Introduzione al C

IL PROBLEMA DEL PROGETTO

- **La descrizione del problema, in genere, non indica direttamente il modo per ottenere il risultato voluto (il procedimento risolutivo)**
- **Occorrono *metodologie* per affrontare il problema del progetto in modo sistematico**

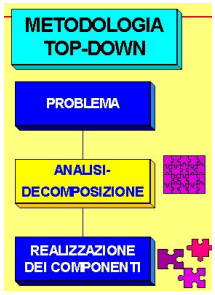
IL PROBLEMA DEL PROGETTO

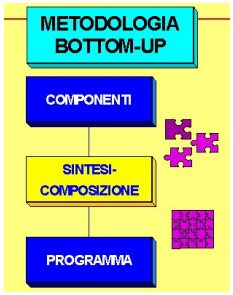
- **Due dimensioni progettuali:**
  - **Programmazione in piccolo (*in-the-small*)**
  - **Programmazione in grande (*in-the-large*)**

Principi cardine:

- **procedere per livelli di astrazione**
- **garantire al programma *strutturazione* e *modularità***

METODOLOGIE DI PROGETTO

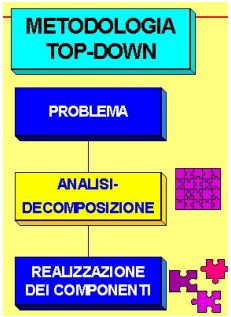




METODOLOGIA TOP-DOWN

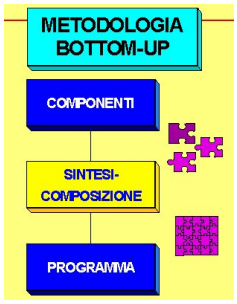
Procede per **decomposizione** del problema in sotto-problemi, per ***passi di raffinamento successivi***

- Si scompone il problema in sottoproblemi
- Si risolve ciascun sottoproblema con lo stesso metodo, fino a giungere a sottoproblemi risolubili con mosse elementari



METODOLOGIA BOTTOM-UP

Procede per ***composizione di componenti e funzionalità elementari***, fino alla **sintesi** dell'intero algoritmo ("*dal dettaglio all'astratto*")



IL PROBLEMA DEL PROGETTO

Dunque, dato un problema ***non si deve iniziare subito a scrivere il programma.***

- così si scrivono *a fatica* programmi semplici
- spesso sono errati, e non si sa perché
- nessuno capisce cosa è stato fatto (dopo un po', nemmeno l'autore...)
- è necessario valutare la soluzione migliore tra tante
- è necessario scrivere programmi facilmente modificabili/estendibili

7

IL PROBLEMA DEL PROGETTO

- **La *specifica* della *soluzione* e la fase di *codifica* sono concettualmente distinte**
- **e tali devono restare anche in pratica!**

8

UN ESEMPIO

Problema:

"Data una temperatura espressa in gradi Celsius, calcolare il corrispondente valore espresso in gradi Fahrenheit"

Approccio:

- **si parte dal problema e dalle proprietà note *sul dominio dei dati***

9

UN ESEMPIO

Problema:

"Data una temperatura espressa in gradi Celsius, calcolare il corrispondente valore espresso in gradi Fahrenheit"

Specifica della soluzione:

Relazioni tra grandezze* esistenti nello specifico *dominio applicativo:

$$c * 9/5 = f - 32$$

10

UN ESEMPIO

L'Algoritmo corrispondente:

- Dato **c**
- calcolare **f** sfruttando la relazione

$$f = 32 + c * 9/5$$

SOLO A QUESTO PUNTO

- **si sceglie un linguaggio**
- **si *codifica* l'algoritmo in tale linguaggio**

11

IL LINGUAGGIO C

UN PO' DI STORIA

- definito nel 1972 da Dennis Ritchie (AT&T Bell Labs) per sostituire l'Assembler
- prima definizione precisa: Kernighan & Ritchie (1978)
- prima definizione ufficiale: ANSI (1983)

