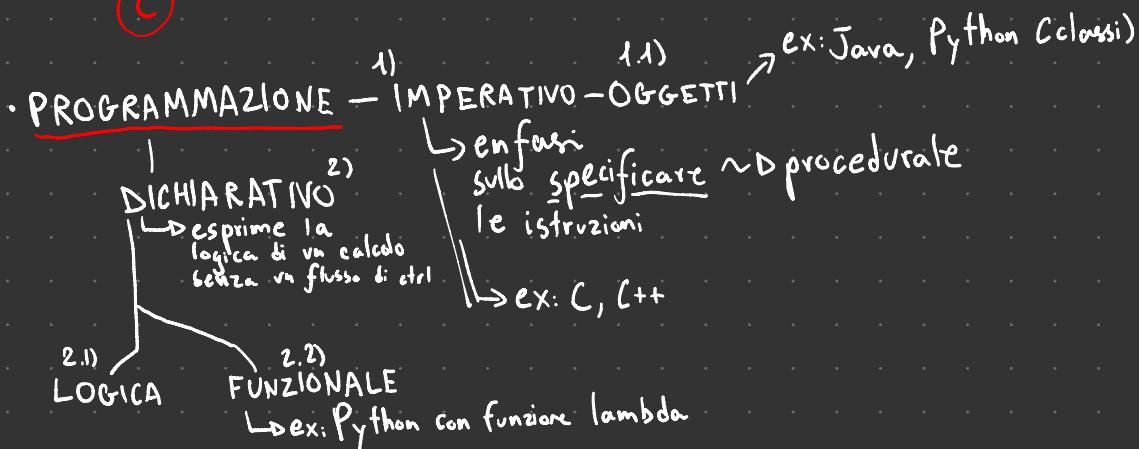


# CASA DEI PUFFI borgo san Sergio >> DA EVITARE!!

28.09

- 7 paradigmi di programmazione:
  - macroaree in cui si sviluppa un linguaggio

concetto  
del calcolo  
C



Ex: ascensore

IMPERATIVO	DICHARATIVO
- ATTESA	- ARRIVATO e APERTO, ENTRA
- APRI	- SE PUOI ENTRARE,
- CHIUDI	DEVI ASPETTARE ARRIVI
- BOTONE	- ...
-	-

CD impone delle istruzioni CD impone una struttura LOGICA → paradigma 2.1)

! domani porto PC!

- Programma: Una descrizione eseguibile → esiste qualcosa che da un CALCOLATORE di un metodo (algoritmo) per il calcolo di un risultato (output) a partire da un input → COMPLESSITÀ
- Moti programmi → devono coesistere condividono la RAM (uno spazio di memoria) → memoria virtuale

↑ trascrivere del DNA  
analoga DNA-proteine

$$f(x) = y \rightarrow \text{ex: } f(x) = \log(x) + 1$$

DEF • Variabile: Un nome associato ad  
un valore modificabile

DEF • Stato: Un insieme di variabili che  
qualcosa che mi interessa



Esempio:  $f(x, y) = \log(x) + (y - 1)$



$$\ell + (y - 1) \quad (\ell + e = f(x, y))$$

↳ ha definito una variabile  $\ell$       ↳ operazione di ASSEGNAZIONE

La sintassi per l'assegnamento

NOME = EXPR ;      obbligatorio

definisce la programmazione imperativa

↳ tutte quelle combinazioni di simboli che mettono var.

hanno una catena di simboli che mettono assiemi operatori, costanti, ...

(ex:  $\pi$ ,  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $($ ,  $)$ ,  $V$ ,

FUN, ...)

↳ precedenza & precendenza di operatori, costanti, ...

↳ albero di sintesi astratta

x espressioni ambigue

Esempio:  $\cdot (3+4)-5 ; \frac{3+4-5}{2}$

$\cdot (3+4)3 ; \frac{3+4}{2}3$

$\cdot (3+4)2 ; \frac{3+4}{2}2$

↳ sarebbero operazioni ambigue

- Con 30000€ voglio coprire il costo dei miei spostamenti in auto in un anno

↓  
voglio formalizzare un ragionamento x questo

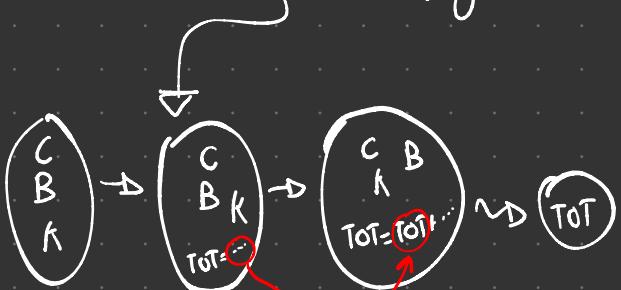
(D) 1. Ragiona sulle variabili; faccio delle ipotesi

I. Suppongo  $\begin{cases} C = 20.000 \text{€} (\text{ho una panda}) \\ B = 0.2 \text{€/km} (\text{benzina}) \\ K = 10.000 \text{km/A} \end{cases}$  ho fatto 3 assegnamenti

input  $\boxed{\text{TOT}}$   $B * K = 2000 \text{€/A}$  tot =  $b * k;$

III.  $C + \dots = 22.000 \text{€}$  tot = tot + c;

CD Si RAGIONA sullo STATO del programma



### Algoritmo di moltiplicazione del contadino russo

- Voglio moltiplicare  $146 \times 37 = 5402$

1.	<u>146</u>	<u>37</u>
div x2	146/2	73
cappox (per bifatto)	<u>73</u> /2	<u>36</u>
finché ho 1	... 18	236
	... 9	592
	... 4	1184
	... 2	2368
	1	4736

2. Elimino righe con numeri pari a dx

III.  
Somma rimanti:  $5402 = 74 + 592 + 1376$

- Lo trasformo in un programma

•  $f(m, n) = mn$

•  $p = 0$ ; p è prodotto

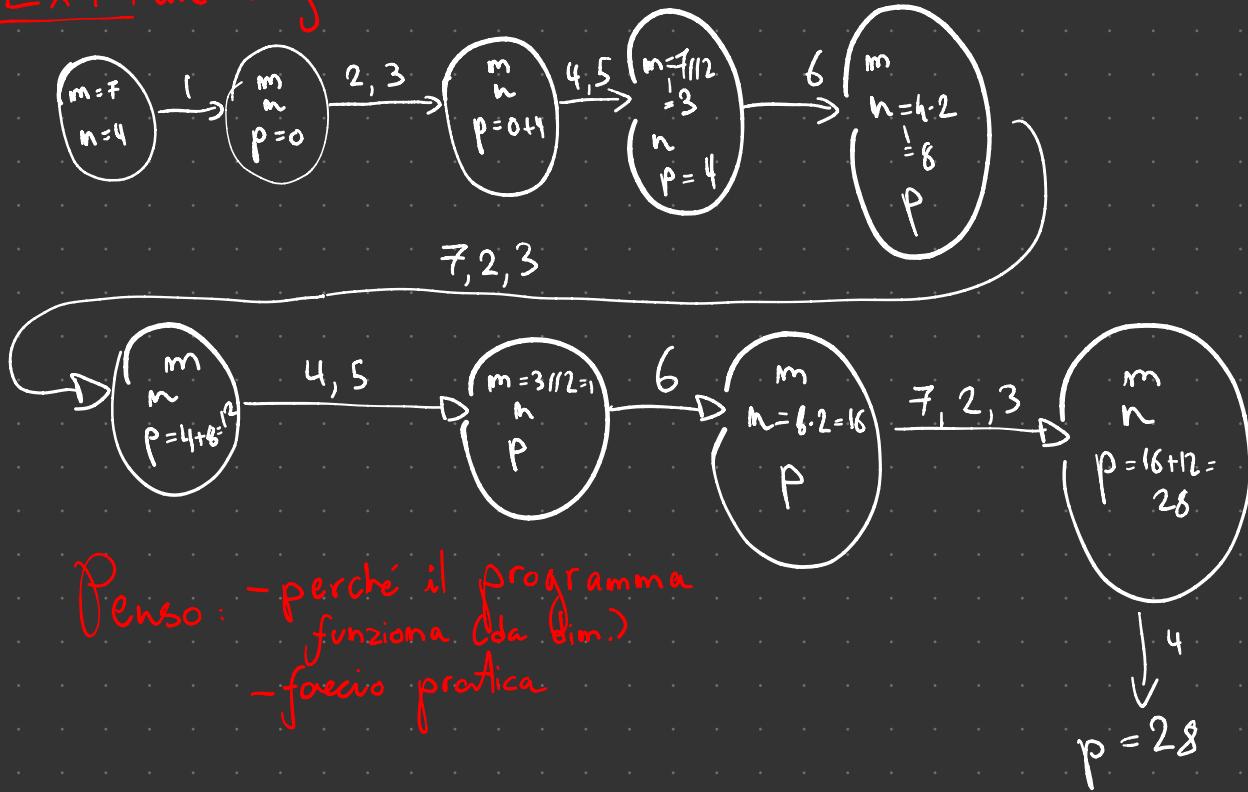


## Pseudocodice

- 1) CREA P
- 2) SE  $m$  È PARI, SALTA A ⑤
- 3) ASSEGNA  $p = p + n$
- 4) SE  $m = 1$ , FINE
- 5) ASSEGNA  $m = m // 2$
- 6) ASSEGNA  $n = n * 2$
- 7) SALTA A ②

→ sposta il flusso di esecuzione

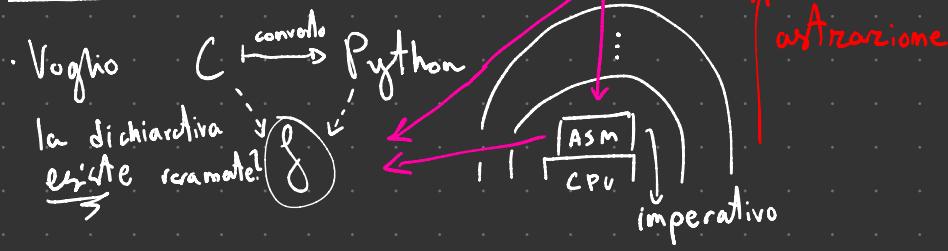
Ex : Fare diagramma da



Penso : - perché il programma funziona (da dim.)  
 - faccio pratica

$$p = 28$$

04.10:



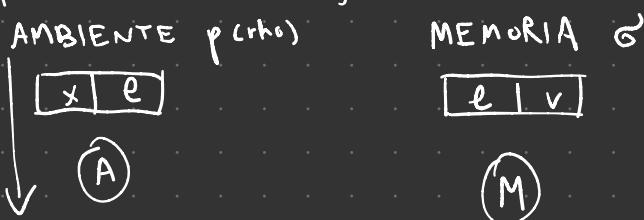
Computazione come ??

- NON è vero; non è così che si rapp. lo stato.
- ambiente memoria] diversi ma collegati;

Se  $\Gamma$  (insieme delle variabili)

- L'ambiente: associa ad ogni var. una locazione (indirizzo)
- La memoria: associa ad ogni locazione un valore

Quindi lo stato è fatto così:



è come una funzione con d

$$\rho : \Gamma \rightarrow \underline{\mathcal{Y}}$$

insieme delle locazioni

$$\delta : \underline{\mathcal{Y}} \rightarrow \underline{\text{VAL}}$$

insieme dei valori

Questo permette di avere molti valori

y	l <sub>1</sub>
x	l <sub>0</sub>

 $\delta$ 

l <sub>1</sub>	w
l <sub>0</sub>	v

Quindi in uno stato ci sono 2 scatolotti;  
 per fare di più si dovrà complicare di più il programma

La dichiarazione di variabili  
è l'op. per cui si crea un scatolotto

SINTASSI      ① tipo namej x v → è SOLO una dichiarazione, senza valore.  
                ② tipo name = valorej

Serre

Serre  
 DECODIFICARE il tipo del valore j  $\Rightarrow$  algebra:  $\mathbb{N}$ ,  $\mathbb{Z}$ , sono di tipo diverso  
 il num binario quali sono?

- int  $\rightarrow \mathbb{Z}$   
- double  $\rightarrow$  com lar virgulas

$\downarrow$   
D: precision

ex: int x; x e c\|

int y = 7;    

x	l.
---	----

l.	7
----	---

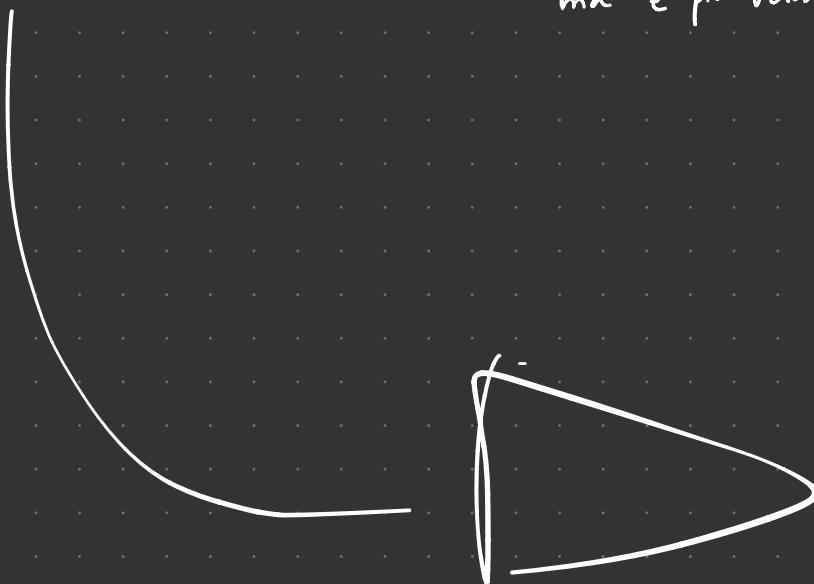
se var. possono  
essere cambiati dinamicamente

In alcuni linguaggi non serve specificare il tipo; essi sono non-tipati (es: Python)

ma questi hanno un cattivo: se ho  $a+b$

1) Se è non- tipato, il PC deve controllare a runtime quali sono i tipi

2) C'è fortemente tipato, si dev'essere PRECISI  
ma è più veloce



## L'esempio della macchina co' moto?)

flutto

int k = 10'000; (km/A)

double b = 0.2; ( $\epsilon/\text{km}$ )

int c = 20'000; (km)

↓ (double) tot; → qual'è tip? → double perciò abbiamo  
int + double; ci serve il tip più generale/preciso

tot = b \* k;

tot = tot + c;

$\downarrow$   
caso  
specifico (senza virgola)  
di double

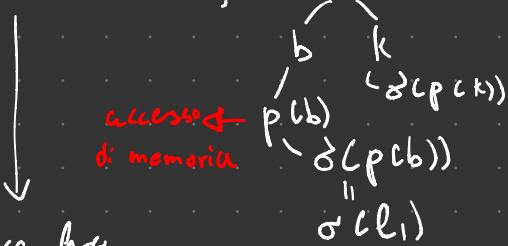
## Il disegno della memoria

1. Si parte dall'alto verso il basso (metto le var.)

P	o
K l <sub>0</sub>	l <sub>0</sub> 10'000
b l <sub>1</sub>	l <sub>1</sub> 0.2
c l <sub>2</sub>	l <sub>2</sub> 20'000
tot l <sub>3</sub>	l <sub>3</sub> —

l'ordine conta<sup>2</sup>, finora NO  
(al massimo te la si cambiano)

2. Ora  $b * k$ ; Il PC guarda nell'ambiente  
per trovare b; fa quindi p(b)



Ora b è

risult. entrambi  $\rightarrow b * k = 20'000 \Rightarrow$  dove lo metto? in  $l_3$   
i membri

perciò  $tot = b * k$ ;  
faccio  $\delta(p(tot))$

P	o
K l <sub>0</sub>	l <sub>0</sub> 10'000
b l <sub>1</sub>	l <sub>1</sub> 0.2
c l <sub>2</sub>	l <sub>2</sub> 20'000
tot l <sub>3</sub>	l <sub>3</sub> 2'000

$\rightarrow$  gli assegnamenti  
si fanno solo in memoria

3.  $tot = tot + c$ ;

$$\text{faccio } \delta(p(tot)) + \delta(p(c)) = 2000 + 20'000 = 22'000$$

P	o
K l <sub>0</sub>	l <sub>0</sub> 10'000
b l <sub>1</sub>	l <sub>1</sub> 0.2
c l <sub>2</sub>	l <sub>2</sub> 20'000
tot l <sub>3</sub>	l <sub>3</sub> 22'000

lo metto su  
 $p(tot)$

• Come mai serre tutto questo?  
 ↳ locazione doppia

• int  $x;$   
 int  $y = x + 2;$

Disegno lo stato

①	P	D
②	$\begin{array}{ c c } \hline x & l_0 \\ \hline y & l_1 \\ \hline \end{array}$	$\begin{array}{ c c } \hline l_0 & 1 \\ \hline l_1 & \delta(p(w)) + 2 \\ \hline \end{array}$

→ ???

