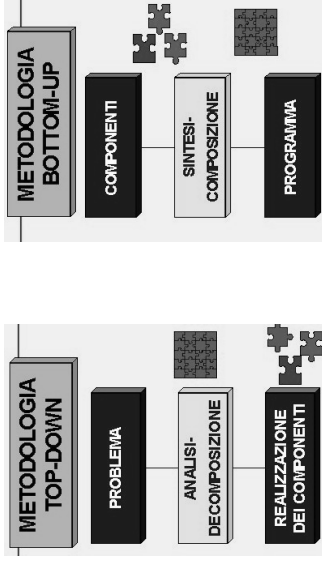


IL PROBLEMA DEL PROGETTO

- 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

METODOLOGIE DI PROGETTO



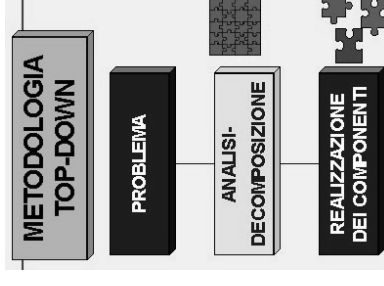
IL PROBLEMA DEL PROGETTO

- Due dimensioni progettuali:
 - Programmazione in piccolo (*in-the-small*)
 - Programmazione in grande (*in-the-large*)
- procedere per livelli di astrazione
- garantire al programma *strutturazione e modularità*

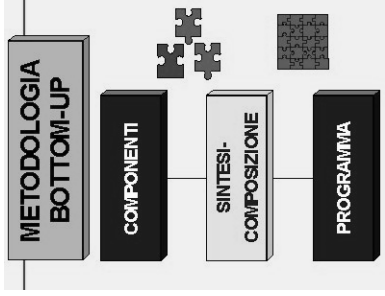
METODOLOGIA TOP-DOWN

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 risolvibili 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

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

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...)
- e' necessario valutare la soluzione migliore tra tante
- e' necessario scrivere programmi facilmente modificabili/estendibili

UN ESEMPIO

Problema:

"-ata 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**

UN ESEMPIO

Problema:

“+ata una temperatura espressa in gradi Celsius, calcolare il corrispondente valore espresso in gradi Fahrenheit”

Specifica della soluzione:

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

$$c = (f - 32) * 5/9 \text{ oppure } f = 32 + c * 9/5$$

LINGUAGGI DI PROGRAMMAZIONE

Il “**potere espressivo**” di un linguaggio è caratterizzato da:

- **quali tipi di dati** consente di rappresentare (direttamente o tramite definizione dell'utente)
- **quali istruzioni di controllo** mette a disposizione (quali operazioni e in quale ordine di esecuzione)

PROGRAMMA = DATI + CONTROLLO

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**

IL LINGUAGGIO C

UN PO' DI STORIA

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

IL LINGUAGGIO C

CARATTERISTICHE

- linguaggio *sequenziale, imperativo, strutturato* a blocchi
- usabile anche come linguaggio di sistema
 - adatto a software di base, sistemi operativi, compilatori, ecc.
- portabile, efficiente, sintetico
 - ma a volte poco leggibile...

ESEMPIO: Un semplice programma

- Codifica in linguaggio C dell'algoritmo che converte gradi Celsius in Fahrenheit

```
main(){
float c, f; /* Celsius e Fahrenheit */
printf("\nInserisci la temperatura da convertire");
scanf("%f", &c);
f = 32 + c * 9/5;
printf("Temperatura Fahrenheit %f", f);
}
```

IL LINGUAGGIO C

Basato su pochi concetti elementari

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

IL LINGUAGGIO C

- Un elaboratore è un manipolatore di *simboli (segni)*
- L'architettura fisica di ogni elaboratore è *intrinsecamente capace di trattare vari domini di dati, detti tipi primitivi*
 - dominio dei *numeri naturali e interi*
 - dominio dei *numeri reali* (con qualche approssimazione)
 - dominio dei *caratteri*
 - dominio delle *stringhe di caratteri*

TIPI DI DATO

Il concetto di *tipo di dato* viene introdotto per raggiungere due obiettivi:

- esprimere in modo sintetico
 - la loro rappresentazione in memoria, e
 - un insieme di operazioni ammissibili
- permettere di *effettuare controlli statici* (al momento della compilazione) sulla *correttezza* del programma.

