# Corso di Programmazione

## Progettazione di programmi

Prof.ssa Teresa Roselli
teresa.roselli@uniba.it

# Tipi di Problemi

- Semplici: facile individuazione di algoritmi
  - Risolubili tramite
    - Un’azione primitiva o una sequenza di azioni primitive

- Complessi: difficilmente la soluzione è data pensando al problema nella sua interezza
  - Non risolubili tramite un’azione nota
    - Al solutore
    - All’esecutore

# Tipi di Problemi

- La progettazione di un algoritmo non è immediata, avviene alternando fasi di analisi e scelte realizzative.

- Da un'analisi generale del problema si passa ad una soluzione a grandi linee e, per gradi, si procede alla costruzione della soluzione finale specificando un numero sempre maggiore di dettagli.

# Problemi Complessi
## Soluzione

- Scomposizione in sottoproblemi
  - Fino a trovare un insieme di sottoproblemi primitivi che risulti equivalente al problema di partenza
- Individuazione del procedimento che porta alla soluzione
  - Processo di cooperazione tra sottoproblemi dei quali si è in grado di fornire facilmente una soluzione

# Problemi Complessi
## Approccio

- Principio del *Divide et Impera*
  - Ridurre la complessità del problema
    - Scomposizione in sottoproblemi più semplici
      - Individuare Struttura e Relazioni fra di essi
    - Progettazione di un algoritmo per ciascun sottoproblema
      - Se è complesso, riapplicare la scomposizione

    fino ad arrivare a problemi primitivi
    - Risolubili tramite azioni note

# Scomposizione di Problemi

- Tecnica per raffinamenti successivi

  (STEP-WISE REFINEMENT o TOP DOWN DESIGN)

  - Uno dei fondamenti della programmazione strutturata
  - Trasformazione di un problema in una *gerarchia di problemi* di difficoltà decrescente
    - Di un problema si affronta, prima, l'aspetto più generale e si passa, poi, a livelli sempre più dettagliati di descrizione sino ad arrivare agli elementi fondamentali

# Scomposizione di Problemi

- Scomporre il problema originario in sottoproblemi più semplici
- Affrontare un sottoproblema per volta