↓  
 non c'è  $l_1 + 2$ ? più avan 1, 5, ...

↓  
 è SBAGLIATO, non eseguibile

• Si può usare le expr, MA devono essere prima definite  $\Rightarrow$  quindi l'ordine conta

Altro prog.

```

1 int x;
2 int y;
3 int z, w;
4 x = 1;
5 y = 3;
6 z = x + y;
7 w = 2 * z + (z - 1);

```

Disegno

1, 2, 3, 4, 5:

p

x	l <sub>0</sub>
y	l <sub>1</sub>
z	l <sub>2</sub>
w	l <sub>3</sub>

o

l <sub>0</sub>	1
l <sub>1</sub>	3
l <sub>2</sub>	/
l <sub>3</sub>	/

6,

z = x + y;

↓

x	l <sub>0</sub>
y	l <sub>1</sub>
z	l <sub>2</sub>
w	l <sub>3</sub>

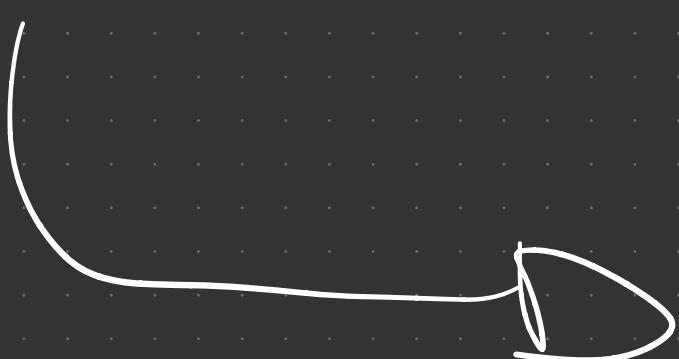
$$\delta(p(l_0)) + \delta(p(l_1)) = 1 + 3 = 4$$

→ o

l <sub>0</sub>	1
l <sub>1</sub>	3
l <sub>2</sub>	4
l <sub>3</sub>	11

$$\begin{aligned}
 & \begin{array}{c} + \\ \diagdown \quad \diagup \\ * \quad z \\ \diagdown \quad \diagup \\ z \quad z-1 \end{array} \\
 & w = 2 * z + (z - 1) \\
 & = 2 * 4 + 3 = 11
 \end{aligned}$$

$$\begin{aligned}
 & (2 * (\delta(p(z))) + \\
 & (\delta(p(z)) - 1)
 \end{aligned}$$



30 ho:

```
int x = ... (10);  
int y = ... (25);  
int max;
```

Come si trova max, f.c. si ha valore massimo?  
↳ fin'ora è IMPOSSIBILE

Davo introduzione i c.d. costrutti condizionali:

if  $(x > y)$  condizione logica  $\rightarrow$  se è VERA, il prog. esegue su ciò che c'è  
 ramo if  $\rightarrow \max = x;$  sul ramo if; senno' else  
 else  
 ramo else  $\rightarrow \max = y;$  stato del programma  $\rightarrow$  ca  
 ct  
 sa  
 me

if, else sono flussi di controllo;

Il programma prende una "decisione" → modificare la sequenza del programma

→ if > -D ; o va da una parte e dall'altra

Il flusso condizionale può essere ottenuto in un senso, senza else



int x = ...;

$$\text{int } y = \dots;$$

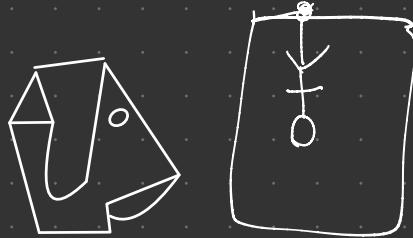
```
int max;
```

$\max = y$ ;  $\rightarrow$  ipotizzo al  $y$  è più grande

if  $(x > y) \rightarrow M^A$  se sbaglio, cambio idea  
 $\max \in X$

if  $(x > y) \rightarrow$  MA se sbagliò, cambio idea  
 $\max = x$ .

05.10



- float  $\rightarrow$  numero con virgola (floating point)
- char  $\rightarrow$  carattere

forma generica costrutto condizionale

if (cond.)

{ X

}

else

{

}

Le graffe sono IMPORTANISSIME per C!

$\downarrow$   
posso non metterle  
se ho solo una riga

$\downarrow$   
Su Python

è inline; indentazioni

({) if ... ]  $\rightarrow$  le graffe sono  
come le  
tabulazioni  
()}

if (A)      in realtà  
                c'è A e B / A  $\wedge$  B

  if (B)      }  
  {      X

    in quanto eseguito solo  
    se entrambe vere

    ↑      }  
    if (A  $\wedge$  B)  
    {

    AND      "C commerciale"      AND  
                OR      A  $\vee$  B      !A  
                     ↑      ↑      ↑

A	B	A $\wedge$ B	A $\vee$ B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

else      ]  $\rightarrow$  quando viene  
eseguito?      }

ex:  $\text{if}(x > 3)$

Voglio  $x \geq 3$ , come?  $>=$

analoga per  $x \leq 3 \Leftrightarrow x <= 3$

Voglio  $x \stackrel{?}{=} 3 \Leftrightarrow ==$ ; **NON** =  $\rightarrow$  l'operazione di assegnamento

ci dà un valore di verità;

$x = 3 \rightsquigarrow \textcircled{1}$  (se l'operazione  
è andata a buon fine)

Quindi non si va mai al ramo  
else

• Posso cogliere <sup>cond.</sup> la **composite**, come

$\text{if}((x-3) \stackrel{?}{=} (y+1)) \rightarrow$   
 $\downarrow$   
 $\stackrel{?}{=}$

Se  $x = 3, y = -1$  allora è falsa

Se  $x$  è dichiarata MA non serializzata,  
allora abbiamo errore

Ex: Io voglio:  $x, y$ ; vedere se sono  
concorde - discordi

int x = ...;

int y = ...;

$\downarrow$   $\left[ \begin{array}{l} \text{if } (x > 0) \& (y > 0) \\ \text{or } (x < 0) \& (y < 0) \end{array} \right]$

però fa schifo, farci quindi:

int concordi\_positivi = ( $x > 0$ ) && ( $y > 0$ );

int concordi\_negativi = ( $x < 0$ ) && ( $y < 0$ );

$\text{if}(\text{concordi\_positivi} \text{ or } \text{concordi\_negativi})$

{

...

}

ancora più corta?

int z = x \* y;  $\rightarrow$  modo più ingenuo  $\Rightarrow$  sfrutto le proprietà  
algebriche del segno

$\text{if } (z > 0)$

{

...

}

Se ho disponibile  
a sono &

$\rightarrow$  questo c'

MOLTO  
meglio e  
chiaro

$\Downarrow$

da fare

## Il conte alla rovescia

int  $x = 3;$

$x = x - 1;$  ②

$x = x - 1;$  ①

$x = x - 1;$  ③

BOOM

non è  
generale  
↑

- problema  $\rightarrow$  devo ripetere  $\text{int } x = x - 1;$   $x$  volte  $\Rightarrow$  sintatticamente è già codificato SOLAMENTE per lavorare con 3
- voglio rendere programma arbitrario  $x$  volte

int  $x = 16 \text{ h } 2;$

if ( $x \neq 0$ )

{

$x = x - 1;$

}

$\rightarrow$  voglio tornare in sv  $\Rightarrow$  c'è? sì, col continue

russo

c'è possibile farlo grazie al `while()`

↓  
condiz.  $\rightarrow$  come col costrutto condizionale if

while (condizione)

{

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

  |

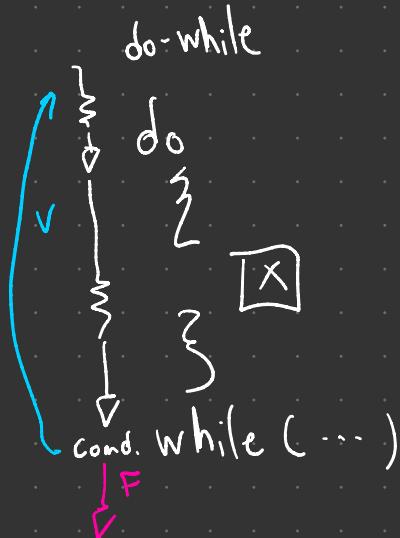
  |

  |

  |

int  $x = \dots;$   
 while ( $x \neq 0$ )  
 { if ( $x > 0$ )  $x = x - 1;$   
   else  $x = x + 1;$   
 }

proprietà dell'algebra?  
 ↳ "commutativa" → while e if  
 ↓  
 non sempre



Esempio:

$$n! = \begin{cases} 1 & \text{se } n=0 \\ n(n-1)! & \text{altrimenti} \end{cases}; \quad 5! = 5 \cdot 4! = 5 \cdot 4 \cdot 3! = \dots = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

fattoriale

↓  
 def. simile al while;  
 - si applica le moltiplicazioni finché la condizion ...

↓

è RICORSIVA  $\xleftrightarrow[a]{\text{oppone}}$  iterativa

programma

1 int  $n = \dots;$   
 int  $f = 1;$

while ( $n \geq 0$ )

$\Rightarrow \delta(p(n)) \geq 1 \Rightarrow V$ ; faccio blocco

$f = f * n;$

$n = n - 1;$

}

memorien

p	d
$n$ $f$	$l_0$ $l_1$ $s$ $l_1$ $l_1$

$l_0$	$s$
$l_1$	$l_1$

$f$	$n$
(1)	(5)

↑

$f$	$l_0$
$l_1$	$l_1$

$l_0$	$l_1$
$l_1$	$l_1$

$l_1$	$l_1$
$l_1$	$l_1$

$\Rightarrow 20, 3; 60, 2; 120, 1; 120, 0 \Rightarrow$

$f$	$l_0$
$l_1$	$l_1$

$l_0$	$l_1$
$l_1$	$l_0$

• posso cambiare il programma?

sì, ma non molto rilassante ( $\rightarrow$  al posto di 3)

• Adesso vogliamo far questo:

•  $x, y > 0, x, y \in \mathbb{N}$

• sottrarre il più piccolo dal più grande

• continua fino a che  $x = y$  non sono uguali

• quando  $z = \underline{\text{mcd}}(x, y)$

massimo  
comune  
divisore

7/8 righe

int  $x, y; z \parallel \mathbb{N}^+$

while ( $x \neq y$ )

{

if ( $x > y$ ) {

$x = x - y$

}

if ( $y > x$ ) { / else {

$y = y - x$

}

}

{

int  $z = x;$

• quanti passi su  $n!$ ;  $n = n+1, \dots$

• q.v.  $\rightarrow$  lo puoi determinare a priori? NO (sicuramente dipende da  $x, y$ )

• l'iterazione per  $n!$  è

DETERMINATA  $\rightsquigarrow$  quindi ha senso col costrutto indeterminato.

iterazione  
 $\parallel$  INDETERMINATA

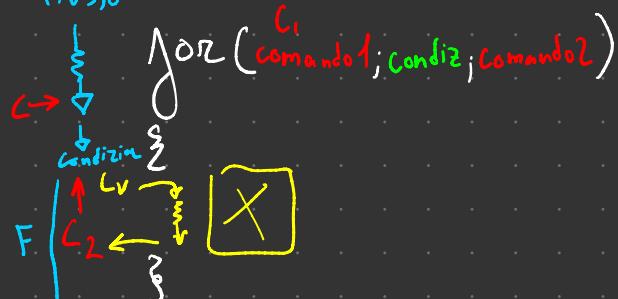
il while permette al programma di fare un numero di op. indeterminata

funziona  
ma è sbagliato  $\leftarrow$  NO.

ci serve  
un'iterazione determinata (for, range)

## Istruzione for

flusso



## Esempio

for (int i=0; i < 10; i++)

{

i = i+1;

}

idea: ho memoria e ambiente

↳ faccio dichiarazione i=0  $\boxed{i}$   $\boxed{e_0}$   $\boxed{e_0}$   $\boxed{0}$

↳ il blocco modifica  $\boxed{X}$   $\xrightarrow{P} \boxed{\square}$   $\xrightarrow{Q} \boxed{\square}$  ...

