.

# Ricorsione

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

**Programmazione**

anno accademico 2016–2017

## Funzioni ricorsive

Chiamata è ciascun uso di una funzione nel `main` o in un'altra funzione

- la funzione è una (una dichiarazione e una definizione)
- le chiamate possono essere molte e sono strutturate ad albero

Funzione ricorsiva è una funzione che chiama sé stessa

Si noti che la funzione è la stessa, le chiamate sono diverse

```
int fattoriale (int n)
{
  if (n <= 1)
    return 1;
  else
    return n * fattoriale(n - 1);
}
```

.

## Elementi fondamentali

Una funzione ricorsiva deve contenere un blocco di selezione con

1. un caso base che calcola esplicitamente un risultato
2. un caso ricorsivo che "riduce" i dati e riapplica la funzione

```
int fattoriale (int n)
{
  if (n <= 1)                        /* selezione */
    return 1;                        /* 1. caso base */
  else
    return n * fattoriale(n - 1);    /* 2. caso ricorsivo */
}
```

I parametri attuali devono essere "più piccoli" dei parametri formali

## Elementi fondamentali

Una funzione ricorsiva deve contenere un blocco di selezione con

1. un caso base che calcola esplicitamente un risultato
2. un caso ricorsivo che "riduce" i dati e riapplica la funzione

```
int fattoriale (int n)
{
  if (n <= 1)                        /* selezione */
    return 1;                        /* 1. caso base */
  else
    return n * fattoriale(n - 1);    /* 2. caso ricorsivo */
}
```

I parametri attuali devono essere "più piccoli" dei parametri formali

.

## Elementi fondamentali

Una funzione ricorsiva deve contenere un blocco di selezione con

1. un caso base che calcola esplicitamente un risultato
2. un caso ricorsivo che "riduce" i dati e riapplica la funzione

```
int fattoriale (int n)
{
  if (n <= 1)                        /* selezione */
    return 1;                        /* 1. caso base */
  else
    return n * fattoriale(n - 1);    /* 2. caso ricorsivo */
}
```

I parametri attuali devono essere "più piccoli" dei parametri formali

## Esecuzione (1)

Conviene seguire sullo *stack* l'esecuzione della funzione ricorsiva

```
i = fattoriale(3);
```

0. Lo *stack* contiene solo le variabili locali del `main` (`i`);
   per valutare `i = fattoriale(3)`, si chiama la funzione
1. Prima chiamata
   - si alloca lo spazio per `n` e per il risultato di `fattoriale`
   - si valuta il parametro attuale (3) e si copia il valore in `n`
   - si esegue `if (n <= 1)`
     - `if (3 <= 1)`
   - si comincia a valutare `n * fattoriale(n-1)`
     - `3 * fattoriale(2)`
   - si valuta `fattoriale(n-1)` richiamando la funzione
     - `fattoriale(2)`

.

## Esecuzione (2)

2. Seconda chiamata
   - si alloca lo spazio per `n` e per il risultato di `fattoriale`
   - si valuta il parametro attuale (2) e si copia il valore in `n`
   - si esegue `if (n <= 1)`
     - `if (2 <= 1)`
   - si comincia a valutare `n * fattoriale(n-1)`
     - `2 * fattoriale(1)`
   - si valuta `fattoriale(n-1)` richiamando la funzione
     - `fattoriale(1)`
3. Terza chiamata
   - si alloca lo spazio per `n` e per il risultato di `fattoriale`
   - si valuta il parametro attuale (1) e si copia il valore in `n`
   - si esegue `if (n <= 1)`
     - `if (1 <= 1)`
   - si copia 1 nel risultato e si termina la terza chiamata deallocando lo spazio (il risultato è trasmesso al chiamante)

## Esecuzione (3)

2. Seconda chiamata
   - si termina la valutazione di `n * fattoriale(n-1)`, usando il risultato (1) come valore della funzione
     - `2 * 1`
   - si copia 2 nel risultato e si termina la seconda chiamata deallocando lo spazio (il risultato è trasmesso al chiamante)
1. Prima chiamata
   - si termina la valutazione di `n * fattoriale(n-1)`, usando il risultato (2) come valore della funzione
     - `3 * 2`
   - si copia 6 nel risultato e si termina la prima chiamata deallocando lo spazio (il risultato è trasmesso al chiamante)
0. Si termina la valutazione di `i = fattoriale(3)`, usando il risultato (6) come valore della funzione

.

## Caso base e riduzione

Il caso base corrisponde a dati per cui è facile trovare il risultato

$\Rightarrow$ si scrive direttamente del codice per trovarlo

Il caso ricorsivo corrisponde a dati per cui è difficile trovare il risultato, ma diventa facile conoscendo il risultato di problemi "più piccoli"

$\Rightarrow$ si costruiscono i dati di tali problemi e si richiama la funzione

Per funzionare, il metodo richiede

1. un algoritmo per il caso base
2. una procedura di "riduzione" dei dati
3. la garanzia che i dati ricadano prima o poi nel caso base
4. una procedura di "ricostruzione" del risultato

## Esempio 1: potenza

Calcolo della potenza $n$-esima di un numero intero $i$

- caso base ($n = 0$): la potenza è 1 per qualsiasi $i$

$$i^0 = 1 \qquad \forall i \in \mathbb{N}^+$$

- riduzione dei dati e ricostruzione del risultato: nota la potenza $(n-1)$-esima è facile trovare la potenza $n$-esima

$$i^n = i \cdot i^{n-1} \qquad \forall i \in \mathbb{N}^+, n \in \mathbb{N}^+$$

- garanzia di terminazione: $n$ cala di 1 ad ogni chiamata

$\Rightarrow$ arriva certamente a 0

.

## Esempio 2: ordinamento

Ordinamento di un vettore di numeri interi (*Selection Sort*)

- caso base (`n <= 1`): un vettore vuoto o di un solo elemento è già ordinato
- riduzione dei dati e ricostruzione del risultato: noto il vettore ordinato che contiene gli `(n-1)` elementi minimi, è banale accodarvi l'elemento massimo
- garanzia di terminazione: `n` cala di 1 ad ogni chiamata $\Rightarrow$ arriva certamente a 1

## Esempio 3: ancora ordinamento

Ordinamento di un vettore di numeri interi (*Insertion Sort*)

- caso base (`n <= 1`): un vettore vuoto o di un solo elemento è già ordinato
- riduzione dei dati e ricostruzione del risultato: noto un vettore ordinato che contiene `(n-1)` elementi qualsiasi, è facile inserirvi in modo ordinato l'elemento residuo
- garanzia di terminazione: `n` cala di 1 ad ogni chiamata $\Rightarrow$ arriva certamente a 1

.

## Ricorsione e iterazione

Ogni algoritmo ricorsivo ammette un algoritmo iterativo del tutto equivalente, cioè che fa le stesse operazioni nello stesso ordine

- la forma ricorsiva è spesso più sintetica ed elegante
- può esistere una forma iterativa più efficiente (calcoli non ripetuti)

Calcolo dei numeri di Fibonacci

```
if (n <= 1)
  return 1;
else
  return Fibonacci(n-1) +
         Fibonacci(n-2);
```

```
i = fpi = fi = 1;
while (i < n)
{
  i++;
  old_fi = fi;
  fi = fi + fpi;
  fpi = old_fi;
}
return fi;
```