TIPI DI DATO PRIMITIVI IN C

- **reali**
 - `float` singola precisione (32 bit)
 - `double` doppia precisione (64 bit)
- **boolean**
 - *non esistono in C come tipo a sé stante*
 - si usano gli interi:
 - `zero` indica **FALSO**
 - ogni altro valore indica **VERO**
 - convenzione: suggerito utilizzare **uno per VERO**

TIPI DI DATO PRIMITIVI IN C

- **caratteri**
 - `char` caratteri ASCII
 - `unsigned char`
- **interi con segno**
 - `short (int)` -32768 ... 32767 (16 bit)
 - `int` ???????
 - `long (int)` -2147483648 2147483647 (32 bit)
- **naturali (interi senza segno)**
 - `unsigned short (int)` 0 ... 65535 (16 bit)
 - `unsigned (int)` ???????
 - `unsigned long (int)` 0 ... 4294967295 (32 bit)

Dimensione di `int` e `unsigned int` non fissa. Dipende dal compilatore

COSTANTI DI TIPI PRIMITIVI

- **interi** (in varie basi di rappresentazione)

	2 byte	4 byte
decimale	12	70000, 12L
ottale	014	0210560
esadecimale	0xFF	0x11170
- **reali**
 - in doppia precisione
 - `24.0` `2.4E1` `240.0E-1`
 - in singola precisione
 - `24.0F` `2.4E1F` `240.0E-1F`

COSTANTI DI TIPI PRIMITIVI

- **caratteri**
 - singolo carattere racchiuso fra apici
'A' 'C' '6'
 - caratteri speciali:
'\n' '\t' '\\ ' \\' '\\"'

ESPRESSIONI

- Il C è un linguaggio basato su *espressioni*
- Una *espressione* è una *notazione che denota un valore* mediante un processo di *valutazione*
- Una espressione può essere *semplice* o *composta* (tramite aggregazione di altre espressioni)

STRINGHE

- Una *stringa* è una *sequenza di caratteri delimitata da virgolette*
"ciao" "Hello\n"
- In C le stringhe sono semplici sequenze di caratteri di cui l'ultimo, *sempre presente in modo implicito*, è '\0'
"ciao" = {'c', 'i', 'a', 'o', '\0'}

ESPRESSIONI SEMPLICI

Quali espressioni elementari?

- **costanti**
 - 'A' 23.4 -3 "ciao"
- **simboli di variabile**
 - x pippo pigreco
- **simboli di funzione**
 - f(x)
 - concat("alfa","beta")
 - ...

OPERATORI ED ESPRESSIONI COMPOSTE

- Ogni linguaggio introduce un insieme di operatori
- che permettono di aggregare altre espressioni (*operandi*)
- per formare espressioni composte
- con riferimento a diversi domini / tipi di dato (numeri, testi, ecc.)

Esempi

```
2 + f(x)
4 * 8 - 3 % 2 + arcsin(0.5)
strlen(strcat(buf, "alfa"))
a && (b || c)
...
```

OPERATORI ARITMETICI

operazione	operatore	C
inversione di segno	unario	-
somma	binario	+
differenza	binario	-
moltiplicazione	binario	*
divisione fra interi	binario	/
divisione fra reali	binario	/
modulo (fra interi)	binario	%

NB: la divisione a/b è fra interi se sia a sia b sono interi, è fra reali in tutti gli altri casi

CLASSIFICAZIONE DEGLI OPERATORI

- Due criteri di classificazione:
 - in base al tipo degli operandi
 - in base al numero degli operandi

in base al tipo degli operandi	in base al numero di operandi
<ul style="list-style-type: none">• aritmetici• relazionali• logici• condizionali• ...	<ul style="list-style-type: none">• unari• binari• ternari• ...

