## Funzioni e Ricorsione

- La ricorsione consiste nella possibilità di *definire una funzione in termini di se stessa*

- È basata sul *principio di induzione* matematica:
  - se una proprietà P vale per $n=n_0$
  - e si può provare che, *assumendola valida per n*, allora vale per n+1

  allora P vale per ogni $n \geq n_0$

1

## Ricorsione

- Operativamente occorre:
  - identificare un "**caso base**" la cui soluzione sia "ovvia"
  - di riuscire a **esprimere la soluzione al caso generico *n*** in termini dello *stesso problema in uno o più casi più semplici* (n-1, n-2, etc).

2

## Ricorsione – Esercizi

- Esprimere la soluzione di problemi per via funzionale tramite la ricorsione
  1. Calcolo della funzione H(n) = $\sum_{k=1}^{n} \frac{1}{k}$

     H(n) = 1 + 1/2 + 1/3 + … + 1/n
  2. Calcolo della potenza k-esima

     $b^k$ con $b \in Z$, $k \geq 0$
  3. Calcolo del valore di un polinomio di grado n a coefficienti unitari

     P(x,n) = $x^0 + x^1 + \ldots x^n$

3

## Funzione H(n)

- H: N → R (int → double)

  H(n) = 1 + 1/2 +1/3 + ... + 1/n
- **Per n>1** la funzione si riscrive come:

  H(n) = (1 + 1/2 +1/3 + ... + 1/(n-1)) **+ 1/n**

  ossia come

  H(n) = H(n-1) + 1/n

  mentre, ovviamente, H(1)=1

4

## Funzione H(n)

- Dunque,

  H(n) = 1 per n = 1

  H(n) = 1/n + H(n-1) per n > 1

```
double H(int n)
{
    return (n == 1) ? 1
                    : 1.0/n + H(n-1);
}
```

Altrimenti che succede?!

5

## Potenza k-esima

- **$b^k$ con $b \in Z$, $k \geq 0$**
- **power**: $R \times N \to R$ (double×int → double)

  **$b^k = 1$ per $k = 0$**

  **$b^k = b * b^{k-1}$ per $k > 0$**
- da cui:

```
double power(double b, int k)
{
  return (k == 0) ? 1 : b * power(b, k-1);
}
```

6

## Polinomio

- Calcolo del valore di un polinomio di grado $n \geq 0$ *a coefficienti unitari*

  $P(x, n) = x^0 + x^1 + \ldots x^n$

- **Per n>0** P(x, n) si riscrive come:

  $P(x, n) = (x^0 + x^1 + \ldots x^{n-1}) \mathbf{+ x^n}$

  ossia come

  $P(x, n) = P(x, n-1) + x^n$

  mentre, ovviamente, $P(x, 0) = 1$

7

## Polinomio

- Dunque,

  pol(x, n) = 1 per n=0

  pol(x, n) = $x^n$ + pol(x, n-1) per n>0

- da cui...
  1. La funzione **`pol`** accetta un **`double`** (**`x`**) ed un **`int`** (**`n`**)
  2. Se n è uguale a zero, restituire **`n`**...
  3. ...altrimenti restituire la somma di $x^n$ – **`power(x, n)`** – del risultato della funzione **`pol`** stessa invocata con i valori **`x`** e **`n-1`**

8

## Polinomio

```
double pol(double x, int n)
{
    return (n==0) ? 1
      : power(x,n) + pol(x, n-1);
}
```

9

## Massimo Comun Divisore

- Trovare il massimo comun divisore tra due numeri N e M

$$\text{MCD}(m, n) = \begin{cases} m, & \text{se } m=n \\ \text{MCD}(m-n, n), & \text{se } m>n \\ \text{MCD}(m, n-m), & \text{se } m<n \end{cases}$$

10

## Massimo Comun Divisore

Codifica

```
int mcd(int m, int n)
{
  if(m == n)
     return m;
  else
     return (m > n) ? mcd(m-n, n) :
                      mcd(m, n-m);
}
```

11

## Massimo Comun Divisore... ?!

- Il risultato viene sintetizzato via via che le chiamate si aprono, *"in avanti"*
- Quando le chiamate si chiudono non si fa altro che riportare indietro, fino al cliente, il risultato ottenuto

→ La ricorsione di questo tipo viene detta *Tail Recursion*!

