

## Funzioni e Ricorsione

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



Nel codice di una funzione ricorsiva **compare una** (o più di una) **chiamata alla funzione stessa**

### ■ Operativamente occorre:

- **identificare un “caso base”** la cui soluzione sia “ovvia” → termine della ricorsione
- 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)

1

## Ricorsione – Esercizi

- Esprimere la soluzione dei 3 problemi seguenti tramite algoritmi ricorsivi

1. Calcolo della funzione  $H(n) = \sum_{k=1}^n \frac{1}{k}$   
 $H(n) = 1 + 1/2 + 1/3 + \dots + 1/n$

2. Calcolo della potenza  $k$ -esima  
 $b^k$  con  $b \in \mathbb{Z}$ ,  $k \geq 0$

3. Calcolo del valore di un polinomio di grado  $n$  a coefficienti unitari

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

2

## Funzione $H(n)$

- $H: \mathbb{N} \rightarrow \mathbb{R}$  (int  $\rightarrow$  double)

$$H(n) = 1 + 1/2 + 1/3 + \dots + 1/n$$

- Per  $n > 1$  la funzione si riscrive come:

$$H(n) = (1 + 1/2 + 1/3 + \dots + 1/(n-1)) + 1/n$$

ossia come

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

mentre, ovviamente,  $H(1)=1$

3

## Funzione $H(n)$

- Dunque,

$$H(n) = 1 \quad \text{per } n = 1$$

$$H(n) = H(n-1) + 1/n \quad \text{per } n > 1$$

```
double H(int n)
```

```
{
```

```
    return (n == 1) ? 1  
                : 1.0/n + H(n-1);
```

```
}
```

Se anziché  
“1.0” ci fosse  
stato “1”?

4

## Potenza k-esima

- $b^k$  con  $b \in \mathbb{Z}, k \geq 0$
- **power**:  $\mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$  (double  $\times$  int  $\rightarrow$  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);
}
```

5

## Polinomio

- Calcolo del valore di un polinomio di grado  $n \geq 0$  a coefficienti unitari  
 $P(x, n) = x^0 + x^1 + \dots + x^n$
- Per  $n > 0$   $P(x, n)$  si riscrive come:  
 $P(x, n) = (x^0 + x^1 + \dots + x^{n-1}) + x^n$   
ossia come  
 $P(x, n) = P(x, n-1) + x^n$   
mentre ovviamente  $P(x, 0) = 1$

6

## Polinomio

$$\begin{aligned} \text{pol}(x, n) &= 1 && \text{per } n=0 \\ \text{pol}(x, n) &= x^n + \text{pol}(x, n-1) && \text{per } n>0 \end{aligned}$$

da cui...

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

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

7

## Massimo Comun Divisore

Trovare il massimo comun divisore tra due numeri **n** e **m**

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

8

## 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);
}
```

9

## MCD – Particolarità e Osservazioni

---

- 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 chiamante, il risultato ottenuto
- La ricorsione di questo tipo viene detta *Tail Recursion!*

10

## MCD: Versione Iterativa

---

- Identici parametri d'ingresso (ovviamente)
  - Si itera finché  $n \neq m$ ; se i due valori sono uguali, MCD è il valore comune
- Se  $m > n$  si assegna a  $m$  il valore  $m-n$
  - Altrimenti si assegna a  $n$  il valore  $n-m$