OPERATORI: OVERLOADING

- In C (come in Pascal, Fortran e molti altri linguaggi) operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo (ad esempio, le operazioni aritmetiche su reali o interi).
- In realtà l'operazione è diversa e può produrre risultati diversi.

```
int X, Y;
se X = 10 e Y = 4;
X/Y vale 2
```

```
int X; float Y;
se X = 10 e Y = 4.0;
X/Y vale 2.5
```

```
float X, Y;
se X = 10.0 e Y = 4.0;
X/Y vale 2.5
```

CONVERSIONI DI TIPO

- In C è possibile combinare tra di loro operandi di tipo diverso:
 - espressioni **omogenee**: tutti gli operandi sono dello stesso tipo
 - espressioni **eterogenee**: gli operandi sono di tipi diversi.
- **Regola adottata in C**:
 - sono eseguibili le espressioni eterogenee in cui tutti i tipi referenziati risultano **compatibili** (cioè: dopo l'applicazione della regola automatica di conversione implicita di tipo del C risultano omogenei).

CONVERSIONI DI TIPO

```
int x;  
char y;  
double r;  
(x+y) / r
```

La valutazione dell'espressione procede da sinistra verso destra

- **Passo 1:** **(x+y)**
 - y viene convertito nell'intero corrispondente
 - viene applicata la somma tra interi
 - **risultato intero tmp**
- **Passo 2**
 - tmp / r tmp viene convertito nel double corrispondente
 - viene applicata la divisione tra reali
 - **risultato reale**

CONVERSIONI DI TIPO

- Data una espressione *x op y*.
 - 1. Ogni variabile di tipo **char** o **short** viene convertita nel tipo **int**;
 - 2. Se dopo l'esecuzione del passo 1 l'espressione e' ancora eterogenea, rispetto alla seguente gerarchia
 - `int < long < float < double < long double`si converte temporaneamente l'operando di tipo *inferiore* al tipo *superiore* (**promotion**);
 - 3. A questo punto l'espressione e' **omogenea** e viene eseguita l'operazione specificata. Il risultato e' di tipo uguale a quello prodotto dall'operatore effettivamente eseguito. (In caso di overloading, quello più alto gerarchicamente).

OPERATORI RELAZIONALI

Sono tutti operatori **binari**:

relazione	C
uguaglianza	==
diversità	!=
maggiore di	>
minore di	<
maggiore o uguale a	>=
minore o uguale a	<=

OPERATORI RELAZIONALI

Attenzione:

- non esistendo il tipo *boolean*, in C le espressioni relazionali denotano un valore intero
 - 0 denota *falso* (condizione non verificata)
 - 1 denota *vero* (condizione verificata)

OPERATORI LOGICI

- Anche qui sono possibili espressioni miste, utili in casi specifici
5 && 7 0 || 33 !5
- Valutazione in *corto-circuito*
 - la valutazione dell'espressione cessa appena si è in grado di determinare il risultato
 - il secondo operando è valutato solo se necessario

OPERATORI LOGICI

connettivo logico	operatore	C
not (negazione)	unario	!
and	binario	&&
or	binario	

- Anche le espressioni logiche denotano un valore intero
- da interpretare come vero (1) o falso (0)

VALUTAZIONE IN CORTO CIRCUITO

- 22 || x
già vera in partenza perché 22 è vero
- 0 && x
già falsa in partenza perché 0 è falso
- a && b && c
se a&&b è falso, il secondo && non viene neanche valutato
- a || b || c
se a || b è vero, il secondo || non viene neanche valutato

ESPRESSIONI CONDIZIONALI

Una espressione condizionale è introdotta dall'operatore ternario

$condiz ? espr1 : espr2$

L'espressione denota:

- o il valore denotato da $espr1$
 - o quello denotato da $espr2$
 - in base al valore della espressione $condiz$
-
- se $condiz$ è vera, l'espressione nel suo complesso denota il valore denotato da $espr1$
 - se $condiz$ è falsa, l'espressione nel suo complesso denota il valore denotato da $espr2$

ESPRESSIONI CONCATENATE

Una espressione concatenata è introdotta dall'operatore di concatenazione (la virgola)

$espr1, espr2, \dots, esprN$

- tutte le espressioni vengono valutate (da sinistra a destra)
- l'espressione esprime il valore denotato da $esprN$
- Supponiamo che
 - i valga 5
 - k valga 7
- Allora l'espressione: $i + 1, k - 4$ denota il valore denotato da $k-4$, cioè 3.

