

## L'APPROCCIO FUNZIONALE

**Obiettivo:** esprimere la soluzione di un problema sotto forma di funzione

- Quali funzioni primitive?
- Quali meccanismi di combinazione?

## UN POSSIBILE INSIEME DI PRIMITIVE

- La funzione zero:  $zero: N \rightarrow N$
- La funzione successore:  $succ: N \rightarrow N$

Partendo da

- le funzioni primitive sui naturali
- l'operatore di uguaglianza
- le espressioni condizionali
- la ricorsione

si possono definire alcune funzioni che siamo abituati da sempre a considerare date a priori

## FUNZIONI ELEMENTARI

In questi esempi:

- denoteremo con 0 il valore restituito dalla funzione zero
- supporremo data la funzione succ tale che
$$succ: N \rightarrow N$$
$$succ(N) ::= N+1$$
- considereremo solo numeri *non negativi*

## PREDECESSORE

Predecessore in termini del successore:

$$y = \text{pred}(x) \Leftrightarrow x = \text{succ}(y)$$

```
int pred(int x){
    return (x==1) ? 0 : raggiungi(x,0);
}

int raggiungi(int x, int y){
    return (x==succ(y)) ? y :
        raggiungi(x,succ(y));
}
```

## PREDECESSORE: FUNZIONAMENTO

<i>Cliente</i>	<i>Chiama</i>
( <i>main</i> )	Pred(3)
Pred(3)	raggiungi(3,0)
raggiungi(3,0)	raggiungi(3,1)
raggiungi(3,1)	raggiungi(3,2)
raggiungi(3,2) → 2	

## SOMMA

Somma in termini di successore e predecessore:

$$x + y = 1 + ((x-1) + y)$$

```
int sum(int x, int y){
    return (x==0) ? y :
        succ(sum(pred(x), y));
}
```

### SOMMA: FUNZIONAMENTO

<i>Cliente</i>	<i>Chiama</i>
<i>(main)</i>	<code>sum(3,5)</code>
<code>sum(3,5)</code>	<code>succ(sum(2,5))</code>
<code>sum(2,5)</code>	<code>succ(sum(1,5))</code>
<code>sum(1,5)</code>	<code>succ(sum(0,5))</code>
<code>sum(0,5) → 5</code>	
	<code>succ(5) → 6</code>
	<code>succ(6) → 7</code>
	<code>succ(7) → 8</code>

### MOLTIPLICAZIONE

Prodotto in termini di predecessore e somma:

$$x * y = y + (x-1) * y$$

```
int mul(int x, int y){  
    return (x==0) ? 0 :  
           sum(mul(pred(x), y), y);  
}
```

### MOLTIPLICAZIONE: FUNZIONAMENTO

<i>Cliente</i>	<i>Chiama</i>
<i>(main)</i>	<code>mul(3,6)</code>
<code>mul(3,6)</code>	<code>sum(mul(2,6),6)</code>
<code>mul(2,6)</code>	<code>sum(mul(1,6),6)</code>
<code>mul(1,6)</code>	<code>sum(mul(0,6),6)</code>
<code>mul(0,6) → 0</code>	
	<code>sum(0,6) → 6</code>
	<code>sum(6,6) → 12</code>
	<code>sum(12,6) → 18</code>

### SOTTRAZIONE

Differenza in termini del predecessore :

$$x - y = (x - (y-1)) - 1$$

```
int diff(int x, int y){  
    return (y==0) ? x :  
           pred(diff(x, pred(y)));  
}
```

### SOTTRAZIONE: FUNZIONAMENTO

<i>Cliente</i>	<i>Chiama</i>
<i>(main)</i>	<code>diff(5,3)</code>
<code>diff(5,3)</code>	<code>pred(diff(5,2))</code>
<code>diff(5,2)</code>	<code>pred(diff(5,1))</code>
<code>diff(5,1)</code>	<code>pred(diff(5,0))</code>
<code>diff(5,0) → 5</code>	
	<code>pred(5) → 4</code>
	<code>pred(4) → 3</code>
	<code>pred(3) → 2</code>

### OPERATORI RELAZIONALI

```
int maggioreDiZero(int x){  
    return (x==0) ? 0 : succ(0);  
}  
  
int minoreDiZero(int x, int y){  
    return maggioreDiZero(diff(y, x));  
}
```

### OPERATORI LOGICI

```
int and(int p, int q){
    return p ? q : 0;
}

int or(int p, int q){
    return p ? p : q;
}
```

### ALCUNI ESERCIZI

**Obiettivo:** esprimere la soluzione di alcuni problemi *per via funzionale, sfruttando la ricorsione*

- Calcolo della funzione  $H(n) = \sum_{k=1}^n \frac{1}{k}$   
 $H(n) = 1 + 1/2 + 1/3 + \dots + 1/n$
- Calcolo della potenza k-esima  
 $b^k$  con  $b \in \mathbb{Z}, k \geq 0$

### PROBLEMA 1: 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,  $H(1)=1$

### PROBLEMA 1: H(n)

- Dunque,  
 $H(n) = 1$  per  $n=1$   
 $H(n) = 1/n + H(n-1)$  per  $n > 1$
- da cui:  