12

## Massimo Comun Divisore... ?!

- La soluzione individuata per l'MCD è *sintatticamente ricorsiva*...
- ...ma dà luogo ad un processo computazionale *iterativo,* quindi ad un tipo di ricorsione *tail*!
- ...è tutta colpa dell'algoritmo...

13

## Massimo Comun Divisore: versione iterativa

- Identici parametri d'ingresso (ovviamente)
- Si itera finché n != m; se i due valori sono uguali, l'MCD è il valore comune
  - Se m > n si assegna ad m il valore m – n
  - Altrimenti si assegna ad n il valore n – m

14

## Massimo Comun Divisore: versione iterativa

```
int mcd(int n, int m)
{
  int a, b;
  a = n; b = m;
  while (a != b)
  {
      if (b > a)
            b = b - a;
      else
            a = a - b;
  }
  return a;
}
```

15

## Massimo Comun Divisore: metodo di Euclide

```
int massimoComunDivisore (int n1, int n2)
{
  int resto, a, b;
  if (n1 > n2)
  {
      a = n1;  b = n2;
  }
  else
  {
      a = n2;  b = n1;
  }
  while (b > 0)
  {
      resto = a % b;
      a = b;
      b = resto;
  }
  return a;
}
```

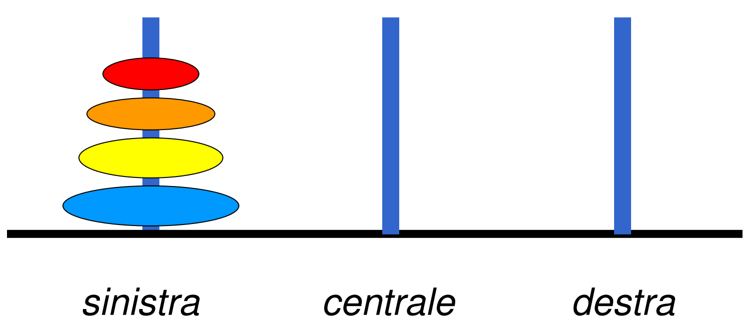
16

## La torre di Hanoi

- *Sono date tre torri (sinistra, centrale, e destra) e un certo numero N di dischi forati*
- *I dischi hanno diametro diverso gli uni dagli altri, e inizialmente sono infilati uno sull'altro (dal basso in alto) dal più grande al più piccolo sulla torre di sinistra*
- Scopo del gioco è *portare tutti e dischi dalla torre di sinistra alla torre di destra*, rispettando due regole:
  a) si può muovere un solo disco alla volta;
  b) un disco grande non può mai stare sopra un disco più piccolo.

17

## La torre di Hanoi




18

## La torre di Hanoi

**Come risolvere il problema?**

- E' impensabile immaginare la serie di mosse che risolve il problema in generale
- E' abbastanza semplice esprimere una soluzione ricorsiva

**Ipotesi:**

- E' facile spostare un <u>singolo disco</u> tra due torri a scelta.

19

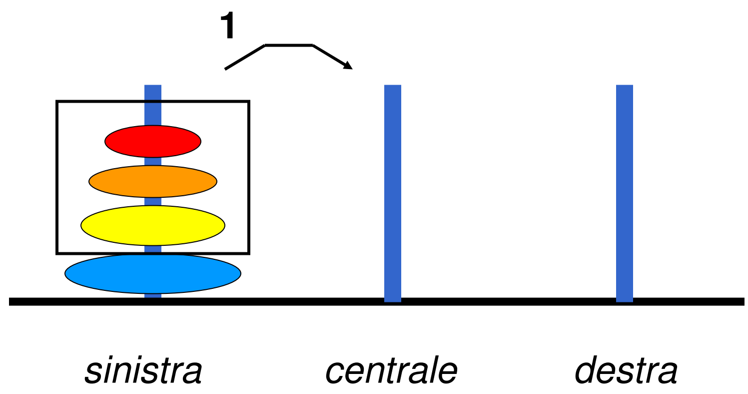
## La torre di Hanoi

**Soluzione ricorsiva**

- *<u>Caso banale</u>*: un singolo disco si sposta direttamente dalla torre iniziale a quella finale
- Caso generale: per spostare N dischi dalla torre iniziale a quella finale occorre
  - spostare N-1 dischi dalla torre iniziale a quella intermedia, che funge da appoggio
  - spostare il disco rimanente (il più grande) direttamente dalla torre iniziale a quella finale
  - spostare gli N-1 dischi "posteggiati" sulla torre intermedia dalla torre intermedia a quella finale