↳  $C_2$   $i++$ ;  $\boxed{i}$   $\boxed{e_0}$   $\boxed{e_1}$

↳ valuto  $i < 10$

↳ ripeto

↳ finché  $i=10$  (quindi 10 volte)

! la CONDIZIONE deve dipendere da  $C_1$

$C_2$  DEVE agire su  $C_1$  !  $\Rightarrow$  per avere senso

Ese:

$\sum_{i=1}^k \boxed{X} \Rightarrow \text{for (int } i=1; i \leq k; i++)$

$n!$  cfor)

int  $n = \dots;$

int  $f = 1;$

for ( $i=0; i < n; i++$ )

{

$f = f * (n - i);$  ← Qui NON si modifica  $i \Rightarrow$  il for non diratta un for

! all'ultimo si fa comunque

$C_2: \times$  la memoria deve  
cmq rappresentar



altri struttura: coda  $\xrightarrow{\text{inserzione dalla testa}}$  eliminazione dalla coda  $\rightarrow$  

caratt. di coda : FIRST IN  $\left.\begin{array}{l} \text{LAST IN} \\ \text{FIRST OUT} \end{array}\right\}$  (FIFO)

" di stack frame: LAST IN FIRST OUT (LIFO)

ex di frame

Se volessimo una variabile "immortale" calcolate)?  $\Rightarrow$  possibile ma è cattiva pratica (perdita di memoria)

ex:  $\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{int } x = 10; \\ \text{int } y = 5; \\ x = x + y; \end{array} \right\} \\ \left\{ \begin{array}{l} \text{int } z = 100; \\ z = x + y + z; \end{array} \right\} \end{array} \right\}$

① creo frame

p	$\emptyset$
x l <sub>0</sub>	l <sub>0</sub> 10
y l <sub>1</sub>	l <sub>1</sub> 5

②  $x = x + y;$

p	$\emptyset$
x l <sub>0</sub>	l <sub>0</sub> 15
y l <sub>1</sub>	l <sub>1</sub> 5

③ nuovo frame

p	$\emptyset$
x l <sub>0</sub>	l <sub>0</sub> 15
y l <sub>1</sub>	l <sub>1</sub> 5
<del>z l<sub>0</sub></del>	<del>l<sub>0</sub> 100</del>
<del>z l<sub>2</sub></del>	<del>l<sub>2</sub> 100</del>

$\delta(p(x)) \rightarrow$  cerco nel frame ma non lo trovo  
 ④  $z = x + y + z;$  in certe circostanze il pc lo cerca nel frame sottostante (padre)  
 ↓  
 p  $\emptyset$  frame padre  
 x l<sub>0</sub> l<sub>0</sub> 15  
 y l<sub>1</sub> l<sub>1</sub> 5  
 z l<sub>2</sub> l<sub>2</sub> 100 120  
 frame fifo

Non puoi modificare l'altra x

Q e se si chiamasse

"z", "x" o "y" var già dichiarata  
 $\hookrightarrow$  il pc lo trova già nel proprio stack

↑  
 l'area di competenza di questa var è questa  
 prendo la x più vicina del programma  
 Scoping statico

questo concetto di frame ci permette di avere più variabili con stesso nome ma identità diverse

# 11.10: Concetto di programma

## PROGRAMMA in C

#include <stdio.h> ~> utilizziamo funzioni (importando da una libreria)

int main(void) no dichiarazione di funzione

{ ... }

↓ chiamata di funzione

{ printf("Hello, World!"); }

return 0; → va messa alla fine del metodo main

→ un blocco di codice (il nostro programma)

→ in C l'esecuzione comincia dalla riga del blocco main.

stringa di caratteri

Dop. di stampa sullo schermo: una STRINGA

nome del file: main.c

→ cosa dobbiamo farci? → 1) tradurre in linguaggio macchina (.exe)

tramite il compilatore

(gcc)

↓

> comando 1.1) main.c -o my-file.out  
input in C File ottenuto  
Compiler (C) → (LBL)  
NON  
mettere  
l'input!  
linguaggio  
basso livello  
uso podi  
sempre

2) eseguo l'eseguibile

→ > ./my-file.out

2.1) > Hello, World!

2.2) > \_

- Ad ogni modifica sintattica nel file C NON corrisponde automaticamente una modifica nell'eseguibile .out

! il compilatore lo fa SOLE SE il programma è sintatticamente corretto !

↳ Se no dà dei messaggi  
↳ ! quindi garantisce  
SOLO la correttezza sintattica !

## Esempio:

```
int y = 10;
```

```
int x = 6;
```

```
printf("il valore di x è %.d", x);
```

```
printf("%.d %.d", x, y);
```

INT

printf per float

prende la  
variabile x e  
la mette dentro  
nella stringa

qui puoi metterci  
anche un'espressione tipo x+1

Voglio <sup>da</sup> base, altezza  $\rightarrow$  print base altezza area

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
int base = ...;
```

```
int altezza = ...;
```

```
int area = base * altezza;
```

```
printf("Se ho b=%.d, h=%.d; allora l'area è %.d", base, altezza,  
area);
```

```
}
```



Adesso: Vogliamo prendere valori dall'utente

- ↓ int base, altezza;
- scanf ("%d", &base); ? il PUNTATORE (lo vedremo) è molto importante  
l'utente aspetta l'input  
↓ dall'utente → consegna &cp(base) = ...  
il programma si FERMA
  - scanf ("%d", &altezza);
  - | int area = base \* altezza; ci sono l'utente non possono abilitare a priori calcoli  
| con float

Differenze: fino IV  
l'utente es. 9

# Funzioni

$f(x) = y \rightarrow$  calcola un valore

## SINTASSI

tipo\_valore\_ritorno namef(tipo\_arg nome1, ...)

ex:

int somma (int x, int y)  $\rightarrow f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

{  
...  
 $\rightarrow$  return val;  
}  
    └──  
        └── sicuramente  
            int

$\rightarrow$  blocco di codice  $\Rightarrow$  finisce con return

una funzione può non restituire nulla (o non avere argomenti)

Void ... (void)

{  
...  
}

Void mcd(int p, int s)

{

while (p != )

{

if (p > s) p = p - s;

else s = s - p;

{

memoria

}

frame  
dei  
parametri

x	l <sub>0</sub>
y	l <sub>1</sub>

l <sub>0</sub>	s
l <sub>1</sub>	10

lego variabili ai valori degli argomenti

frame  
2

p	l <sub>2</sub>
s	l <sub>3</sub>

l <sub>2</sub>	10
l <sub>3</sub>	6

$\delta(p(x))$   
 $\delta(p(y))$

MAIN

int x=10;  
int y=6;

$mcd(x, y)$ ,  $\rightarrow$  chiamo la funzione

$\rightarrow$  il flusso di esecuzione  
salta al blocco della  
nuova funzione  
mcd

$x$	$e_0$
$y$	$e_1$

60	6
11	10

lego variabili ai valori degli argomenti

frame  
2  
frame 3

$r$	$l_2$
$s$	$l_3$

$L_2$	10
$L_3$	6

Novo bloco

mld

$$\Rightarrow \boxed{b_3 | h}$$

Cerco perciò di sfruttando il meccanismo di legatura delle variabili

White  $\rightsquigarrow$  cancella  
alla  
condizione  
falsa

~~poi eseguo, cancello~~  
valuto condizie di muro

cancello |  
alla condizione  
false

Dopo la fusione NON c'è nessuna variazione allo stato

Piú stack → piú profunditá

## Collegamento funzioni - Scoping Attivo;

i nomi degli argenti  
POSSONO essere

... (int x)  
{ } ...  
  x ...

passaggio dei parametri  
per valori

Voglio modificare mod con return;

↳ return  $s_j \circ$  return  $p_j \rightarrow$  é INDIFERENTE

Los il white esse quando

$p = s \rightarrow$  per raggiungere

return  $\Sigma$   
necessaria  
questa  
condizione

Modifica la memoria?  $\rightarrow$  No, ma...



MAIN

int  $x = 10;$

int  $y = 5;$

int  $z = \text{mcd}(x, y);$   $\rightarrow$  ora modifica  
la memoria

- il return può essere  
 $\downarrow$  messo in qualcun punto

ex: codici di errore  $\rightarrow$  in ( non c'è  
la gestione di  
eccellenze)

Es: funzione fattoriale

$$n! := \begin{cases} -1 & \text{se } n \leq -1 \\ 0 & \text{se } 0 \leq n \leq 1 \\ n! & \text{se } n > 1 \end{cases}$$

int fattoriale (int n)

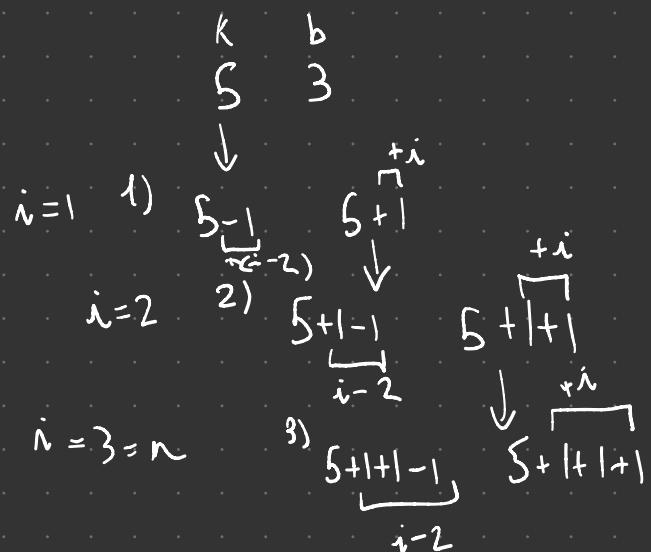
{  
if ( $n \leq -1$ ) return -1;  
if ( $n == 0 || n == 1$ ) return 1;  
else

{  
int  $p = 1;$

for (int  $i = 1; i \leq n; i++$ )

{  
 $p = p * i;$

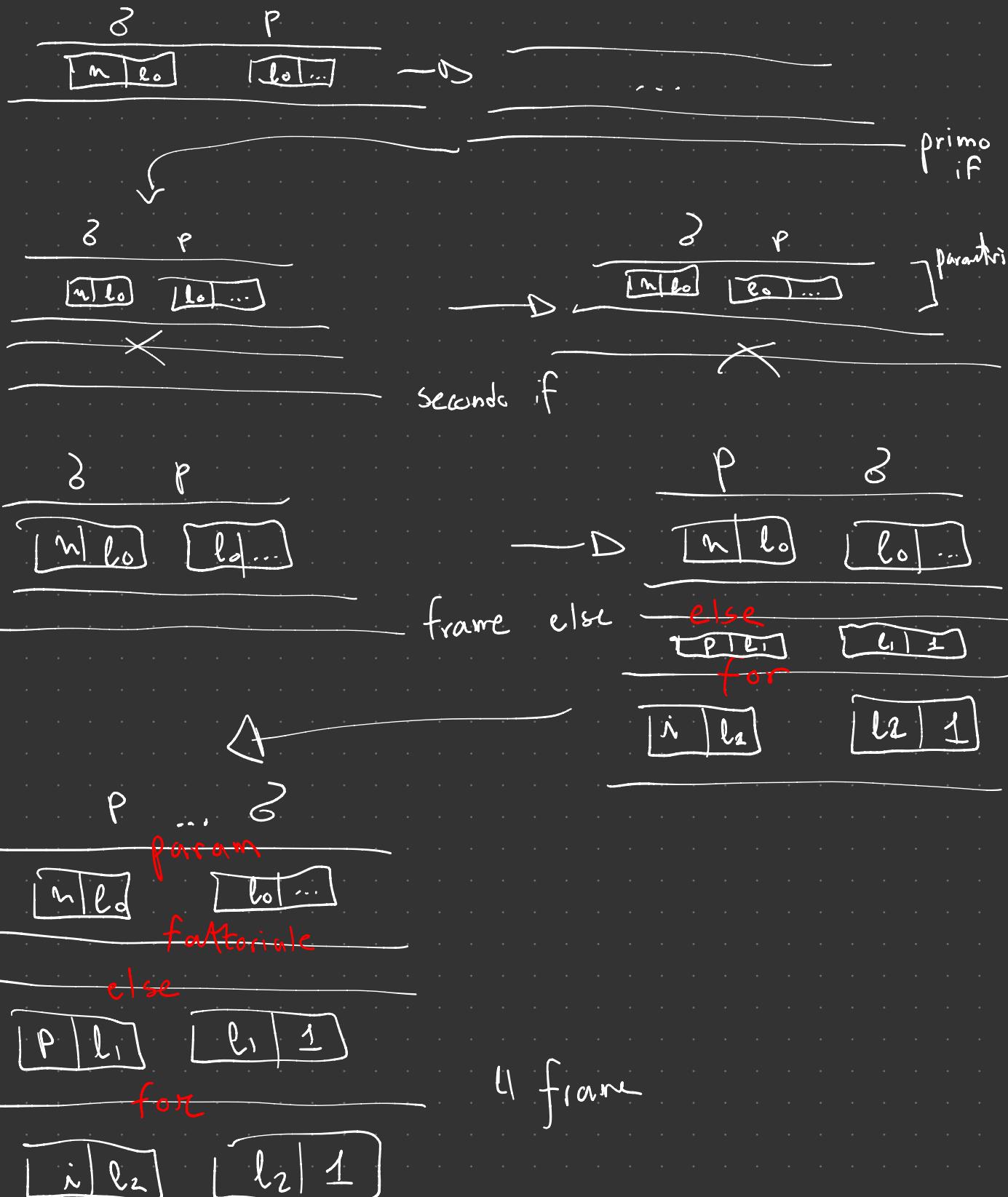
{  
return  $p;$



Disegno la  
memoria al  
massimo

# Memoria fattoriale

ex: tutti gli es.  $\Rightarrow$  faccio funzionare



# 18.10: Programmazione ricorsiva

$$\cdot n! = \begin{cases} 1 & \text{se } n=0 \\ n(n-1)! & \text{altrimenti} \end{cases}$$

per definire

↓ uso il fattoriale stesso

questa è  
una definizione per RICORSIONE

Significa

$n$  → punto da qui (il dominio)

insieme su cui ho una  
relazione d'ordine

ordinata

•  $n-1$  ← dato superiore (per chiamata)

•  $n-2$  ← R

•

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

⋮

• Mandelbrot.

$$\hookrightarrow z_0 = \dots$$

$$z_i = (z_{i-1})^2 + z_i$$

• Sintassi

`def func(x):`

`if (CASO BASE):`  $\rightarrow$  NON c'è  
   `return (EXP)`       $\text{func}(x)$

~~else:~~

$\downarrow$   `return (func(y<x) + ...)`

NON  
serve

i.e. se  
CASO B fosse vera,  
la funzione è  
già finita visto che ho  
un ritorno

$\hookrightarrow$  ho una chiamata `func(y)` con  $y < x$

$\hookrightarrow$  allora diventa  $g(f(y < x))$  ovvero  $f(y < x) + \dots$

Ricorsione in C per n!