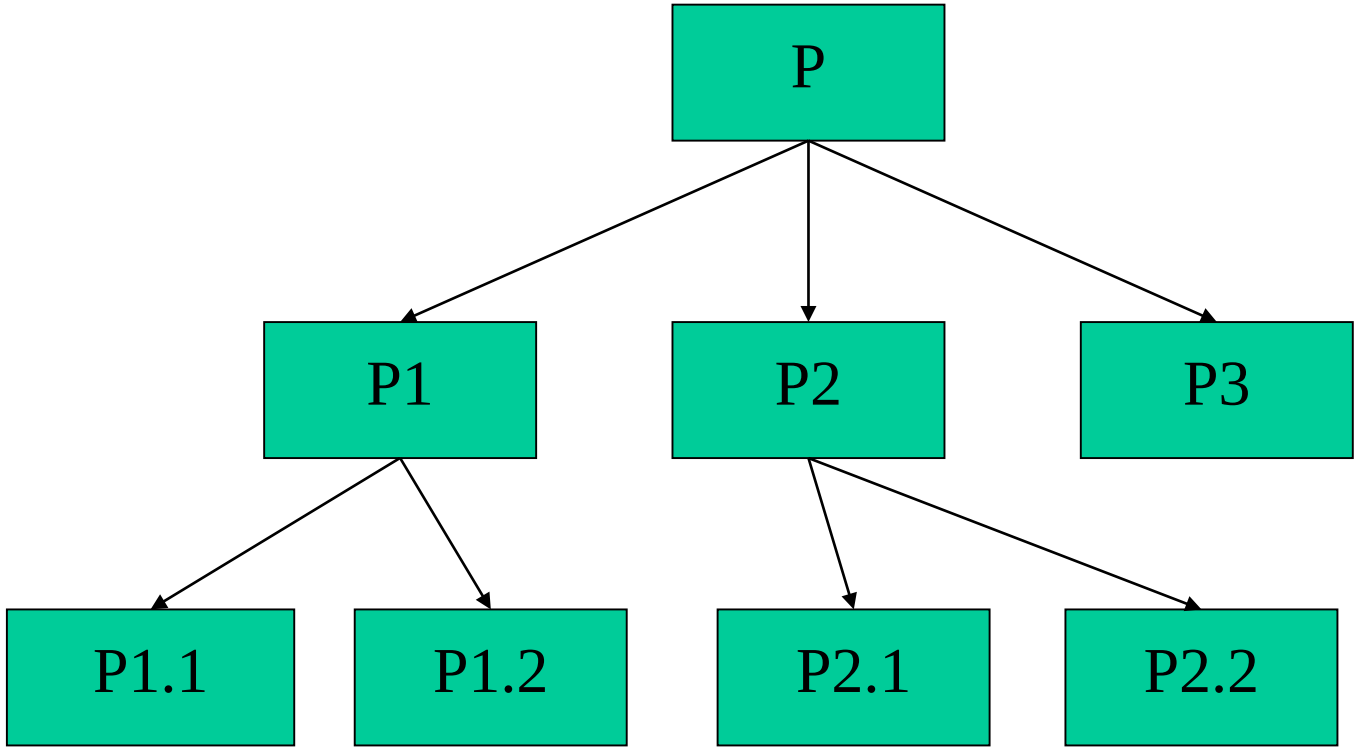
TOP — alto livello di descrizione

DOWN — basso livello di descrizione

# Scomposizione di Problemi

Nel processo di raffinamento inizialmente l'attenzione è rivolta a *cosa* poi, man mano, raffinando si passa a *come* ovvero ad un algoritmo che indichi *come fare per ottenere cosa*.

# Scomposizione di Problemi
## Albero dello sviluppo TOP-DOWN




I scomposizione

II scomposizione

Ad ogni passo di scomposizione ci si allontana dal linguaggio naturale (ad alto livello) e ci si avvicina alla descrizione nel linguaggio di programmazione: uso del linguaggio lineare

# Scomposizione di Problemi

- Conseguenze
  - Aumento del numero problemi
  - Diminuzione della complessità di ciascuno
- Può capitare che il medesimo sottoproblema debba venire risolto più volte, applicandolo a dati diversi, per risolvere il problema complessivo

# Scomposizione di Problemi
## Requisiti

- Ogni passo della suddivisione o specificazione deve garantire che
  - La soluzione dei sottoproblemi conduca alla soluzione generale
  - La successione di passi da eseguire abbia senso e sia possibile
  - La suddivisione dia luogo a sottoproblemi “più vicini” agli strumenti disponibili
    - Risolubili direttamente con gli operatori a disposizione

# Scomposizione di Problemi
## Quando fermarsi?

- Bisogna arrivare al punto in cui
  - Tutti i problemi sono primitivi
  - È fissato l'ordine di soluzione dei sottoproblemi
  - È previsto il modo in cui un sottoproblema utilizza i risultati prodotti dalla soluzione dei sottoproblemi che lo precedono
    - Livello di cooperazione

# Scomposizione di Problemi

- I problemi ottenuti dalla scomposizione sono
  - Indipendenti
    - Si può progettare un algoritmo per ciascuno
    - Soluzione delegabile a solutori diversi (in parallelo)
  - Cooperanti
    - Ciascuno può usare il risultato della soluzione di altri
- Quanto più complesso è il problema, tanti più livelli saranno necessari in profondità
  - Di solito 2 o 3
- Uno stesso problema può essere scomposto in modi diversi

# Scomposizione di Problemi

- I problemi ottenuti dalla scomposizione sono
  - Indipendenti
    - Si può progettare un algoritmo per ciascuno
    - Soluzione delegabile a solutori diversi (in parallelo)
  - Cooperanti
    - Ciascuno può usare il risultato della soluzione di altri
- Quanto più complesso è il problema, tanti più livelli saranno necessari in profondità
  - Di solito 2 o 3
- Uno stesso problema può essere scomposto in modi diversi

# Progettazione

## Esempio della ricerca in una agendina telefonica

- Scomposizione del problema in sottoproblemi più semplici tali che l'insieme dei sottoproblemi sia equivalente al problema di partenza

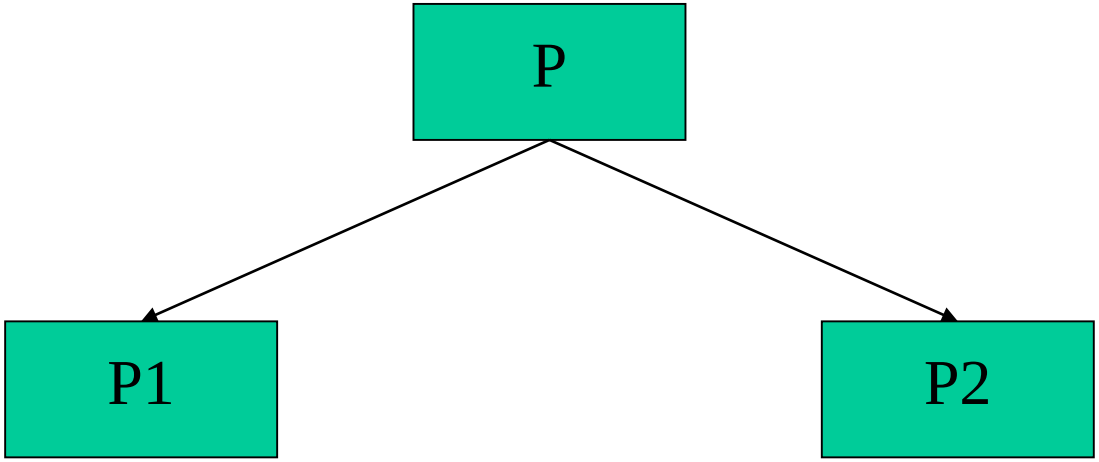
  *P1. Trovare la pagina dell'agendina in cui si dovrebbe trovare (se c'è) il cognome dato*