ESPRESSIONI CONDIZIONALI: ESEMPI

- $3 ? 10 : 20$
denota sempre 10 (3 è sempre vera)
- $x ? 10 : 20$
denota 10 se x è vera (diversa da 0),
oppure 20 se x è falsa (uguale a 0)
- $(x > y) ? x : y$
denota il maggiore fra x e y

OPERATORI INFISSI, PREFISSI E POSTFISSI

- Le espressioni composte sono **strutture** formate da **operatori** applicati a uno o più **operandi**
- **Ma.. dove posizionare l'operatore rispetto ai suoi operandi?**

OPERATORI INFISSI, PREFISSI E POSTFISSI

• Tre possibili scelte:

- prima → notazione *prefissa*
Esempio: **+ 3 4**
- dopo → notazione *postfissa*
Esempio: **3 4 +**
- in mezzo → notazione *infissa*
Esempio: **3 + 4**



E' quella a cui siamo abituati,
perciò è adottata *anche in C.*

OPERATORI INFISSI, PREFISSI E POSTFISSI

- Notazione prefissa:
* + 4 5 6
 - si legge come $(4 + 5) * 6$
 - denota quindi 54
- Notazione postfissa:
4 5 6 + *
 - si legge come $4 * (5 + 6)$
 - denota quindi 44

OPERATORI INFISSI, PREFISSI E POSTFISSI

- Le notazioni *prefissa* e *postfissa* non hanno problemi di *priorità* e/o *associatività* degli operatori
 - non c'è mai dubbio su *quale* operatore vada applicato a *quali* operandi
- La notazione *infissa* richiede *regole di priorità* e *associatività*
 - per identificare univocamente *quale* operatore sia applicato a *quali* operandi

PRIORITA' DEGLI OPERATORI

- **PRIORITÀ**: specifica l'ordine di valutazione degli operatori quando in una espressione compaiono operatori (*infissi*) diversi
- **Esempio**: $3 + 10 * 20$
 - si legge come $3 + (10 * 20)$ perché l'operatore * è più prioritario di +
- NB: operatori diversi possono comunque avere *egual priorità*

ASSOCIATIVITA' DEGLI OPERATORI

- **ASSOCIATIVITA'**: specifica l'ordine di valutazione degli operatori quando in una espressione compaiono operatori (*infixi*) di egual *priorità*
- Un operatore può quindi essere **associativo a sinistra** o **associativo a destra**
- **Esempio:** $3 - 10 + 8$
 - si legge come $(3 - 10) + 8$ perché gli operatori $-$ e $+$ sono equiprioritari e **associativi a sinistra**

PRIORITA' e ASSOCIATIVITA'

- **Priorità e associatività predefinite possono essere alterate mediante l'uso di parentesi**
- **Esempio:** $(3 + 10) * 20$
 - denota 260 (anziché 203)
- **Esempio:** $30 - (10 + 8)$
 - denota 12 (anziché 28)

RIASSUNTO OPERATORI DEL C

Priorità	Operatore	Simbolo	Associatività
1 (max)	chiamate a funzione selezioni	() [] -> .	a sinistra
2	operatori unari: op. negazione op. aritmetici unari op. incr. / decr. op. indir. e deref. op. sizeof	! - + ++ -- & * sizeof	a destra
3	op. moltiplicativi	* / %	a sinistra
4	op. additivi	+ -	a sinistra

RIASSUNTO OPERATORI DEL C

Priorità	Operatore	Simbolo	Associatività
5	op. di shift	>> <<	a sinistra
6	op. relazionali	< <= > >=	a sinistra
7	op. uguaglianza	== !=	a sinistra
8	op. di AND bit a bit	&	a sinistra
9	op. di XOR bit a bit		a sinistra
10	op. di OR bit a bit		a sinistra
11	op. di AND logico	&&	a sinistra
12	op. di OR logico		a sinistra
13	op. condizionale	?...:	a destra
14	op. assegnamento e sue varianti	= += -= *= /= %= = &= = <<= >>=	a destra
15 (min)	op. concatenazione	.	a sinistra