```
int fatt(int n)
{
    if (n <= 0) { return 1; }
    return (n * fatt(n-1));
}
```



Esercizio: funzione g che calcola

$$g(n) = \sum_{i=1}^n i \rightarrow \begin{cases} 1 & se \ i=1 \\ i + g(i-1) & alt. \end{cases}$$

$$\begin{aligned}
 \hookrightarrow ex: g(5) &= 5 + g(4) \\
 &= 5 + 4 + g(3) \\
 &= 5 + 4 + 3 + g(2) \\
 &= 5 + 4 + 3 + 2 + g(1) \\
 &= 5 + 4 + 3 + 2 + 1
 \end{aligned}$$

## 2) RICORSIVA

```

int f2 ( int x )
{
    if ( x == 1 ) return 1;
    return ( x + f2 ( x - 1 ) );
}

```

algoritmo ricorsivo

## 1) ITERATIVA

```

int f ( int x )
{
    int s = 1;
    for ( int i = 0; i < x; i++ )
        s = s + i;
    return s;
}

```

algoritmo iterativo

$$\frac{h}{16} =$$

Es:

$$f(i, n) = \sum_{j=i}^n j * 2^j$$

int f ( int i, int n )

```

{
    if ( i > n ) return -1;
    int s = 0;

```

```

    for ( int j = i; j <= n; j++ )
        s = s + ( j * pow( 2, j ) );

```

```

    return s;
}

```

```

int f2 ( int i, int n )
{

```

```

    if ( i > n ) return -1;
    if ( i == n )

```

```

        return ( i * pow( 2, i ) );
    }

```

```

    return ( f2 ( i + 1, n ) + i * pow( 2, i ) );
}

```

Calcolo  $f(2, 4)$

$$\begin{aligned} \sum_{j=2}^4 j \cdot 2^j &\Rightarrow 2 \cdot 2^2 + 3 \cdot 2^3 + 4 \cdot 2^4 \\ &= 2 \cdot 4 + 3 \cdot 8 + 4 \cdot 16 \\ &= 8 + 24 + 4^3 \\ &= 8 + 24 + 64 = 8 + 88 \\ &= 96 \end{aligned}$$

Allora

$$\text{sia } \sum_{j=i}^n j \cdot 2^j$$

$$S_i = i \cdot 2^i$$

$$S_{n+1} = S_n + (n+1) \cdot 2^{(n+1)}$$

int pow ( int a, int b )

{

int p = 1;

for ( int i = 1; i <= b; i++ )

{

p = p \* a;

}

return p;

{  
 int s=0; potenzer  
 int p=2;  
 questa è corretta solo per i=1;  
 for (int j=i; j<=n; j++)  
 {  
 s=s+j\*p; } più "Furbo"  
 p=p\*2;  
 }  
 return s;  
 }  
 p =  $2^j$  con un ciclo  
 for

X domani faccio funz. rit.

## 19.10: Esercizi, roba sofisticata

↓

Si consideri un numero  $c_1 c_2 c_3 \dots c_n$  come 745.

Si definisce  $C_*$  come  $\sum_{i=1}^n c_i$

Implementazione ricorsiva di  $C_*$ .

$x \% y$  come il resto della divisione  $x$  per  $y$ .  $45 / 10 = 4$

$\hookrightarrow C_* = \begin{cases} \text{se } x / 10 == 0, \text{ allora return } x \% 10 & 745 / 10 = 7 \\ \text{altrimenti: } C_*(x / 10) + x \% 10 & 745 \% 100 = 45 \end{cases}$

int  $C_*(\text{int } x)$  {

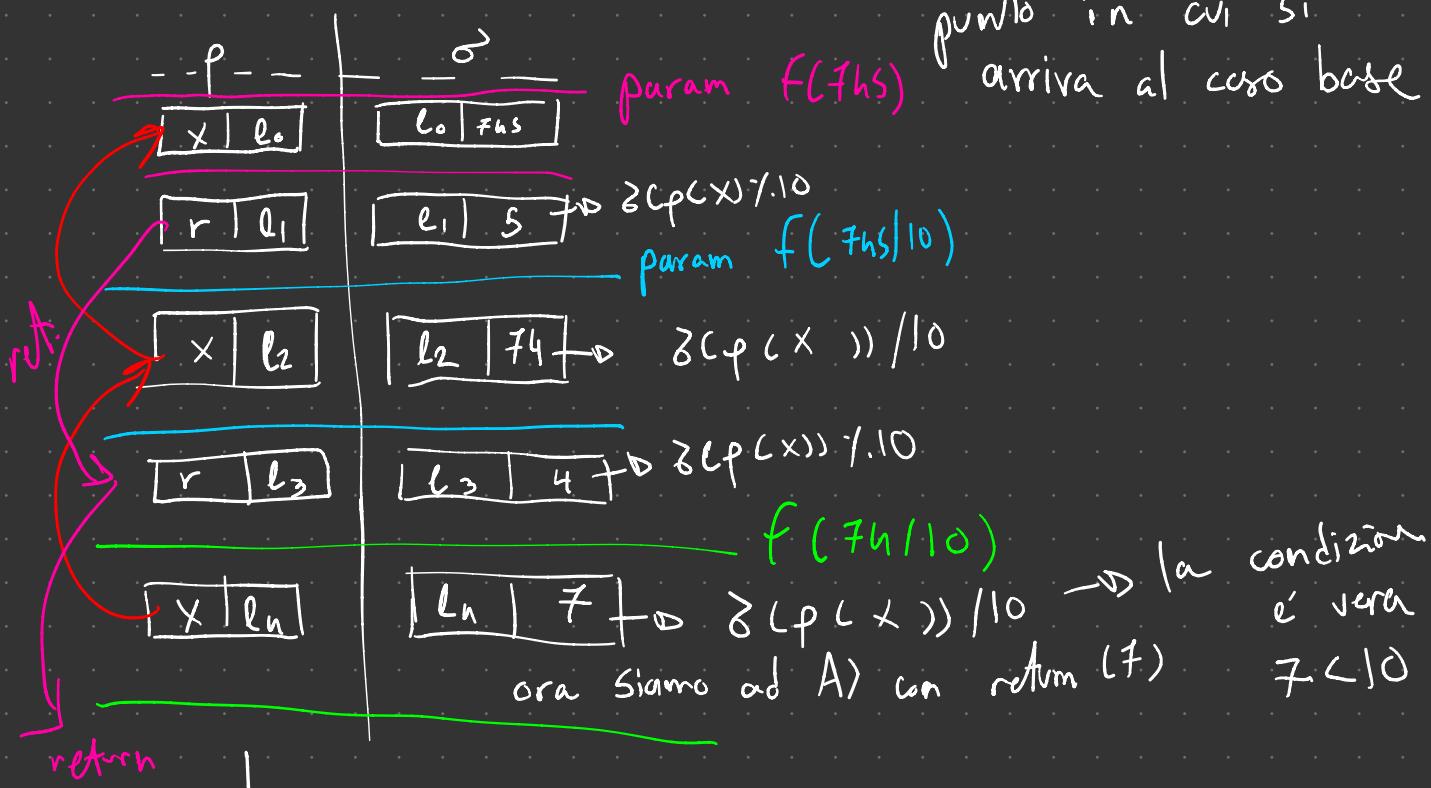
if ( $x / 10 == 0$ ) { return  $x \% 10$ ; }  
return  $(C_*(x / 10) + x \% 10)$ ;

}

$745 / 10 \rightarrow 74 \rightarrow 74 / 10 = 7 \rightarrow 7 / 10 = 0 \rightarrow F(N$   
(base)  
 $\downarrow$   
 $74 \% 10 \rightarrow 5 \quad \downarrow \quad \downarrow$   
 $74 \% 10 = 4 \quad 7 / 10 = 7$

## Esercizio 2

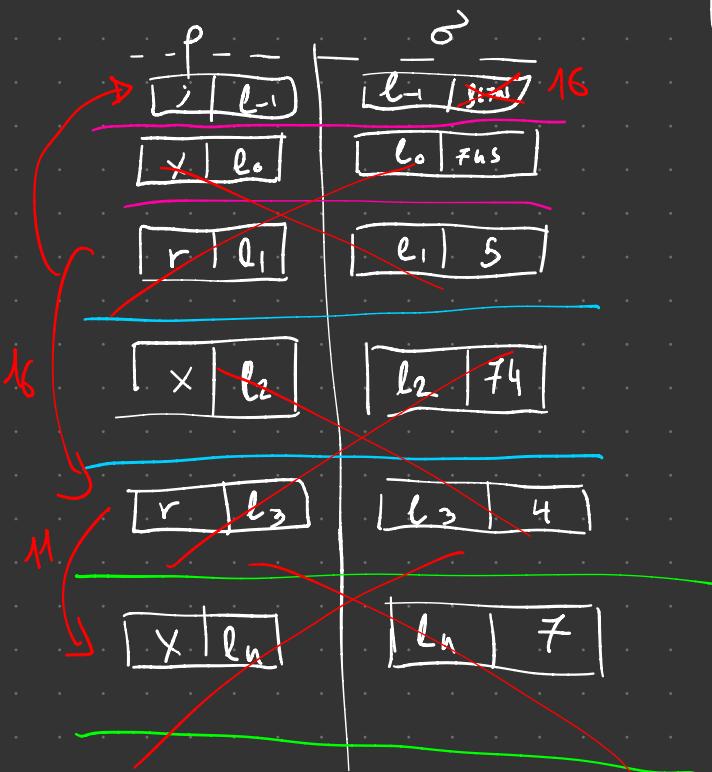
- Simulo l'esecuzione della memoria per  $C^*(fhs)$  al punto in cui si arriva al caso base



poi?  $\rightarrow$  return(7) disarrange il frame

$\hookrightarrow$  poi di funzione a cascata  
i.q. posso valutare tutti gli  
return

↓  
propaggo

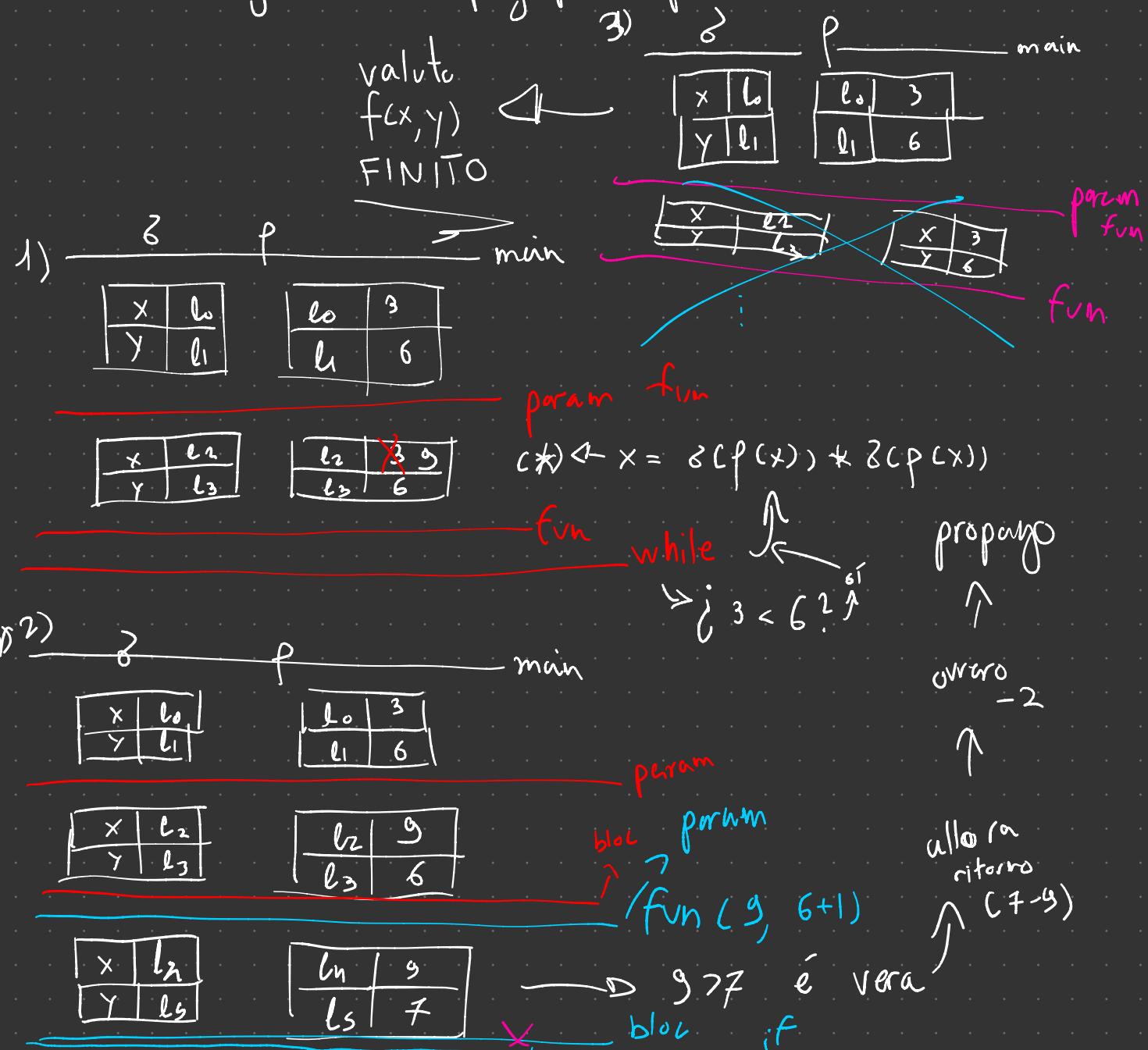


• Es: ho

```
int func(int x, int y)
{
    if (x > y) return (y - x);
    while (x < y) x = x * x;
    return func(x, y + 1);
}
```

```
int main(void)
{
    int x = 3;
    int y = 6;
    func(x, y);
}
```

Memoria ogni volta il prog. passa per \*



## 25.10: Teoria della complessità + strutture semantiche

$$\cdot F_n = \begin{cases} 0 & \text{se } n=0 \\ 1 & \text{se } n=1 \\ F_{n-1} + F_{n-2} & \text{alt} \end{cases}$$

int  $F_{\text{ric}}(\text{int } n)$