  *P2. Ricercare il cognome dato nella pagina trovata*

- Risoluzione dei sottoproblemi individuati
  - Per quanto possa essere chiara la *formulazione* di un problema, essa non fornisce, in genere, un *metodo* che consenta di ottenere il risultato partendo dai *dati iniziali*!

# Progettazione
## Esempio

Albero dello sviluppo TOP-DOWN



# Progettazione
## Esempio

- Sottoproblema 1
  - Si tratta di un problema di ricerca in un insieme di pagine ordinato alfabeticamente
  - L'insieme delle pagine che costituiscono l'agendina è piccolo
  - La scansione è sequenziale
    - Si sfogliano tutte le pagine, una dopo l'altra, a partire dalla prima
  - È una ricerca di sicuro successo
    - La pagina cercata esiste sicuramente nell'agendina

# Progettazione
## Esempio

- *Come* controlliamo se la pagina attuale è proprio la pagina cercata?
    - In tal caso avremmo raggiunto il nostro obiettivo (termine della ricerca)
- *Sfogliare una pagina alla volta, iniziando dalla prima pagina, fino a che la pagina attuale contiene nell'intestazione la lettera iniziale del cognome cercato*
  - Confrontare la lettera iniziale del cognome con la lettera o il gruppo di lettere contenute nell'intestazione della pagina attuale

# Progettazione
## Esempio

- Individuazione della pagina in cui cercare il cognome cercato
  - A B C
  - D E F ← se il cognome è DANGELO,
  - G H I J termina la scansione
  - K L M dell'agendina alla ricerca della
  - N O P pagina
  - Q R S
  - T U V
  - W X Y Z

# Progettazione
## Esempio

- Entità da rappresentare
  - Sequenza di pagine
    - 2 componenti
      - Intestazione
      - Insieme di righi
  - Ogni pagina ha un insieme di righi
    - Tutti con la stessa struttura
  - Ogni rigo ha le stesse suddivisioni
    - Ciascuna destinata ad un'informazione diversa
      - Cognome
      - Numero telefonico

# Progettazione
## Esempio

- Sottoproblema 2
  - Si tratta di un problema di ricerca in un insieme di cognomi non ordinati
  - La ricerca può non avere successo
  - La ricerca è di tipo sequenziale (come per il sottoproblema 1)
    - Si parte dal primo cognome presente sulla pagina e si passa al cognome immediatamente successivo

# Progettazione
## Esempio

- La scansione dei cognomi nella pagina ha termine quando:
    - si è trovato un cognome uguale a quello cercato
    - il numero riportato accanto è il risultato

  oppure

  quando:
    - si sono scanditi tutti i cognomi senza trovare quello cercato
    - il risultato è “non presente”