12

CARATTERISTICHE

- linguaggio *sequenziale*
  - le istruzioni sono eseguite in sequenza, non in parallelo (non si possono eseguire due istruzioni contemporaneamente)
- *imperativo*
  - le istruzioni sono tipicamente dei comandi dati all'imperativo. Assegnamento distruttivo
- *strutturato* a blocchi
  - ogni istruzione o costrutto ha sempre un punto di ingresso ed uno di uscita
- usabile anche come linguaggio di sistema
  - adatto a software di base, sistemi operativi, compilatori, ecc.
- portabile, efficiente, sintetico
  - ma a volte poco leggibile...

IL LINGUAGGIO C

Basato su pochi *concetti elementari*

- dati (tipi primitivi, tipi di dato)
- espressioni
- dichiarazioni / definizioni
- funzioni
- istruzioni / blocchi

STRUTTURA DI UN PROGRAMMA C

- **In prima battuta, la struttura di un programma C è definita nel modo seguente:**

```
<programma> ::=
  {<unità-di-traduzione>}
  <main>
  {<unità-di-traduzione>}
```

– *Intuitivamente un programma in C è definito da tre parti:*
  - *zero o più unità di traduzione,*
  - *il programma vero e proprio (main)*
  - *zero o più unità di traduzione*

STRUTTURA DI UN PROGRAMMA C

- **La parte <main> è l'unica *obbligatoria,* ed è definita come segue:**

```
<main> ::= main() <blocco>
<blocco> ::=
  {[<dichiarazioni-e-definizioni>]
   [<sequenza-istruzioni>]
  }
```

•Intuitivamente il <main> è definito dalla parola chiave `main()` *e racchiuso tra parentesi graffe al cui interno troviamo*

•le dichiarazioni e definizioni
•una sequenza di istruzioni
} *opzionali* `[ ]`

STRUTTURA DI UN PROGRAMMA C

- `<dichiarazioni-e-definizioni>`
  introducono i nomi di costanti, variabili, tipi definiti dall'utente
- `<sequenza-istruzioni>`
  sequenza di frasi del linguaggio ognuna delle quali è un'istruzione

Il `main()` è *una particolare unità di traduzione* (una *funzione*).

COMMENTI

- I **commenti** servono per inserire un testo all'interno del programma. Sono spiegazioni che vengono date a chi legge il programma. Vengono scartati dal compilatore (non viene prodotto codice eseguibile per i commenti). E` importante mettere commenti nel programma per spiegare che cosa fa (altrimenti anche l'autore dopo un po' non capisce più che cosa fa il programma).
- **Sintassi:** sequenze di caratteri racchiuse fra i delimitatori `/*` e `*/` oppure da `//` fino alla fine della riga

```
<Commento> ::= /* <frase> */
    | // <frase> \n
<frase>  ::= {<carattere> }
```

- i commenti non possono essere innestati.

La prima istruzione

- Sintassi semplificata:

```
printf("testo");
```

visualizza il testo riportato fra virgolette

- il punto e virgola serve per terminare l'istruzione: in una sequenza di istruzioni, il C capisce dove ne finisce una e comincia la successiva grazie al punto e virgola
- **`printf`** non è un'istruzione predefinita del C, ma è definita nella libreria **`stdio`**. Come vedremo, per usare le librerie si utilizza la direttiva **`#include`**.

```
#include <stdio.h>
```

che va inserita all'inizio del programma

- Si possono usare i caratteri di controllo:

newline	**\n**	**tab**	**\t**
backspace	**\b**		
carriage return	**\r**		

19

Il primo programma C

```
#include <stdio.h>
/* programma che visualizza la
  scritta "hello world" */

main()
{
    printf("hello world\n");
}
```

Indentazione: per far capire che **`printf`** è dentro al **`main`**, lasciamo un po' di spazio (allineiamo le cose che sono allo stesso livello)

20