```
{ if (n==0)
    return 0;
if (n==1)
    return 1;
return Fric(n-1) + Fric(n-2); }
```

$$= F_{\text{ric}}(2) + F_{\text{ric}}(1)$$

$$= F_{\text{ric}}(1) + F_{\text{ric}}(0)$$

Voglio  $F_5 \rightarrow F_{\text{ric}}(5) = F_{\text{ric}}(4) + F_{\text{ric}}(3)$

$$= F_{\text{ric}}(3) + F_{\text{ric}}(2) = F_{\text{ric}}(1) + F_{\text{ric}}(0)$$



Cosa noto:

- dei termini compaiono molte volte

- $F_2$  ha un "costo" di chiamate  $\mapsto$  richiede spazio e tempo

→ qui direttamente una questione pratica  $\mapsto$  ho funzioni che prendono input  $\mapsto$  dimensione programma (nº di passi)

so soprattutto ho grandi numeri

Allora definisco la funzione di costo

$T(n)$  in funzione del programma  $F_{RIC}$

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 1 \end{aligned} \quad \left\{ \text{i casi } \rightarrow \text{ottengo } \underline{\text{subito}} \text{ un return} \right. \rightarrow \text{costanti}$$

$$T(3) = \underbrace{1}_{\text{costante}} + T(2) + T(1) = 1 + 1 + 1 = 3$$

$$T(h) = 1 + T(3) + T(2) = 1 + 3 + 1 = 5$$

$$T(6) = 1 + T(5) + T(4) = 1 + 9 + 5 = 15$$

:



Quanto costa?

Posso ragionare che  $\frac{3}{2}2^2 - 1 \quad (T_3, \frac{3}{2}2^2 - 1)$

$$\text{ma } T(h) > 2^2 - 1$$

$$\text{poi } T(5) > 2^3 - 1$$

$$\therefore T(6) > 2^3 - 1$$

troverei che

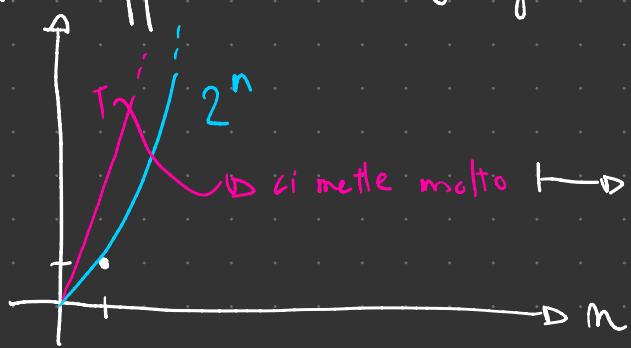
$$T(7) = 25 > 2^4 - 1$$

$$T(8) = 41 > 2^5 - 1$$

Posso dimostrare per induzione che

$$\boxed{T(n) \geq 2^{\lceil \frac{n}{2} \rceil} - 1} \quad (?) \quad \rightarrow \text{in realtà non è vero}$$

quindi rappre. il costo come esponenziale



ci mette molto  $\rightarrow$  infatti diventa impossibile

fare  $F(100)$  i.q.  
devo aprire almeno

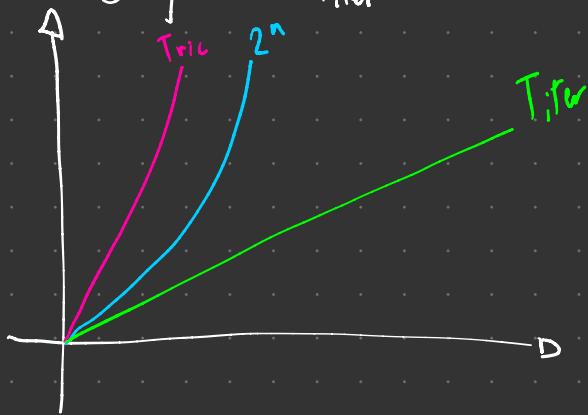
$2^{100} - 1$  frame

Conclusion

l'implementazione

fric è facile MA è ingestibile

Consideriamo la ver. iterativa  $\rightarrow$  uso i calcoli precedenti,  
 li ricordo  
 poi uso alla prox. iterazione



Faremo prog. di  
test. alle compagnie

$$T_{\text{fric}}, T_{\text{fitur}}$$

# ↑ Paradigma

### Ricorsivo

14

+ natural  
- fast-<sup>si</sup> ↑

# Semantica

---

# Operazionale

# Sulle FONDAMENTA di informatica → certe domande



Ex:  $h_0$

... CONDITIONE ...

---

$\underbrace{<P, \delta>, P}_{\text{input}} \rightarrow <P', \delta'>$

- relazioni ricorsive  
per schematizzare le f. ric.

$\downarrow <\rho, \delta>, \underline{x = EXP} \rightarrow$  valuto un ASSEGNAMENTO determino il valore  
 definisco  $\rho' = \rho \left[ \frac{x}{\ell^*} \right]$   $\hookrightarrow$  allora prendo EXP  $\rightsquigarrow V$   
 $\downarrow$   $\delta = \delta \left[ \frac{\ell^*}{V} \right]$  ex:  $EXP = 3 + h \rightsquigarrow 7$   
 $\downarrow$   $<\rho', \delta'>$  "metto uno scatolotto sulla testa"

5) se/else?

Altro ex:  $\delta[x/y](z) := \begin{cases} y & \text{se } x = z \\ \delta(z) & \text{alt.} \end{cases}$

chiamo  
algoritmo  
di ricerca  
nella memoria  
↓  
memoria  
modificata

- Altra regola  $\xrightarrow{\text{2 comandi}} \xrightarrow{\text{la COMPOSIZIONE delle due memorie}}$   
 $\underbrace{\langle p, \delta \rangle, C_1, C_2 \longrightarrow \langle p'', \delta'' \rangle}_{\langle p, \delta' \rangle, C_1 \rightarrow \langle p', \delta' \rangle; \langle p', \delta' \rangle, C_2 \rightarrow \langle p'', \delta'' \rangle}$
- Condizionale  
 $\langle p, \delta \rangle, \text{if EXPR then } C_1 \text{ else } C_2 \longrightarrow \langle p', \delta' \rangle$ 

regola  $\text{EXPR} \rightarrow \text{VERO} \quad \langle p, \delta \rangle, C_1 \rightarrow \langle p', \delta' \rangle$

reg. inversa  $\text{EXPR} \rightarrow \text{FALSO} \quad \langle p, \delta \rangle, C_2 \rightarrow \langle p', \delta' \rangle$

↓  
qui mi interessa  $C_2$

la valutazione  
di EXPR è molto  
importante
- Iterazione

$\langle p, \delta \rangle, \text{while EXPR do } C \xrightarrow{\text{risultato}} \langle p', \delta' \rangle \text{ while EXPR do } C$

↓  
no 2 com

$\text{EXPR} \rightsquigarrow \text{FALSO}$   
 $\longrightarrow \langle p, \delta \rangle \text{ (non modif.)}$

$\text{EXPR} \rightsquigarrow \text{VERO}$

$\langle p, \delta \rangle, C \rightarrow \langle p', \delta' \rangle \text{ while EXPR do } C$

(D ... → da qui parte la RICORRENZA)

• sì, parte dal nulla e si  
va ad aumentare lo stato con assegnazioni

! questa regola è  
↓ RICORSIVA ! → def.  
infatti il caso base  
per ricorrenze

è  $\text{EXPR} \rightsquigarrow \text{FALSO}$   
la ricorsione è fatto sullo  
spazio della memoria



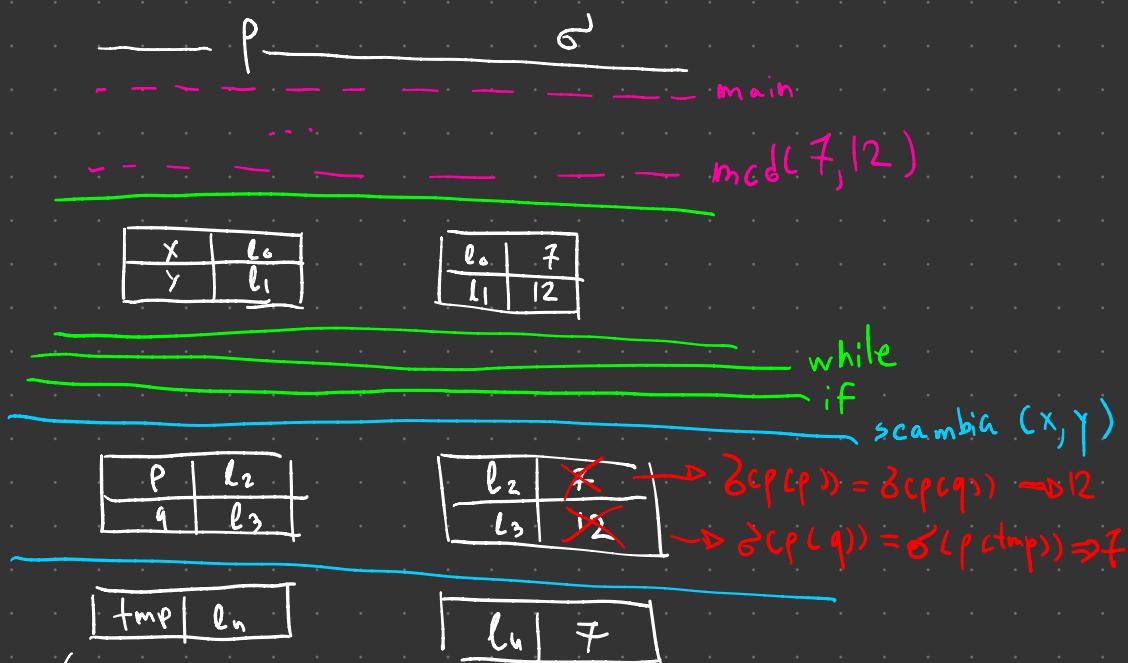
26.10.2023: 5 puntatori

- Abbiamo:

```
void scambia(Cint p, int q) { int mcd(int x, int y)
{ int tmp=p;
  p=q;
  q=tmp;
}
}

while(x!=y)
{ if (x>y) scambia(x,y);
  x=x-y;
}
return x;
```

Lo stiamo:

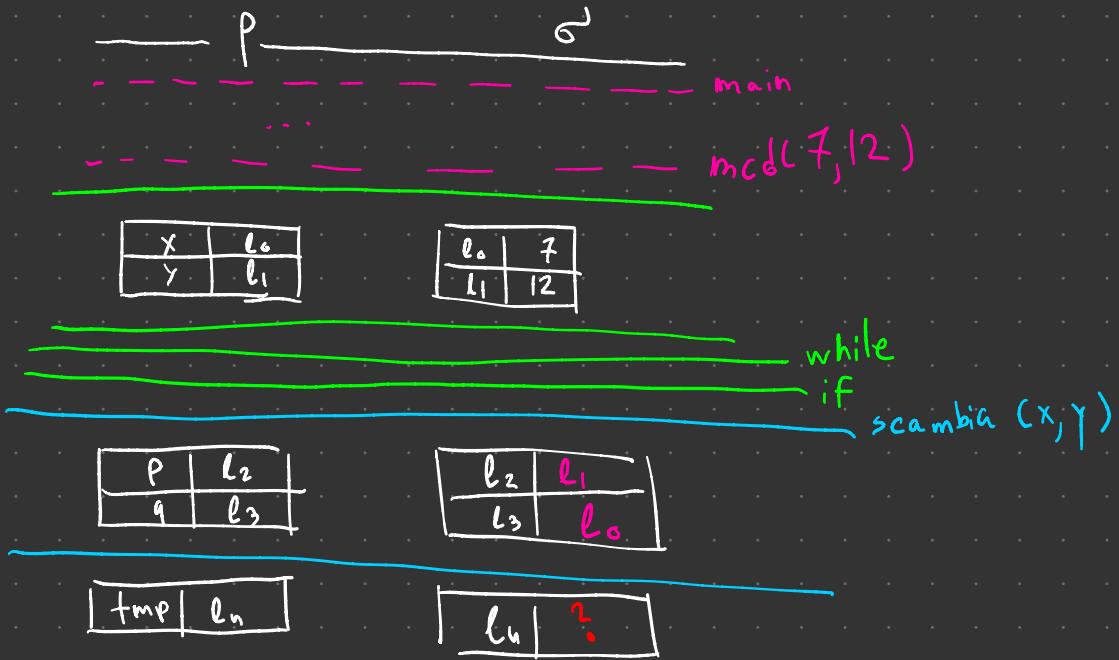


→ poi salta tutti i frame di `scambia(x, y)` → ma cmq. rimane  
che non ho scambiato  $x, y$

Qual è il problema?

così  $x, y$  e  $p, q$  si comportano indipendentemente  
tra di loro ⇒ quindi vogliamo che  $p, q$  fossero  
in qualche modo  $l_0, l_1$ . ↗ come?  
Voglio variabili i cui valori sono INDIRIZZI delle variabili

Ovvvero



Quindi ho dalla GdM

$\sigma: \mathcal{L} \rightarrow \bigvee \mathcal{U} \mathcal{L}$  → basta? NO. Devo poter all'indirizzo "puntato"

Allora ho il puntatore

↓  
Void scambia( int \*p, int \*q )  
..

Ex: int \* nome;  
double \* ... j

1) Devo espliarne? 2) Cambia il modo in cui chiamo funzione? O def?

Oss:  $p \Leftarrow \delta(p(p)) \rightarrow$  dc il valore di  $p \Leftarrow$  NON va bene, solo un indirizzo

\* $p \Leftarrow \delta(\delta(p(p))) \rightarrow$  OK

va a

"dereferenziazione" → voglio il **VALORE** puntato

int tmp = \*p;  
\*p = \*q;  
\*q = tmp;

↓  
· Manca ancora qualcosa? Sí, il "contrario" (avrò da una variabile prendo l'indirizzo)

altro

Scambia( $x, y$ )  $\xrightarrow{\text{diventa}}$  Scambia( $\&x, \&y$ );  
allora perché

scanf("r.d",  $\&x$ );?

vorrei che la modifica sia  
visibile ANCHE del blocco "padre"

· I param. come sono passati?

↳ SEMPRE per valore  $\rightarrow$  si copia l'INDIRIZZO,  
in C/C++ il che è un valore

Ese: funz che incrementa n si ripercorre sul fronte chiamate

void inc(int \*n)

{

\*n = \*n + 1;

}

la chiamo come inc( $\&x$ );

int main(Void)

{ int x=1;

inc( $\&x$ );

}

$p$	$\sigma$
$\boxed{x   e_0}$	$\boxed{e_0   1}$
$\boxed{x   e_1}   \boxed{e_1   e_0}  $	

• le fun con i pfr. sono SEMPRE void?

( $\Rightarrow$  i pfr. VANNO sull'elenco (risegno memoria)

Ese: funzione "somma prodotto" che prende almeno 2 parametri  
 $x, y$  e fornisce 2 risultati:  $(x+y)$  e  $(x*y)$   
 $\downarrow$   
 $\hookrightarrow$  termine volutamente ambiguo  $\rightarrow ?$

mia soluzione:  
void SP(int a, int b, <sup>int</sup>\* <sub>int</sub>s, <sup>int</sup>\* <sub>int</sub>p)  
{  
 \*s = a + b;  
 \*p = a \* b;  
}

$\hookrightarrow$  la chiamavo come  $\underbrace{SP(\dots, \&s, \&p)}_{\text{rel main}}$   $\rightarrow$  i risultati

Oss: "fornisce" NON deve NECESSARIAMENTE dire che ... : risultati dove vuoi

è possibile  
farlo void SP\_2(int \*x, int \*y)

con  
2 var.

int sum = \*y + \*x;

int prod = \*y \* \*x;

\*x = sum;  
\*y = prod;

int tmp = <sup>una variabile</sup> <sub>di appoggio</sub> \*x;

\*x = \*x + \*y;

\*y = tmp \* \*y;

}

SENZA

void  $\underbrace{SP_{\text{NO APP}}}_{*x = *x + *y}(\text{int } *x, *y) \{$

$$x = (x + y)$$

Oss:

$$(x + y) * y = xy + y^2$$
$$\underline{xy + y^2 - y^2}$$

$$*y = (*x) * (*y) - (*y * *y);$$

}

15.11.23 : Array

- Come si fa a rappre certe info? (insiemi)

↳ Vettore → Come in algebra

$$\hat{x} \in \mathbb{R}^n, \quad \hat{x} = (x_1, \dots, x_n)$$

Sintassi

int, float, ...  
tipi nome[n]j → numero di elementi

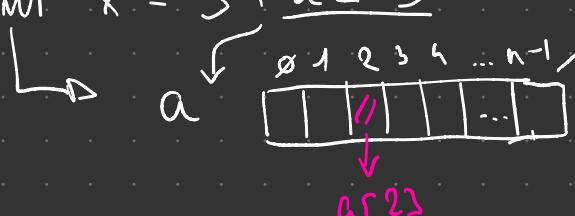
ILLEGALE!