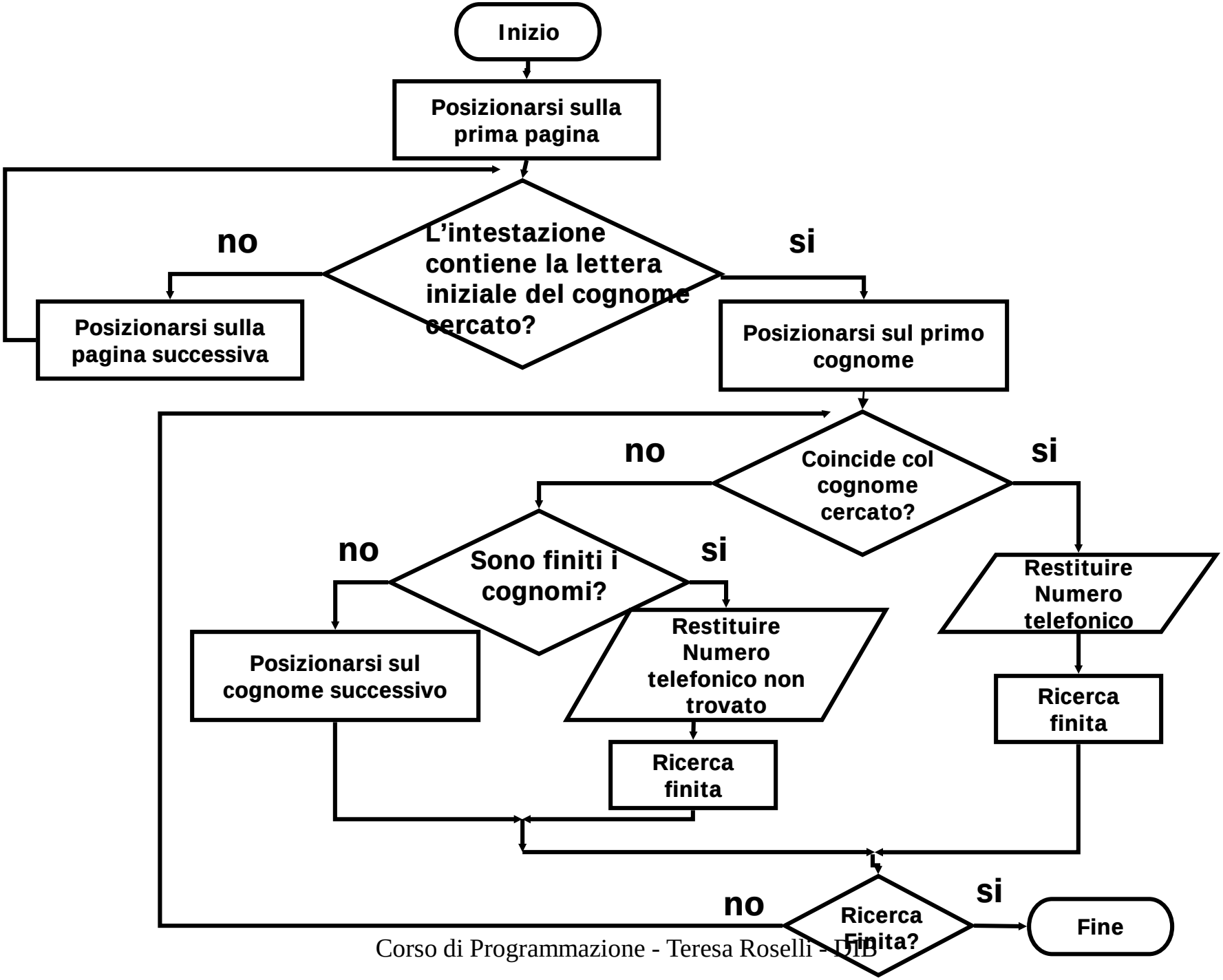
# Progettazione
## Esempio

- Sfogliare una pagina alla volta, iniziando dalla prima pagina, fino a che la pagina attuale contiene nell'intestazione la lettera iniziale del cognome cercato
- Scandire sequenzialmente un cognome alla volta, iniziando dal primo, fino a che il cognome attuale coincide con il cognome cercato
  - in tal caso comunicare il numero riportato accanto

  oppure non ci sono più cognomi
  - in tal caso comunicare "non presente"

# Progettazione

- Elementi a disposizione
  - Dati in ingresso
  - Dati in uscita
  - Relazioni esistenti fra essi
- Prodotto
  - Rappresentazione delle entità del problema
  - Suddivisione in sottoproblemi
  - Strategia per passare dalle entità in ingresso a quelle in uscita


Inizio
Posizionarsi sulla prima pagina
L'intestazione contiene la lettera iniziale del cognome cercato?
no
si
Posizionarsi sulla pagina successiva
Posizionarsi sul primo cognome
Coincide col cognome cercato?
no
si
Sono finiti i cognomi?
no
si
Posizionarsi sul cognome successivo
Restituire Numero telefonico non trovato
Ricerca finita
Restituire Numero telefonico
Ricerca finita
Ricerca Finita?
no
si
Fine

# *S*-composizione di Problemi
## sviluppo BOTTOM-UP

Metodo opposto al Top-Down (dal basso verso l'alto)

Si parte dalle azioni primitive e costruendo da queste algoritmi semplici si collegano tra loro per ottenere algoritmi sempre più complessi sino ad arrivare all'algoritmo finale che costituisce la soluzione completa del problema.