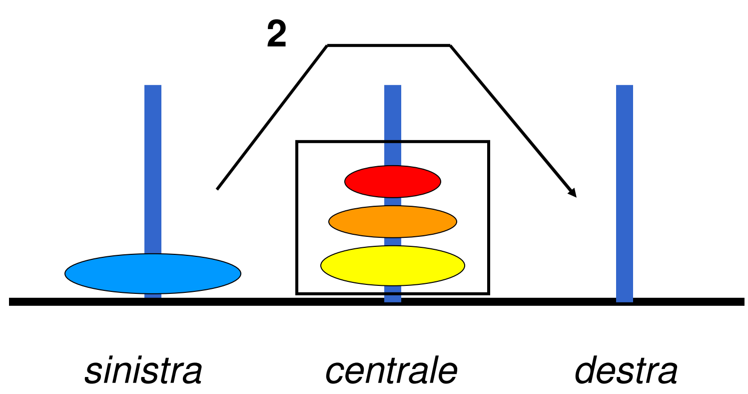
20

## La torre di Hanoi



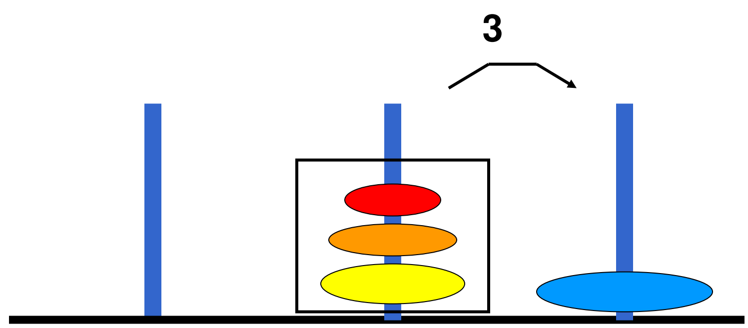

21

## La torre di Hanoi




22

## La torre di Hanoi




23

## La torre di Hanoi

- Notare che:
  - È possibile usare come torre di transito una torre dove ci siano dischi più grandi di quelli da spostare...

24

## La torre di Hanoi

**La soluzione delineata per il caso "N dischi" presuppone**

- di sapere spostare N-1 dischi
  → ***stesso problema in un caso più semplice***
- di sapere spostare un singolo disco
  → ***abilità che si possiede per ipotesi.***

**È una ricorsione *non lineare***

- il problema con N dischi si esplode in due sottoproblemi con N-1 dischi
- con N dischi, $2^N$-1 attivazioni della funzione.

25

## La torre di Hanoi

**Specifica**

- rappresentiamo le tre torri con un intero
- rappresentiamo ogni mossa tramite la coppia di torri coinvolte (in futuro le scriveremo sull'output)
- la funzione `hanoi` ha come parametri
  - il numero di dischi (`d`) da spostare
  - la torre `iniziale`
  - la torre `finale`
  - la torre da usare come appoggio (`transito`)
- non ha tipo di ritorno, è una procedura → `void`

26

## La torre di Hanoi

Pseudocodifica

1. Se c'è un solo disco da trasferire, trasferirlo direttamente dalla torre **`iniziale`** a quella **`finale`** e terminare, altrimenti...
2. ...trasferire **`d-1`** dischi dalla torre **`iniziale`** alla torre di **`transito`**
3. Trasferire un disco dalla torre **`iniziale`** alla torre **`finale`**
4. Trasferire **`d-1`** dischi dalla torre di **`transito`** alla torre **`finale`**

27

## La torre di Hanoi

Interfaccia (header file)

```
void hanoi(int dischi,
  int torreIniziale, int torreFinale,
  int torreTransito);
```

28

## La torre di Hanoi

```
void hanoi(int d, int iniziale, int finale,
  int transito)
{
  if (d==1)
  {
     printf("Muovi un disco dalla torre %d "
       "alla torre %d\n", iniziale, finale);
  }
  else
  {
     hanoi(d-1, iniziale, transito, finale);
     printf("Muovi un disco dalla torre %d "
       "alla torre %d\n", iniziale, finale);
     hanoi(d-1, transito, finale, iniziale);
  }
}
```

29

## La torre di Hanoi

Possibile estensione alla soluzione – per quando si conosceranno gli array:

- Rappresentare le torri con array di interi
  - Ogni intero all’interno dell’array rappresenta la dimensione del disco
  - Uno zero, rappresenta l’assenza del disco

30