```
int x; → non noto a priori  
scanf("f.d", &x);  
int a[x];
```

Pur farlo useremo la memoria dinamica

Allora per ora facciamo che x è noto

• int x = 3 + a[2]



! Si conta da 0  
perché? x costanza

• int a[7] = SI →

Esempio : int a[5];

a[0] = 2;

a[1] = -1; alt.

a[2] = 3;

a[3] = 4;

a[4] = -1;

int a = {2, -1, 3, 4, -1};

Voglio azzardare l'elenco di a

↓

for (int  $i=0$ ;  $i < n$ ;  $i++$ ) { if ( $a[i] < 0$ )  $a[i] = 0$ ; }

• Qs: Incremento tutti elementi pari

• Qs: Calcolo la somma degli elementi di un array di 5 el.  
↳ elementi li metto io.

int  $a[5] = \{ 7, -2, 2, 0, 0 \}$

// calcolo somma

int  $s=0$ ;

for (int  $i=0$ ;  $i < 5$ ;  $i++$ ) {  $s += a[i]$ ; }

• Qs: Parto da un array con  $n$  fissato,  
vooglio definire un altro array  $b^{[n]}$  che contiene  
in  $b[i] = \sum_{x \leq i} a_i$

int  $a[5] = \{ 1, 2, -2, 0, 7 \}$ ;

int  $b[5]$ ;

int  $s=0$ ;

for (int  $i=0$ ;  $i < a$ ;  $i++$ )

{  $s += a[i]$ ; alt.

}  $b[i] = s$ ;

uso prop

$$\sum_{i=0}^{j+1} a_i = \sum_{i=0}^j a_i + a_{j+1}$$

$b[0] = a[0]$ ;  
for (int  $i=1$ ;  $i < n$ ;  $i++$ )

{

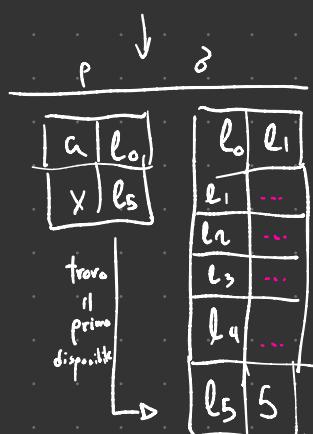
$b[i] = b[i-1] + a[i]$

}



• Qi ha

```
int a[4];
int x = 5;
```



*note qualsiasi gli array sono PUNTATORI*

$a[0] \rightarrow$  vanno in memoria contigue (uno sopra l'altro)  
 $a[1]$   
 $a[2]$   
 $a[3]$

Schema generale: f. con array

int somma\_elementi (int a[], int n)

{ ... }

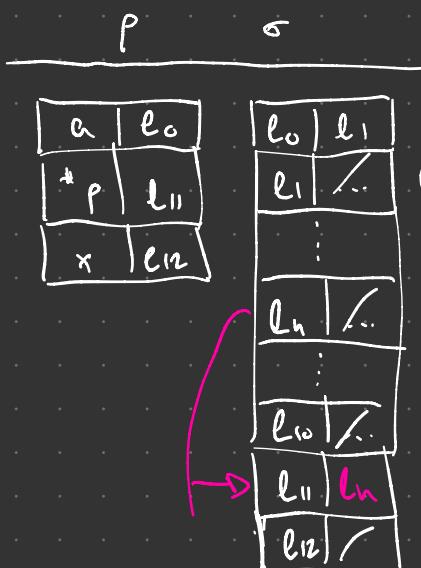
*il TIPO degli elementi*

qui non esiste  
(en LAS)

*ad un array di n elementi, la DIMENSIONE*

Q.s.

```
int a[10];
int *p;
int x;
p = &a[3];
```



$a[0]$

$a[3] \rightarrow 6 \cdot a[3] = l_4$

$a[9]$

Ese:

int  $a[10]$ ,

int \*p;

int

$p = \&a[3];$

$x = *(p+1) \rightarrow p(p[l_4+1])$

a	e <sub>0</sub>
*p	e <sub>1</sub>
x	e <sub>2</sub>

e <sub>0</sub>	e <sub>1</sub>
e <sub>1</sub>	...
l <sub>0</sub>	...
l <sub>1</sub>	...
l <sub>2</sub>	...
l <sub>3</sub>	...
l <sub>4</sub>	...
l <sub>5</sub>	...
l <sub>6</sub>	...
l <sub>7</sub>	...
l <sub>8</sub>	...
l <sub>9</sub>	...

$a[0]$

$a[3]$

$a[9]$

$\rightarrow a[n]$

Aritmetica dei sto sommendo  
indirizzi  
puntatori)

$l_4 + 1 = l_5$

$*(l_5) = a[4]$

Si può fare  
i.e. contiene

Se  $x = \underline{*(a[4])}$

$\downarrow$   
 $*(l_5) = a[4]$

Cos'è  $*(a+0)$ ? è  $a[0]$

$*(a+1) \rightarrow a[1]$

Le scritture sono  
EQUIVALENTI.

Si può avere avere ARRAY DI PUNTATORI

int \*\*a;

e <sub>0</sub>	e <sub>1</sub>
e <sub>1</sub>	e <sub>2</sub>
e <sub>2</sub>	e <sub>3</sub>
e <sub>3</sub>	e <sub>4</sub>
:	:
e <sub>n</sub>	e <sub>i</sub>



Un array di  
n elementi  
è una  
MATRICE

17.11.23 : Array, parte 2

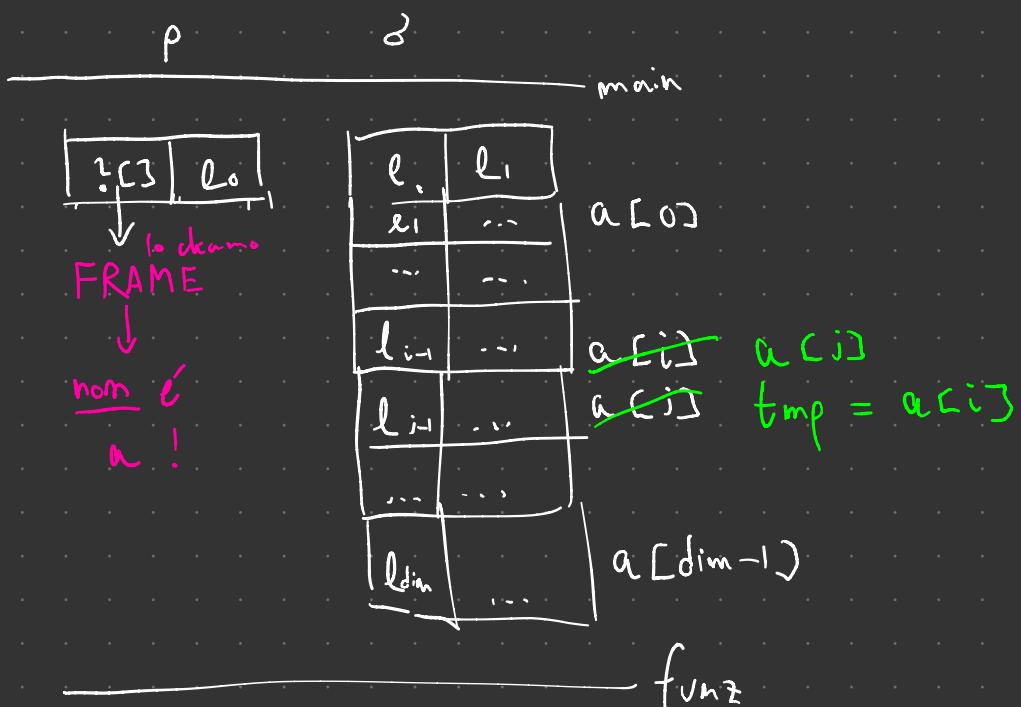
Sia void scambia(int v[], int dim, int i, int j)

```
{  
    if (i > dim || j > dim || (j * i) < 0) {}  
    else  
    {  
        int t = v[j];  
        v[j] = v[i];  
        v[i] = t;  
    }  
}
```

uso l'antitica dei puntatori:

```
void scambia(int *a; int dim, int i, int j)  
{  
    if (i < dim && j < dim && i * j > 0)  
    {  
        int tmp = *(a + (i));  
        *(a + (i)) = *(a + (j));  
        *(a + (j)) = tmp;  
    }  
}
```

# Memoria



funz

*a	l <sub>A</sub>
dim	l <sub>B</sub>
v	l <sub>C</sub>
j	l <sub>D</sub>

l <sub>A</sub>	l <sub>E</sub>
l <sub>B</sub>	dim
l <sub>C</sub>	j
l <sub>D</sub>	j

copio SOLO il  
produtt

da stabilità

f. if

*tmp	l <sub>E</sub>
------	----------------

l <sub>E</sub>	*(a+i)
----------------	--------

$\rightarrow l_1 + i = \text{FRAME}[i]$

$= ??$  (un valore)

## Caso matematico

1)  $\exists x : a[x] = v \rightarrow$  c'è almeno un elemento  $v$  in  $a$

2)  $\forall j, a[j] = v \rightarrow$  tutti gli elementi  $v$  sono  $v$

3)  $\forall j \geq 1, a[j] > \sum_{i=0}^{j-1} a[i]$   $\rightarrow$  a partire dal secondo elemento, tutti gli elementi di  $a$  sono più grandi della somma di tutti gli elementi fino all'el. precedente

0	j-1	j	i	n-1
(1)	...	(1)	(1)	...

$$\sum_{0 \leq i < j} i$$

1)  $\text{int esiste\_v}(\text{int } a[], \text{int dim}, \text{int } v)$

```

    {
        int flag = 1;
        for (int i=0; i < dim && flag; i++) alt.
        {
            if (a[i] == v) flag = 0;    ↓
        }                          ↓
        return (1 - flag);
    }

```

$\text{int trovato} = 0;$   
 è un "determinante";  
 → al MASSIMO dura  $\text{dim}$  passi → ma può essere più breve  
 $\text{while } C!.\text{trovato} \& B \text{ dim} > 0)$   
 { trovato =  $a[\text{dim}-1] == v$ ;  
 dim --; } → alt. trovato  
 return ( $\text{dim} \geq 0$ );  
 ↳ alla fine se NON mi sono mai fermato

2)  $\text{int tutti\_v}(\text{int } a[], \text{int dim}, \text{int } v)$

```

    {
        int flag = 1;
        for (int i=0; i < dim && flag; i++) alt.
        {
            if (a[i] != v) flag = 0;    →
        }                          ↓
        return flag;
    }

```

$\text{while } (\text{flag} \& \text{dim} \geq 0)$   
 { flag =  $a[\text{dim}-1] != v$ ;  
 dim --; }  
 return flag;

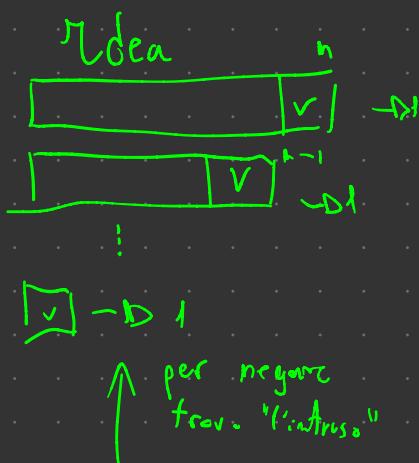
1) ricorsiva

$\text{int esiste\_v}(\text{int } a[], \text{int dim}, \text{int } v)$

```

    {
        if (dim < 0) return 0;
        if (a[dim] == v) return 1;
        return esiste_v(a[], dim-1, v);
    }

```



$\text{int tutti\_v}(\text{int } a[], \text{int dim}, \text{int } v)$

```

    {
        if (dim < 0) return 1;
        if (a[dim] != v) return 0; alt.
        return tutti_v(a[], dim-1, v);
    }

```

$\text{if } (\text{dim} == 0) \text{return } (a[\text{dim}] == v);$   
 $\text{return } (a[\text{dim}] \& tutti_v(a[], \text{dim}-1, v));$

1 settimana per fare tutti gli esercizi sugli Array,  
la memoria → da fare: memoria dinamica (malloc)  
programmazione delle liste  
Python (punti sulle classi, iteratori)

Collegamento successioni con array

Ordinare array (sorting)

# 29.11<sup>23</sup>: Gestione della memoria

fin'ora ho auto:

- `int A[10];` → 

Questa è STATICA  
nolo faccio al momento della dichiaraz.

Limitazione: DEVO conoscere la dim a priori

- Per superare ho la MEMORIA DINAMICA

ho dei costrutti: - m alloc = memory allocation

Allora faccio

- `int *array = (int *) malloc(sizeof(int) * 10);`

Cos'è il CAST; qualunque cosa restituisce malloc direttamente un puntatore intero

→ ex: `(double)7` → 7.0  
→ qui è esplicito

malloc(sizeof(int) \* 10)  
→ QUANTO spazio devo allocare?

Spazio di un qualsiasi intero

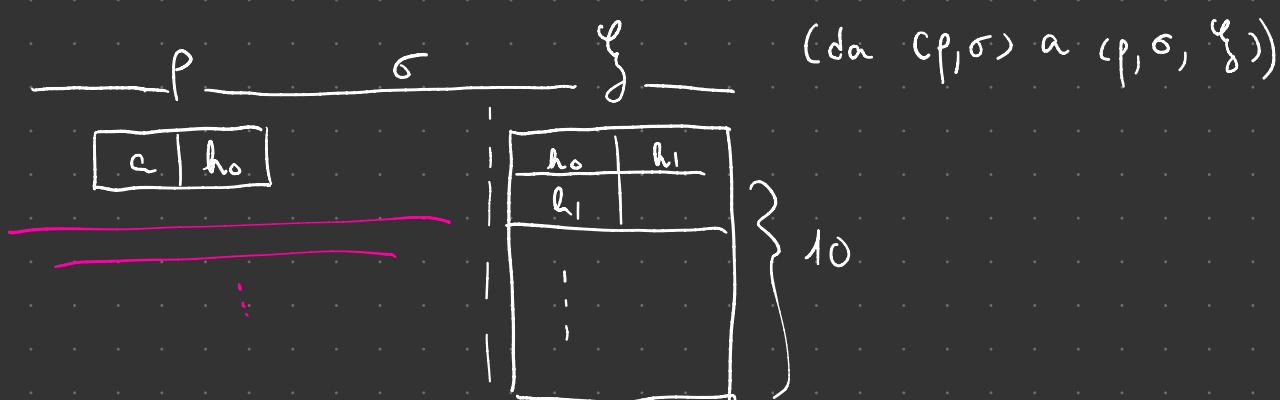
un n ∈ N qualsiasi

→ di solito

un int occupa 4 bit; però può dipendere per calcolatore per calcolatore

Lo Auto della memoria dinamica

- prima introduciamo l'ultimo concetto: HEAP ({}')



Questa inserisce dei Frame

lo HEAP è indipendente dai frame

Cosa devo fare a g, alla fine? devo DISTRUGGERLO

con op: · free (array)

se non lo uso lo rischio:

while C...)

{

int \*a = ... malloc(...);  
... → non metto free(...)

}

continua ad allocare  
array senza distruggere

Lo g cresce fino a ∞  
ma la RAM non è infinita

=> ERRORE!

A questo punto devo

usare l'aritmetica dei puntatori

Ese] Sia

Allora la memoria è:

int x = 10;

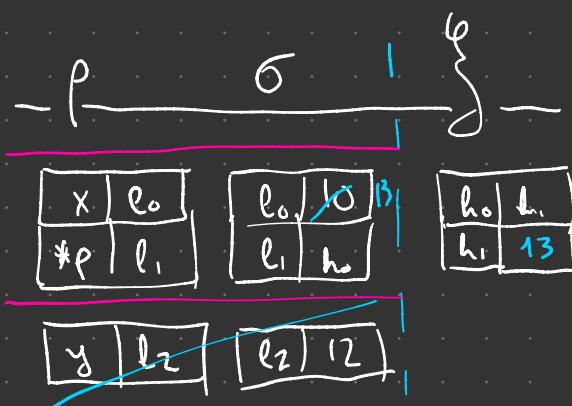
int \*p;

{ int y = 12;

p = (int \*)malloc(sizeof(int));

\*p = y + 1;

x = \*p;



OK, ma manca il  
free(p);



Esercizio Array con dim variabile, funzione che riempie l'array con 0