Nella costruzione di un nuovo algoritmo ⟶ Top-Down

Nell'adattamento per scopi diversi di programmi già scritti ⟶ Bottom-Up

Sia la scomposizione che la composizione dà luogo a costruzioni

ben strutturate ⟶ necessità di linguaggi che consentano l'articolazione in sottoproblemi

# Scomposizione di Problemi
## Strumenti

- I costrutti offerti dai linguaggi di programmazione strutturata supportano le scomposizioni
  - Sequenziale
    - Suddividere un problema in parti disgiunte
  - Selettiva
    - Trattamento differenziato in casi differenti
  - Iterativa
  - Ricorsiva

# Scomposizione Sequenziale

- La soluzione si ottiene tramite una sequenza di passi
  - I passi sono eseguiti uno alla volta
  - Ogni passo è eseguito una sola volta
    - Nessuno è ripetuto o omesso
  - L'ordine in cui i passi vanno eseguiti è lo stesso in cui sono scritti
  - L'ultimo passo equivale alla terminazione del procedimento

# Scomposizione Sequenziale
## Esempio

- Problema: Preparare una tazza di tè
- Soluzione: l'algoritmo per la tazza di tè
  1. Bollire l'acqua
  2. Mettere il tè nella tazza
  3. Versare l'acqua nella tazza
  4. Lasciare in infusione per 3 minuti
- Non sono problemi primitivi
  - Ciascuno va ulteriormente scomposto

# Scomposizione Sequenziale
## Esempio

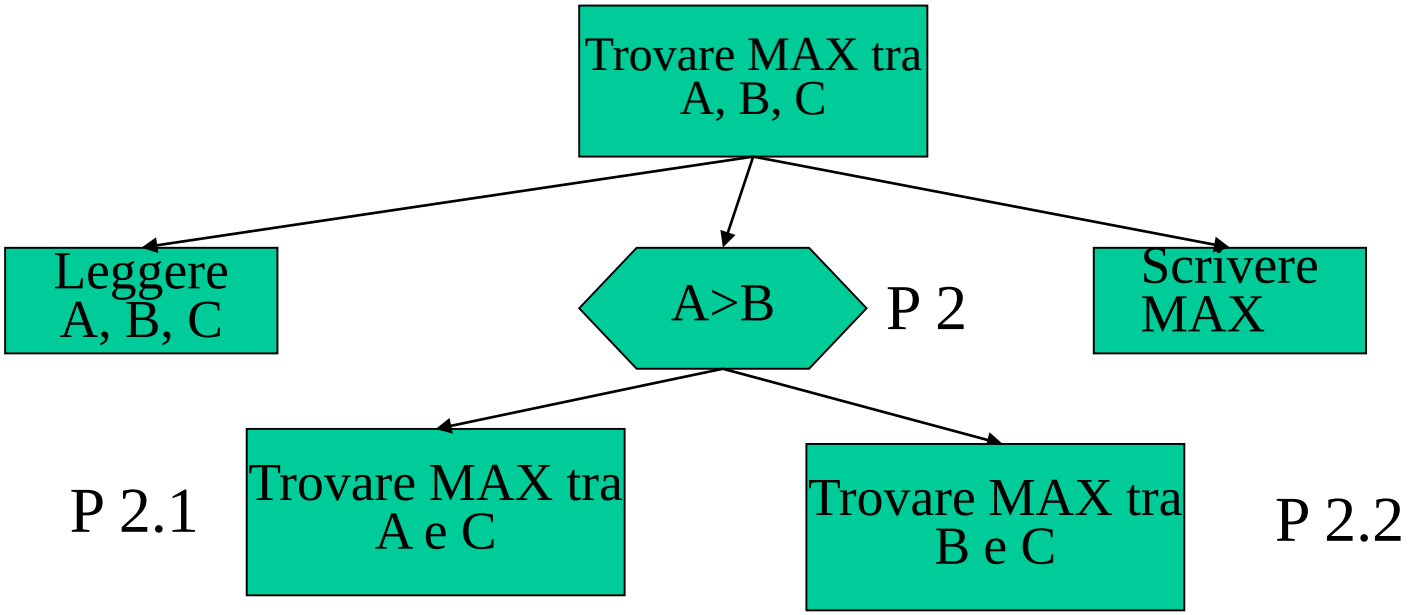
1.
   - 1.1 Riempire d’acqua il bollitore
   - 1.2 Accendere il gas
   - 1.3 Aspettare che l’acqua bolla
   - 1.4 Spegnere il gas
2.
   - 2.1 Aprire la scatola del tè
   - 2.2 Prendere un sacchetto-filtro
   - 2.3 Chiudere la scatola del tè
   - 2.4 Mettere il sacchetto nella tazza
3.
   - 3.1 Versare l’acqua bollente nella tazza finché non è piena
4.
   - 4.1 Aspettare 3 minuti
   - 4.2 Estrarre il sacchetto-filtro

# Scomposizione Selettiva

- Soluzione ottenuta tramite scelte multiple
  - Strutture di selezione all'interno di altre strutture di selezione
- La scomposizione fa ricorso a strutture multiple
  - Nidificazione (o *annidamento*) di strutture