11

## MCD: 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;
}
```

12

## MCD: Metodo di Euclide

- Esiste una versione molto **più efficiente** dell'algoritmo – usa resti anziché sottrazioni
- Supponendo che sia  $m > n$

$$\text{MCD}(m,n) = \begin{cases} \text{MCD}(m \bmod n, n) & \text{se } (m \bmod n) \geq n \\ \text{MCD}(n, m \bmod n) & \text{se } (m \bmod n) < n \\ m & \text{se } n=0 \end{cases}$$

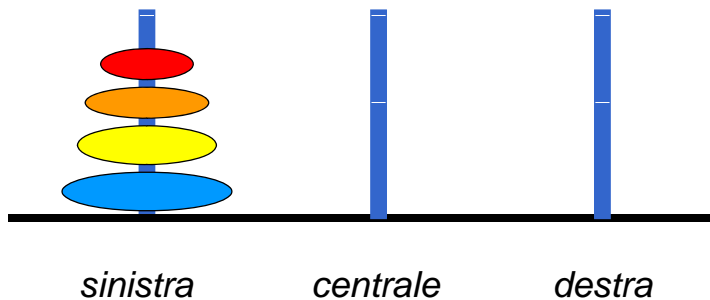
13

## 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

14

## La torre di Hanoi



15

## La torre di Hanoi

### Come risolvere il problema?

- Impensabile immaginare l'esatta sequenza di mosse che risolve il problema in generale
- Abbastanza semplice esprimere una soluzione ricorsiva

### Assunzione di fondo:

- è facile spostare un singolo disco tra due torri a scelta

16

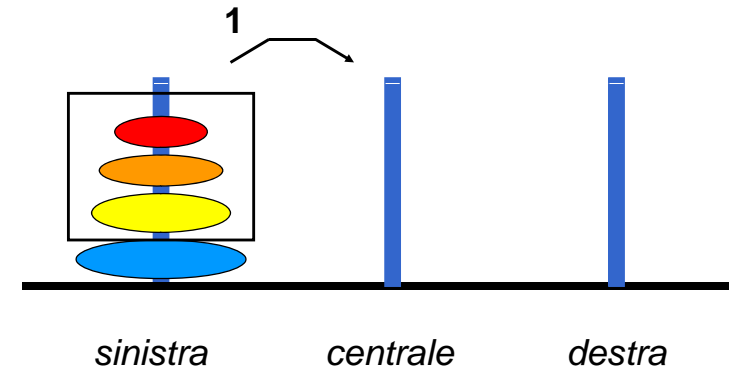
## La torre di Hanoi

### Soluzione ricorsiva

- **Caso banale:** 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

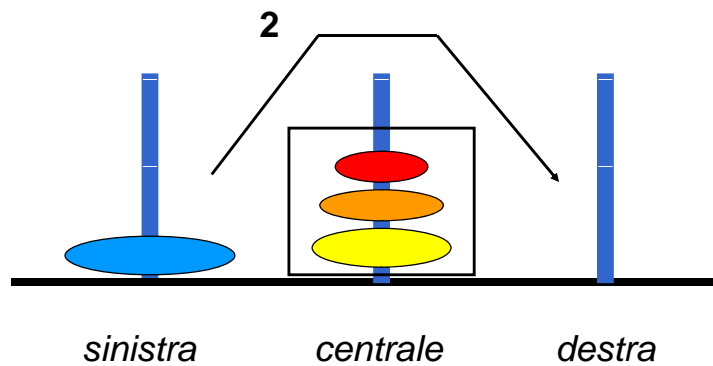
17

## La torre di Hanoi



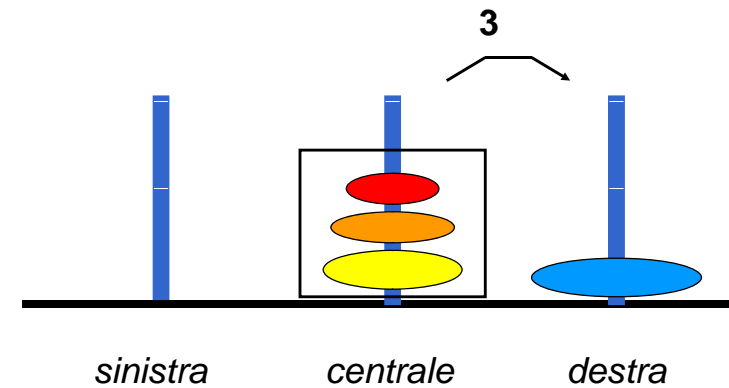
18

## La torre di Hanoi



19

## La torre di Hanoi



20

## 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...

21

## 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 espone in due sottoproblemi con N-1 dischi
- con N dischi,  $2^N-1$  attivazioni della funzione

22

## 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
  - numero di dischi (**d**) da spostare
  - torre **iniziale**
  - torre **finale**
  - torre da usare come appoggio (**transito**)
- non ha tipo di ritorno, è una procedura → `void`

23

## 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**

24

## La torre di Hanoi

---

Interfaccia (header file)

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

25

## 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);  
    }  
}
```

26

## La torre di Hanoi

---

Possibile estensione alla soluzione per quando si lavorerà con 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
- In questo modo sarà possibile mostrare ad ogni mossa lo stato di ogni singola torre

27