```
#include <stdlib.h>
#include <stdio.h>
```

```
void riempি_zeri( int *A, int dim )
```

```
{
```

```
    for (int i=0; i<dim; ++) *(A+i) = 0;
```

```
}
```

```
int main(void)
```

```
{
```

```
    int n;
```

```
    int *A;
```

```
    printf("Inserisci dimensione dell'array: ")
```

```
    scanf("%d", &n);
```

```
    A = (int *)malloc(sizeof(int) * n);
```

```
    riempি_zeri(A, n);
```

```
    free(A);
```

```
    return 0;
```

```
}
```

30.11.23

4

a → n

b → m

$$\exists i \in [0, n] \mid a[i] = \sum_{j=0}^{m-1} b[j]$$

int predicate ( int \*A, int n, int \*B, int m )

{ int flag;

int r = 0;

int s = 0;

// calcolo s

for ( int i=0; i < m; i++ )

{ s += \*(b+i);

}

// controllo sul predicato

while ( !flag && n > 1 )

{

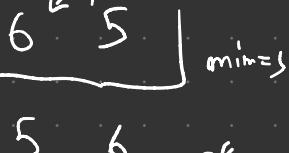
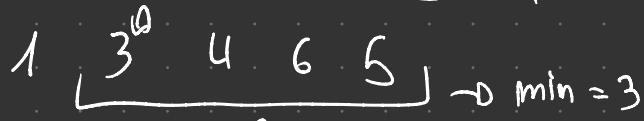
if ( \* (a + n - 1) == s ) flag = 1;

n --;

}

return flag;

}



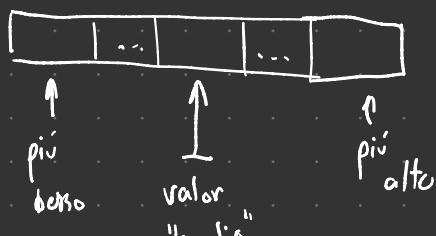
5    6    ok,

01.12. '23 | Es, liste linkate



$\exists x \in A^2 \rightarrow$  faccio un ciclo for

Se A fosse ordinato



$$\Rightarrow \text{sia } m = a[n/2]$$

Se  $x = m \rightarrow$  OK

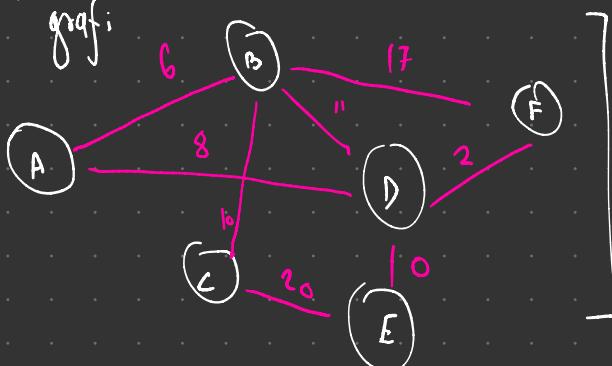
Se  $x > m \rightarrow$  considero l'array  $(m/2, n]$

Se  $x < m \rightarrow$  considero l'array  $[0, m/2)$

poi ripeto finché:  
 - trovo che  $x \in A$   
 - "esaurisco" lo spazio  $\rightarrow$  non c'è

questa è la c.d. "ricerca binaria"

los es di  
grafi:

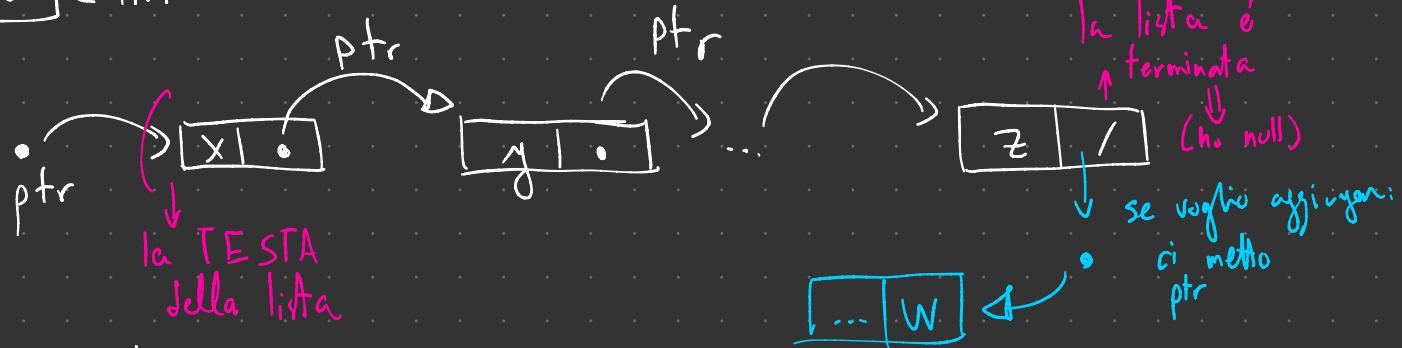


vorrei percorso  
"di costo minimo"  
minimizzare  
los ex con  
Google Maps



Limiti della mem. dinamica  $\rightarrow$  rigidità

Idea | (lide linkate)



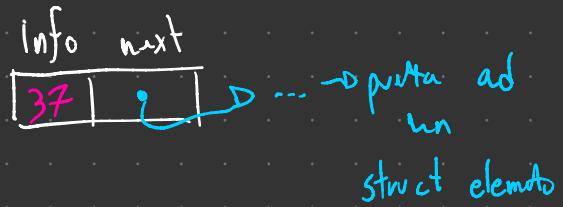
- svantaggio: navigabile in una SOLA direzione (parte dalla testa)

Per farlo si deve introdurre un concetto

OGETTO | (struct)  $\rightarrow$  antecedente dell'oggetto di OOP

struct elemento

```
{  
    int info;  
    struct elemento *next;  
}
```



ex:  $L = x | y | z$

$x \oplus L = y | z$

Altro caso while

```
#typedef struct elemento Elementodilista
```

↳ Cosa manca?  
Il puntatore alla testa

```
#typedef struct Elementodilista * Listadielementi;  $\rightarrow$  puntatore ad una struct
```

infine

ho:

`typedef struct ElementoDiLista {`

ListaDiElementi lista;

`- lista = (int *) malloc (sizeof (ElementoDiLista));`

`(* lista).info = 10;`

`(* lista).next = NULL;`

dereferenzio  $\downarrow$   $\rightarrow$  accedo ai rispettivi "attributi"

è intelligente perché conosce la dimensione d' strutt. elemento

su di

dove alloc lo spazio necessario  
x 1 int e un puntatore

{s] Programmare da crea lista con primi  $n \in \mathbb{N}$

...  $\rightarrow$  strutt, typedef

ListaDiElementi: L;

L = (int \*) malloc (sizeof (ElementoDiLista));

(\*L).info = 0.

(\*L).next = NULL;

int n = ...;

for (int i=1; i <= n; i++)

{

ListaDiElementi = L;

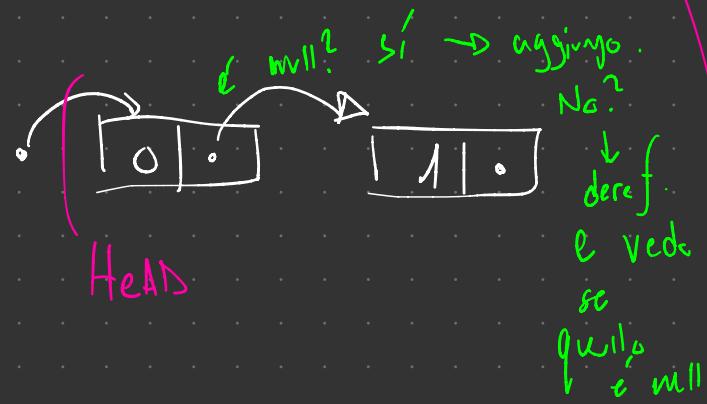
L' = (int \*) malloc (sizeof (ElementoDiLista)); \*(L).info = n;

ElementoDiLista X = (\*L);  $\rightarrow$  punto della testa L

while ((\*X).next != NULL) X = (\*X).next;

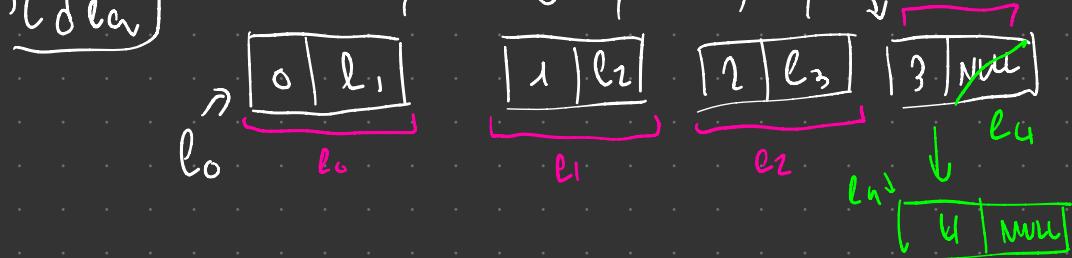
(\*X).next = L';

}



No.  
deref.  
e vede  
so  
quando  
è null

Il deca



il puntatore  
al prox

vado  
filo  
alla radice

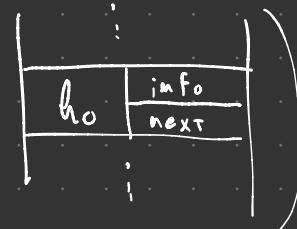
```

ListaDiElementi lista;
int N = ...;
lista = new;
ListaDiElementi new = malloc(sizeof(CElementoDiLista));
(*new).info = i;
for (int i = 2; i <= N; i++)
{
    (*new).next = malloc(sizeof(...));
    new = new -> next;
    new.info = i;     -> (*new).next
}
new->next = NULL;