# Scomposizione Selettiva
## Esempio

- Problema: Trovare il più grande dei numeri *a*, *b*, *c* diversi tra loro
  - Azione primitiva (nota):
    - Verificare se un numero è maggiore di un altro



# Scomposizione Selettiva
## Esempio

- **se** $a > b$ *P2*
  - **allora**: verifica **se** $a > c$ *P2.1*
    - **allora**: $a$ è la soluzione P2.1.1
    - **altrimenti**: $c$ è la soluzione P2.1.2
  - **altrimenti**: verifica **se** $b > c$ *P2.2*
    - **allora**: $b$ è la soluzione P2.2.1
    - **altrimenti**: $c$ è la soluzione P2.2.2

# Scomposizione Selettiva
## Esempio

### Struttura CASE

Esempio di applicazione

In un ospedale lo schedario di cartelle cliniche contiene per ogni cartella

- codice numerico del paziente a cui si riferisce;
- dati anagrafici;
- Informazioni mediche

Si suppone che lo schedario sia ordinato secondo il codice numerico crescente.

# Scomposizione Selettiva
## Esempio

Le operazioni possibili sullo schedario sono:

- eliminazione della cartella di un paziente dimesso
- aggiornamento di una cartella con nuove informazioni
- inserimento di nuove cartelle per nuovi pazienti

Gli aggiornamenti vengono effettuati da un addetto che riceve un insieme di informazioni così organizzate:

- Si tratta di una successione di comandi
- Il comando può essere
  - Inserimento I sulla scheda seguito dal codice dopo il quale inserire
  - Eliminazione E seguito dal codice della scheda da eliminare
  - Aggiornamento A seguito dal codice della scheda da aggiornare

Si usa l'asterisco * per distinguere un comando dall'altro

# Scomposizione Selettiva
## Esempio

Esempio:

```
* I  117
       informazioni su paziente 118
* I  118
       informazioni su paziente 119
* E  120
* I   122
       informazioni su paziente 123
* A  125
       informazioni aggiuntive per
       paziente 125
   ..............
```

# Scomposizione Selettiva
## Esempio

Uso della struttura CASE per l'esecuzione di un singolo comando (scelta multipla)

```
If  c'è "*"  then
            case    lettera
                    I: effettua inserimento
                    E: effettua eliminazione
                    A: effettua aggiornamento
                    else
                    segnala comando errato
            endcase
        else
            segnala assenza scheda comando
endif
```

# Scomposizione Selettiva
## Verso l'Iterazione

- Altro modo per scomporre il problema:

  Trovare il più grande dei numeri *a*, *b*, *c* diversi tra loro
  - Trova il più grande fra *a* e *b*
  - Trova il più grande tra la soluzione del passo precedente e *c*

- Generalizzazione del problema precedente
  - Necessità di definire una *struttura di dati*

# Scomposizione Iterativa

- Successione di problemi tutti dello stesso tipo
  - Ripetere un numero finito di volte un passo di soluzione in modo da avere, alla fine, la soluzione completa

# Scomposizione Iterativa

- Quando durante la scomposizione di un problema la soluzione porta all'individuazione di una catena di sottoproblemi che soddisfano le seguenti condizioni:
  - I sottoproblemi sono uguali, oppure differiscono solo per i dati su cui agiscono
  - I dati su cui agiscono i sottoproblemi sono in relazione d'ordine e un sottoproblema differisce dal precedente solo perchè utilizza il dato successivo.

allora si dice che il problema è risolto ITERATIVAMENTE

# Scomposizione Iterativa
## Esempio

- Trovare il più grande fra $n > 3$ numeri tutti diversi fra loro
  - Supponiamo che i dati siano in relazione d'ordine
    - Si può parlare di 1° dato, 2° dato, … n° dato

# Scomposizione Iterativa
## Esempio

- Trova il più grande tra i primi 2 numeri
- Trova il più grande fra il risultato precedente e il 3° numero
- …
- Trova il più grande fra il risultato precedente e l'ultimo numero (*n*° dato)

- Più concisamente:
  - Trova il più grande fra i primi 2 numeri
  - **Mentre** ci sono numeri da esaminare **esegui**
    - Esamina il primo numero non ancora considerato
    - Trova il più grande tra questo e il più grande precedentemente trovato

# Esempio

Si dispone del livello delle precipitazioni relativo a tutti i giorni di un periodo di 4 settimane . Si vuole determinare il livello totale di precipitazioni per settimana e per tutto il periodo e il giorno di massima precipitazione per settimana.