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

### PROBLEMA 2: $b^k$ 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);
}
```

### ALTRI ESERCIZI

**Obiettivo:** esprimere la soluzione di alcuni problemi *per via funzionale, sfruttando la ricorsione*

- Calcolo del valore di un polinomio di grado  $n$  a coefficienti unitari  
 $P(x,n) = x^0 + x^1 + \dots + x^n$
- Fattoriale "innovativo"

### PROBLEMA 3: 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,  $P(x,0)=1$

### PROBLEMA 3: POLINOMIO

- Dunque,

$$\text{pol}(x,n) = 1 \quad \text{per } n=0$$

$$\text{pol}(x,n) = x^n + \text{pol}(x,n-1) \quad \text{per } n>0$$

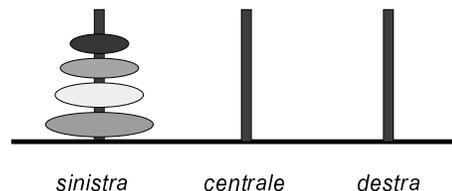
- da cui:

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

### ESERCIZIO: 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 portarli tutti sulla 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

### ESERCIZIO: la Torre di Hanoi



### ESERCIZIO: la Torre di Hanoi

Come risolvere il problema?

- Immaginare la serie di mosse che, *in generale*, risolve il problema è impensabile
- Invece, è abbastanza semplice esprimere una soluzione ricorsiva

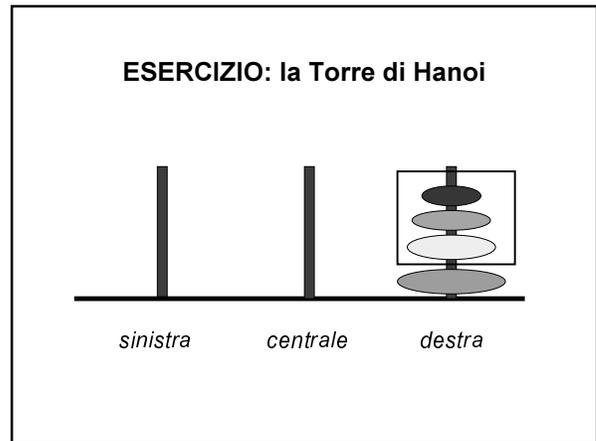
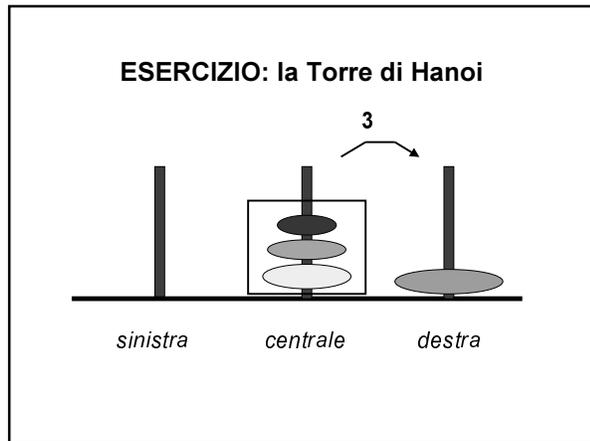
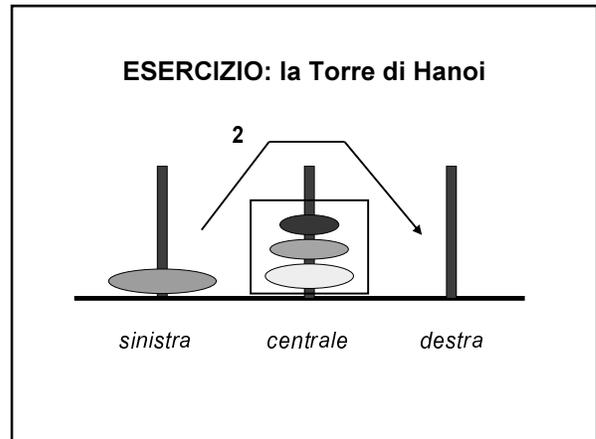
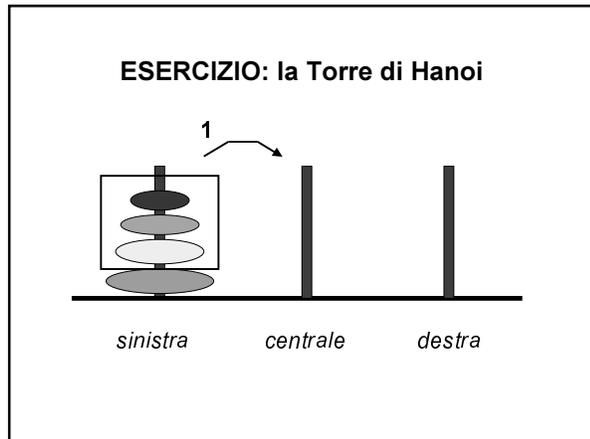
**Ipotesi:**

- Siamo capaci di spostare un singolo disco tra due torri a scelta

### ESERCIZIO: 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



**ESERCIZIO: 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 possediamo 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

**ESERCIZIO: 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 da spostare
  - la torre iniziale
  - la torre finale
  - la torre da usare come appoggio
- non ha tipo di ritorno, è una procedura → `void`

## ESERCIZIO: la Torre di Hanoi

### Interfaccia

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

### Codifica

```
void hanoi(int dischi, int iniziale,  
          int finale, int transito){  
    if (dischi==1) {  
        /* muovi da iniziale a finale */  
    } else {  
        hanoi(dischi-1, iniziale, transito, finale);  
        /* muovi da iniziale a finale */  
        hanoi(dischi-1, transito, finale, iniziale);  
    }  
}
```

## ESERCIZIO: la Torre di Hanoi

### Alcune domande

- come avremmo fatto senza la ricorsione?
- senza debugger, che cosa "computa" alla fine?

### Alcune proposte

- costruire un idoneo cliente e provare il programma
- provare, in un caso semplice, a seguire quello che accade passo per passo
- inserire al posto dei commenti delle istruzioni di stampa a video

## ESERCIZIO: la Torre di Hanoi

### Codifica

```
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);  
    }  
}
```