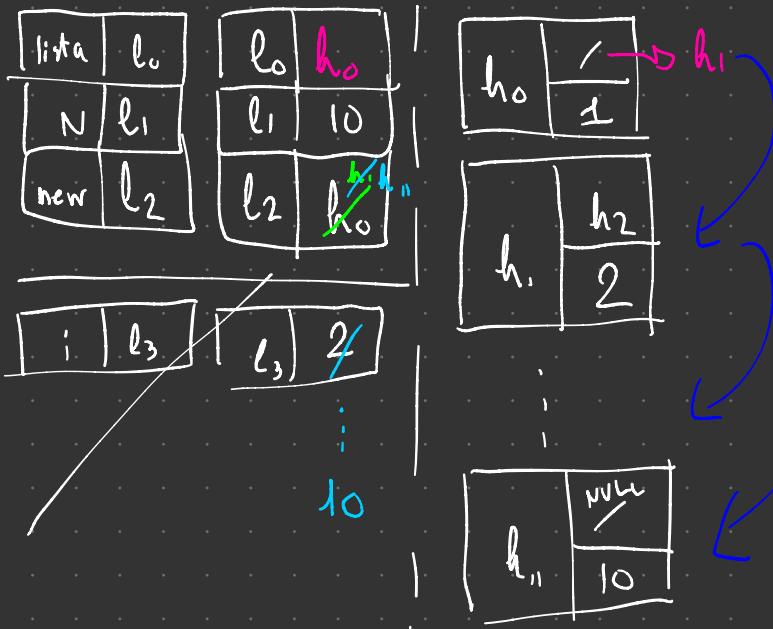
```

# La Memoria

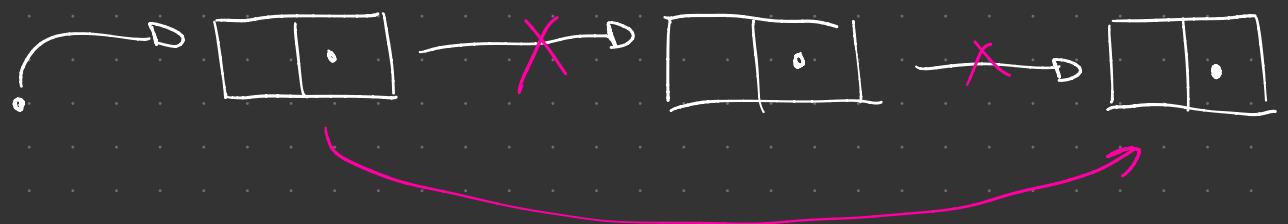
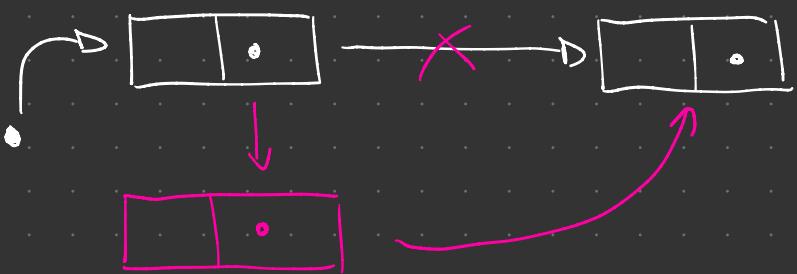
(Per disegnare strutt.)



P σ | g



• cosa si puo fare



6. 12. 23

## Esercizi sulle liste

### Es 1

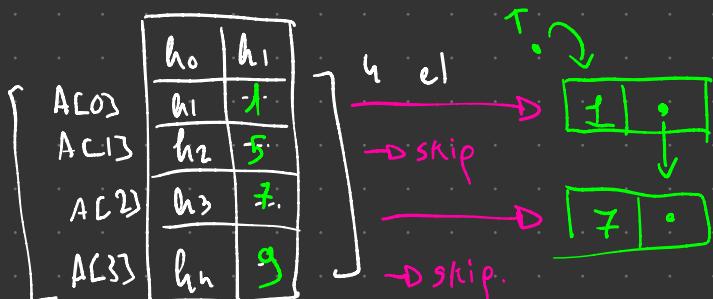
Array di n el. interi

→ Array di indice pari NON so dim a priori

→ aggiungo in mezzo alla lista il terzo el.

→ stampo contenuto

→ libero memoria occupata di TUTTO



↳ iterare con for

Ponte 1.

1. Lista, ElementoDiLista (typedef...)

strutt ElementoDiLista

int info;

ElementoDiLista\* ptr;



Lista Head;

Lista New = malloc(sizeof(ElementoDiLista)); Head → ptr

int \* A = ...; int dimA = n; <sup>calloc</sup> 0

for (int i=1; i < dimA; i++) Head → info = \*(A+i)

{

if (i%2 == 0)

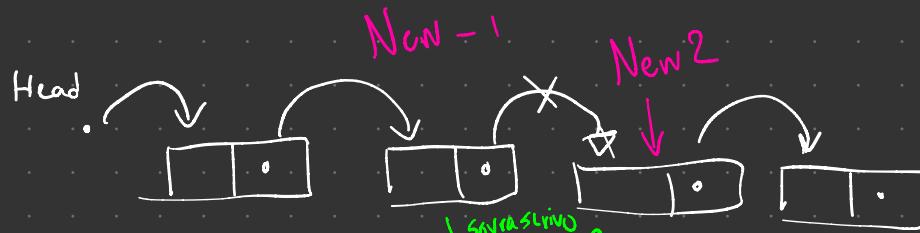
{ New → ptr = malloc(sizeof ... )

New = New → ptr

New → info = \*(A+i) }

## 2) In mezzo alla LISTA

↓  
creo il contatore ctr e faccio  $ctr/2$  j lo uso per "item" la nuova lista



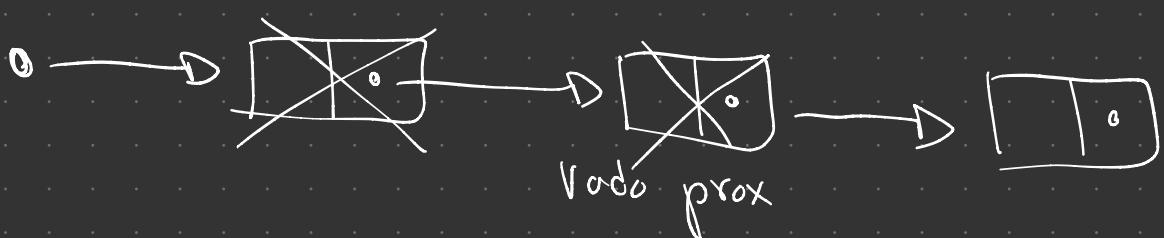
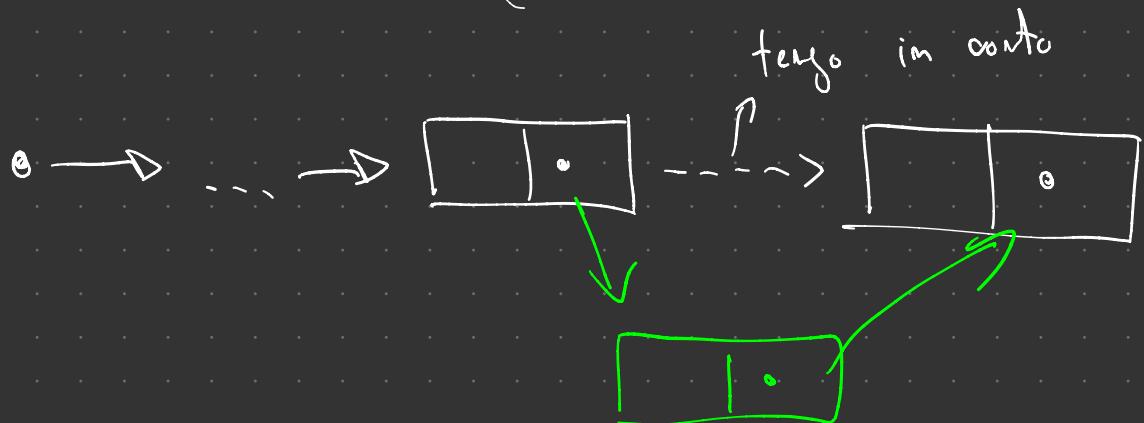
$ctr = h;$

- aggiungo ctr

New2

Stop here;  
tengo conto questa  
è quella di prima

~~$New_1 \rightarrow ptr = malloc \dots$~~   
 ~~$X = New_1 \rightarrow ptr$~~   
 ~~$X \rightarrow info = a[3];$~~   
 ~~$X \rightarrow ptr = New2$~~



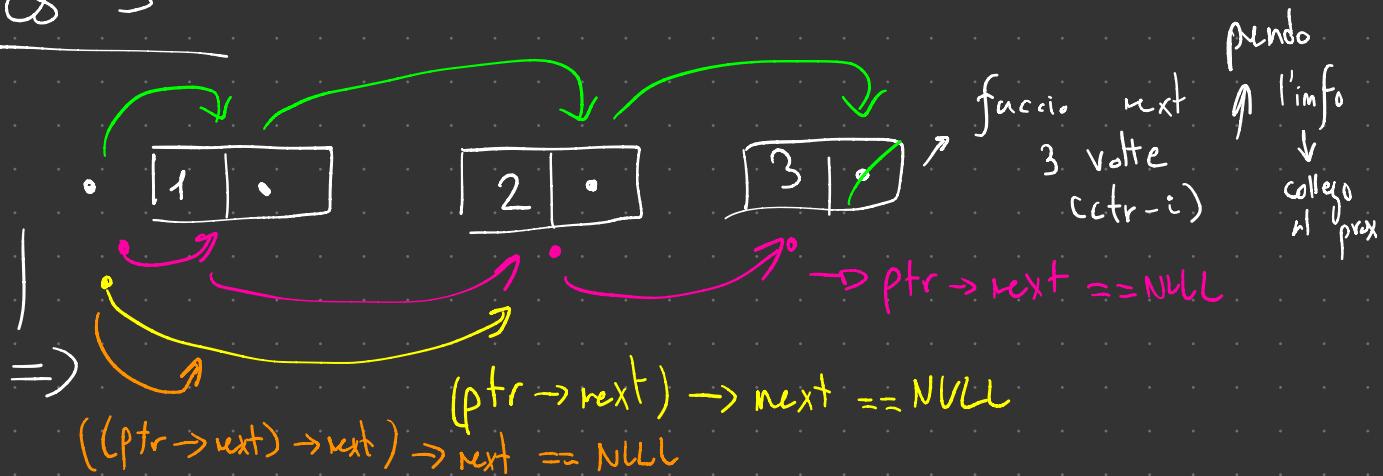
2 ptr d'appoggio

Domanda ric. sulle liste

"Memoria con struct, malloc...

Esercizio 3

$$0 \leq i < 3$$



Ideas

lista InvartiLista(lista Head)

{ // 1 → trovo dim lista

list<sup>a</sup> New\_Corsa = Head; int ctr = 0;

while (New\_Corsa != NULL)

{

    New\_Corsa = New\_Corsa → next;

    ctr ++;

}

// creo nuova lista

lista HeadRisultato; lista New\_Risultato; HeadR = NewR;

// vado all'elemento i-esimo, i → Ar e tayo Ar-i

for (int i = ctr - 1; i ≥ 0; i--)

{ list<sup>a</sup> New\_Esplora = Head;

    for (int j = 0; j < i; j++)

        New\_Esplora = New\_Esplora → next;

    New\_Risultato = New\_Esplora → info

    New\_Risultato → next = malloc ...

    New\_Risultato = New\_Risultato; } return Head\_Risultato;

# 13.12<sup>2023</sup>: Iteratori (Python)

Classe: O messo; OOP → tutto è un oggetto; ovvero l'istanziamento di una classe  
↓  
il senso? → creare ordine del mondo fisico (vero)  
→ rappresentare gli oggetti in una maniera unica → dal generale alla specifica  
in programmazione

- ho
  - [for m in range(10)]: quadrati.append(n\*n) → NON ha nessun valore di ritorno → agisce direttamente sul PUNTATORE
  - ↓ lo riscrivo come  
→ quadrati = [n\*\*2 for m in range(10)]

Questa è una LIST COMPREHENSION

- La forma generica è struct
  - [f(x) for x ∈ X]
    - ↓ ↓  
m\*\*2 x in range(10)    - ↳ si presuppona che X è una collezione iterabile → posso "farsi un percorso"  
↳ q. w. una struttura intrinseca
  - ↓ restituisce una lista;
  - ↓ uso l'iteratore

$[f(x_1), \dots, f(x_n)]$  per  $X = \{x_1, \dots, x_n\}$

- Iteratore → un oggetto su cui posso iterare con nozione implicita di iterazione

(istanziamento di classe)  
OGGETTO

2 metodi → ITER: inizializza lo stato  
NEXT: va all'oggetto successivo (cammino)

ex: stringa

↳ x = "GIULIO" → iter = iter(x) next(iter) → I  
next(iter) → G  
next(iter) → O

### ex] Dizionario

- Abaco ~ iter

next →

next →

next →

next → - Zuzzurrolo

Esempio: Voglio un iteratore infinito che crea numeri.

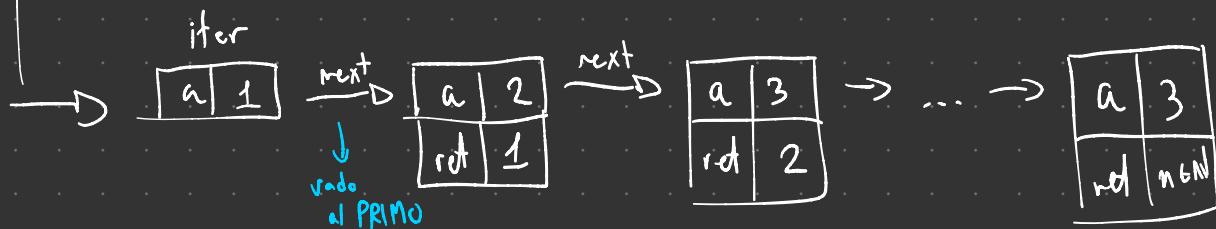
class MyNumeri:

def \_\_iter\_\_(self): → si ricorda quale elemento siamo

self.a = 1  
return self

def \_\_next\_\_(self):

self.a += 1  
return self.a - 1



Voglio pari?

a init → 2

a next += 2

ritorno a - 2

# Esercizio (primo iteratore)

- Parte da  $a$ , genera  $a+1, \dots, a+2$  fino a  $b$

class Iterab:

```
def __init__(self, a, b):
```

```
    self.a = a
```

```
    self.b = b
```

```
def __iter__(self):
```

```
    self.x = self.a
```

```
    return self
```

```
def __next__(self):
```

```
    if self.x > self.b:
```

```
        return None
```

```
    else:
```

```
        self.x += 1
```

```
        return self.x - 1
```

dovrei usare  
il raise

## raise StopIteration

→ segnala "un comportamento  
de non deve avvenire"

→ un caso in cui il  
next non è definito

a	1
b	5

↓  
\_\_iter\_\_

x	1
---	---

x	2
x-1	1

x	3
x-1	2

x	4
x-1	3

x	5
x-1	4

x	6
x-1	5

→ FINITO

x	7
att	None

→ None → ...

• Sia

for element in iterable: ] → cosa fa. / [ n in iterable]

STRUTTURA

...  
iter\_obj = iter(iterable) → chiama \_\_iter\_\_

while True:

try:

element = next(iter\_obj)

except StopIteration:

break

→ devo essere stufo  
di RAGGIUNGERE qui

## Esercizio

[]} iteratore con pow-2 da min a max

class PowDive:

def \_\_init\_\_(self, min, max):

self.m = min

self.M = max

def \_\_iter\_\_(self):

self.i = self.m

while not is\_pow2(self.i):

self.i += 1

return self

def \_\_next\_\_(self):

if (self.i > self.M):

raise StopIteration

else:

self.i \*= 2

return self.i / 2

alt:

def \_\_iter\_\_(self):

i=1

while (i < self.m)

i\*=2

self.i = i

return self

→ meglio con grandi numeri

f. ausiliaria  
che ritorna true o...

false alt.

def is\_pow2(x):

f = True  
while x != 1 and f:

if (x % 2 == 0):

f = False

else:

x /= 2

return f

15.12.23:

Oss: è necessario M

1) Si consideri per  $0 \leq x \leq y$

$$P(x, y) = \left\{ z \in \mathbb{N} \mid \begin{array}{l} \text{d'altro} \\ \text{tutti i numeri tra} \\ \text{di questi} \end{array} \right\} \begin{array}{l} \text{condizioni} \\ \text{regolari} \end{array} \xrightarrow{x < y} z < y - x$$

Fornire classe python di generi  $P(x, y)$  per una qualunque coppia  $x, y$

cls,  $x=3, y=11$

5, 7

$$\hookrightarrow P(x, y) = \left\{ z \in \mathbb{Z}, \mathbb{D} \mid \begin{array}{l} z + x < y \\ 3 \leq z \leq 11 \wedge z < 8 \end{array} \right\} \begin{array}{l} \{ 5 \leq z \leq 7 \\ z < 7 - 5 = 2 \end{array}$$

1.2) Class N:

def \_\_init\_\_(self):  $x + z < y$   
self.x = x  
self.y = y  $z < y - x$

def \_\_iter\_\_(self):  $z \geq 20$   
 $z \leq 30$   
 $z < 10$

self.z = x  
return self  $z > 30$   
 $z \geq 10$

def \_\_next\_\_(self):  $z$  if  $z > 30$

if (self.z + self.x > self.y or self.z > self.y)  
raise StopIteration

else:

self.z += 1  
return self.z - 1

2.1) Definire iteratore in PY  
per generare intervallo chiuso  $[a,b]$  divisibili  
per interi  $> 0$

(loss Ass:

```
-- init --
self.a=a
self.b=b
self.x=x
-- iter --
self.z = a
return self
-- next --
```

while ( self.z < self.x ) = 0 and self.x <= b)

    self.z += 1; self.z < self.x = 0  
if (self.z > b or self.z <= 0) : raise ...

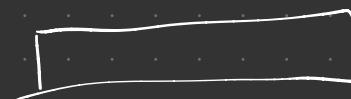
else: self.z += 1; return self.z - 1

7 18 9 10 11 12  
↑ ↑ ↑ ↑ ↑ ↑ ↑

Cos'è  $i \cdot x$ ?

$\hookrightarrow i = 6 \quad x = 5$

$i \cdot x = 1 \rightsquigarrow i$  la "distanza"  
 $\downarrow$   
 $i = (x - i \cdot x)$



$i = 2 \quad x = 6$

$\hookrightarrow 2 \cdot 6 = 4$

$2 + i$