# Esempio

## Dati di Input:

- Numero delle settimane
- Per ogni settimana:
  - Data
  - Precipitazione espressa in mm

## Output:

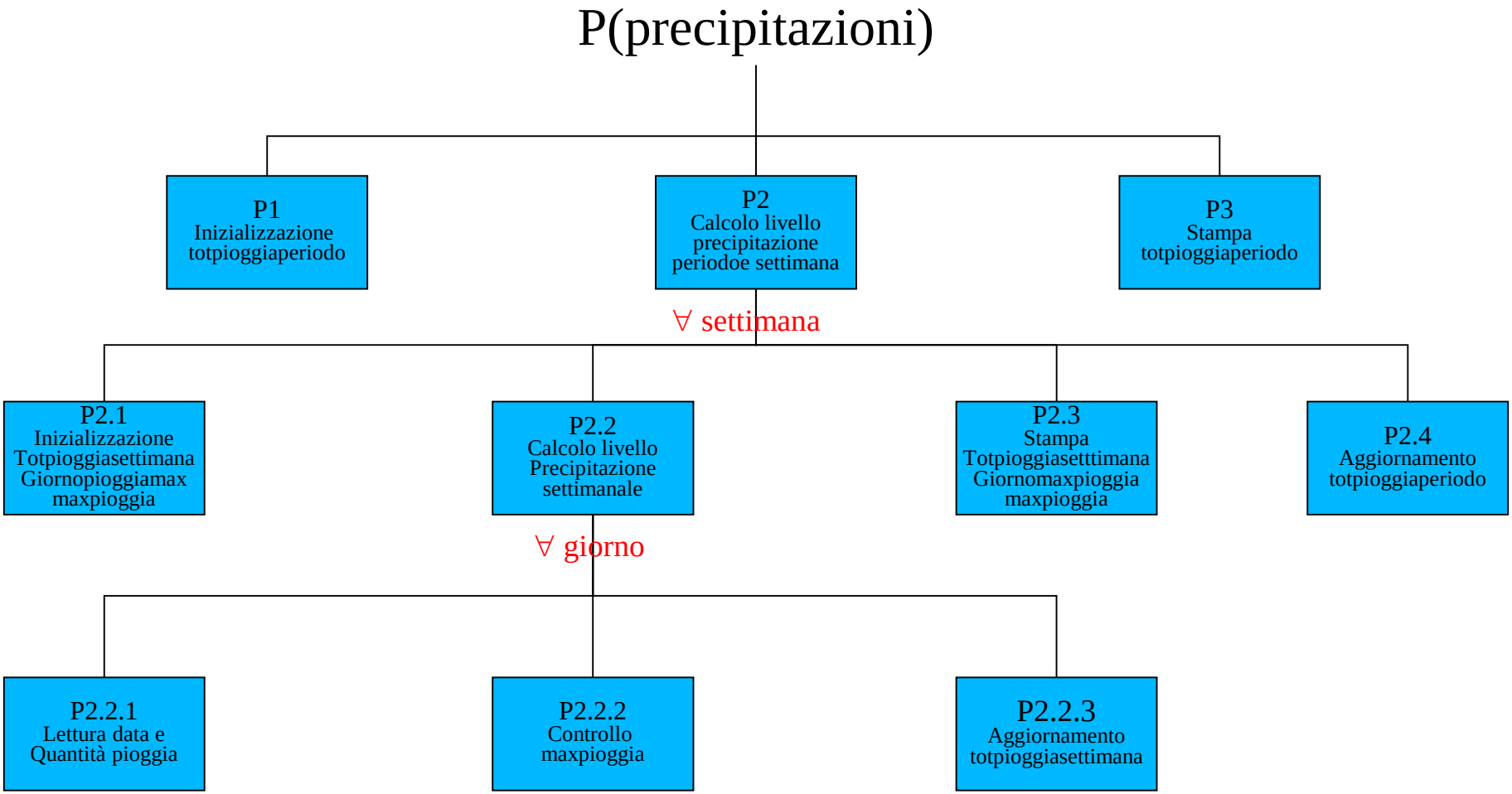
- Livello totale di precipitazione
- Per ogni settimana:
  - Livello totale di precipitazione
  - Data di massima precipitazione

# Esempio

## Metodo di soluzione

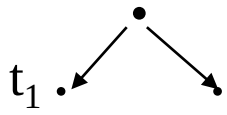
- Per calcolare il livello totale di precipitazione di n settimane occorre sommare i livelli di precipitazione di ogni settimana.
  - Per calcolare il livello di precipitazione di ogni settimana occorre sommare i livelli di precipitazione di ogni giorno della settimana.
    - Per determinare il giorno di massima precipitazione di ogni settimana occorre confrontare i livelli di precipitazione di tutti i giorni della settimana

# Esempio


P(precipitazioni)
P1
Inizializzazione
totpioggiaperiodo
P2
Calcolo livello
precipitazione
periodoe settimana
P3
Stampa
totpioggiaperiodo
∀ settimana
P2.1
Inizializzazione
Totpioggiasettimana
Giornopioggiamax
maxpioggia
P2.2
Calcolo livello
Precipitazione
settimanale
P2.3
Stampa
Totpioggiasetttimana
Giornomaxpioggia
maxpioggia
P2.4
Aggiornamento
totpioggiaperiodo
∀ giorno
P2.2.1
Lettura data e
Quantità pioggia
P2.2.2
Controllo
maxpioggia
P2.2.3
Aggiornamento
totpioggiasettimana

# Oggetto Ricorsivo

- Definito in termini di se stesso
  - Esempi
    - Numeri naturali:
      - 1 è un numero naturale
      - Il successore di un numero naturale è un numero naturale
    - Strutture ad albero:
      - • è un albero (*albero vuoto*)
      - Se $t_1$ e $t_2$ sono alberi, allora anche la struttura




è un albero

# Definizione Ricorsiva

- Ottenuta in termini di versioni più semplici della stessa cosa che definisce
- 2 livelli:
  - Passo
    - Operazione che riconduce la definizione ad un dato livello a quella di livello inferiore
  - Base (o *livello assiomatico*)
    - Definizione di partenza
      - Necessario per ricostruire i livelli successivi
      - Consente di definire problemi di ordine superiore

# Scomposizione Ricorsiva

- Un problema di ordine *n* è definito in termini del medesimo problema di ordine inferiore
  - Entità definita in termini più semplici della stessa entità
  - Nella definizione deve apparire il livello assiomatico (o *di ricostruzione*)
    - Problema risolubile tramite un'azione primitiva

# Scomposizione Ricorsiva
## Requisiti

- Almeno uno dei sottoproblemi è formalmente uguale al problema di partenza, ma di ordine inferiore
- Al momento della scomposizione è noto l'ordine del problema
  - Scomposizione attuabile mediante *regola di copiatura*
    - Sostituire ad ogni occorrenza del problema la scomposizione ricorsiva

    fino ad avere problemi primitivi (raggiungere il livello assiomatico)

# Scomposizione Ricorsiva
## Esempio

- Trovare un metodo per *invertire* una sequenza di lettere
  - **se** la sequenza contiene una sola lettera **allora**
    - Scrivila (il problema è risolto)
  - **altrimenti**:
    - Rimuovi la prima lettera dalla sequenza
    - *Inverti* la sequenza rimanente
    - Appendi in coda la lettera rimossa

# Scomposizione Ricorsiva
## Esempio

- Invertire “roma”
  - “roma” (4 lettere): rimuovi “r”, inverti “oma”
    - “oma” (3 lettere): rimuovi “o”, inverti “ma”
      - “ma” (2 lettere): rimuovi “m”, inverti “a”
        - » “a” (1 lettera): è già invertita
      - Appendi “m”: “am”
    - Appendi “o”: “amo”
  - Appendi “r”: “amor”

# Scomposizione Ricorsiva
## Caratteristiche

- Un problema espresso ricorsivamente termina sempre
- Un problema esprimibile ricorsivamente si può risolvere iterativamente
- Una funzione esprimibile ricorsivamente è computabile
  - Esiste un algoritmo che la calcola e termina sempre
- Ogni funzione computabile per mezzo di un programma è ricorsiva

# Iterazione e Ricorsione
## Esempio

- Calcolo del prodotto di due interi *n* e *m*
  - Definizione iterativa:
    - $n \times m = m + m + \ldots + m$ (*n* volte)
  - Definizione ricorsiva:
    - $$n \times m = \begin{cases} 0 & \text{se } n = 0 \\ (n - 1) \times m + m & \text{se } n > 0 \end{cases}$$

# Iterazione e Ricorsione
## Esempio

- Soluzione iterativa

inizialmente sia il risultato uguale a 0

**ripeti** per ogni intero da 1 ad *n*

somma *m* al risultato per ottenere un nuovo risultato

il prodotto è il risultato finale

- Soluzione ricorsiva

**se** *n* è uguale a 0

**allora** il prodotto è 0

**altrimenti**

calcola il prodotto di $n - 1$ per $m$

somma *m* al prodotto

# Scomposizione di Problemi
## Conclusioni

- Può esserci più di una scomposizione di un problema in sottoproblemi
- Cause di difficoltà nella scomposizione
  - Comprensione intuitiva della complessità di un problema
  - Scelta tra le possibili scomposizioni
  - Necessità di una formulazione “adeguata” per ciascun sottoproblema

# Scomposizione di Problemi
## Livelli di Complessità

- Corrispondenti ai metodi di scomposizione
  - Formula
  - Sequenza di formule
    - Risultati intermedi
  - Algoritmi condizionali
  - Algoritmi iterativi
  - Algoritmi ricorsivi