# Fondamenti di Informatica

## Introduzione

Tullio Facchinetti
<tullio.facchinetti@unipv.it>

http://robot.unipv.it/toolleeo

Ultimo aggiornamento: 08-03-2019

## Informazioni generali

sito di riferimento:

http://robot.unipv.it/toolleeo/

## Informazioni generali

parte del materiale è disponibile all'indirizzo:

http://robot.unipv.it/toolleeo/contrib/c/

testo di riferimento:

Facchinetti, Larizza, Rubini

Programmare in C – Concetti base e tecniche avanzate

Hoepli Editore, 2015

## Programma degli argomenti del corso

### parte "teoria della programmazione":

- problemi e risoluzione mediante algoritmi
- strutture di controllo, algoritmi e programmi, diagrammi di flusso
- ambiente di programmazione (Linux), comandi Unix - shell

### parte "programmazione":

- linguaggio C

### parte "teoria della programmazione" (2):

- semplici algoritmi e strutture dati, con uso del C per gli esempi

### parte "teoria della programmazione" (2):

- semplici algoritmi e strutture dati, con uso del C per gli esempi

# Cos'è un problema

per problema si intende un compito che si vuole far risolvere automaticamente a un calcolatore

per risolvere un problema bisogna svolgere le seguenti attività:

1. comprendere il problema
2. definire un procedimento risolutivo (algoritmo) per il problema
3. implementare l'algoritmo in un linguaggio di programmazione

la descrizione del problema non fornisce (in genere) un metodo per calcolare il risultato

# Esempi di problemi

- dati due numeri trovare il maggiore
- dati $a$ e $b$, risolvere l'equazione $ax + b = 0$
- calcolare il massimo comun divisore fra due numeri dati
- dato un insieme di numeri, ordinarli in ordine crescente
- dato un elenco di nomi e relativi numeri di telefono trovare il numero di telefono di una determinata persona
- dato l'archivio dell'anagrafe comunale, trovare tutti i nuclei familiari composti da più di 4 persone
- dato l'archivio dei dipendenti di un'azienda, calcolare lo stipendio di ogni dipendente dell'azienda

# Problemi non risolvibili

non tutti i problemi sono risolvibili attraverso l'uso del calcolatore

esistono classi di problemi per le quali la soluzione automatica non è possibile:

- se il problema presenta infinite soluzioni
- per alcuni dei problemi non è stato trovato un metodo risolutivo
- per alcuni problemi è stato dimostrato che non esiste un metodo risolutivo automatizzabile

considereremo problemi che ammettono un metodo risolutivo

# L'algoritmo

algoritmo: insieme di passi elementari dalla interpretazione univoca (istruzioni) che, eseguiti secondo un ordine prestabilito, permettono di arrivare ai risultati a partire dai dati del problema

un algoritmo è assimilabile a:

- una ricetta di cucina
- le istruzioni per il montaggio di un mobile
- le istruzioni per far funzionare un elettrodomestico
- le istruzioni per installare un programma

# Caratteristiche di un algoritmo

un algoritmo deve

- essere corretto, ovvero deve permettere effettivamente di risolvere il problema
- essere applicabile a qualsiasi insieme dei dati di ingresso appartenenti al dominio di definizione dell'algoritmo

# Non ambiguità di un algoritmo

un algoritmo deve essere espresso in termini delle istruzioni di un esecutore automatico (calcolatore)

## non ambiguità

- essere costituito da operazioni appartenenti ad un determinato insieme di operazioni fondamentali (sistema formale)
- ogni operazione deve essere univocamente interpretabile dall'esecutore, cioè deve essere descritta in modo preciso
- carattere deterministico: il risultato non deve cambiare al variare dell'esecutore (macchina/persona) dell'algoritmo

# Finitezza di un algoritmo

**finitezza**

l'intera sequenza di istruzioni deve poter essere eseguita in tempo finito, per ogni possibile insieme di ingresso che soddisfa la pre-condizione del problema

per ogni insieme di dati di ingresso:

- ciascuna istruzione deve poter essere eseguita dall'esecutore in tempo finito (proprietà dell'esecutore)
- il numero totale di istruzioni da eseguire è finito
- le operazioni da esse specificate devono essere eseguite un numero finito di volte

## Realizzabilità ed efficienza di un algoritmo

- realizzabilità: ogni operazione prevista dall'algoritmo deve essere eseguibile con le risorse a disposizione
- efficienza: l'esecuzione dell'algoritmo deve richiedere un uso limitato di risorse

> tipiche risorse che devono essere salvaguardate sono il tempo di esecuzione e la memoria utilizzata

ultimamente sta diventando sempre più importante l'efficienza energetica

# Parametricità di un algoritmo

**parametricità**

l'algoritmo deve essere formulato in modo da dipendere da dati (parametri) i cui valori non sono noti al momento della formulazione dell'algoritmo

## problemi parametrici:

- dipendono da dati i cui valori non sono noti al momento della formulazione dell'algoritmo risolutivo
- i dati sono i parametri da fornire alla procedura di risoluzione al momento in cui questa viene eseguita

## Comprensibilità e manutenibilità di un algoritmo

un algoritmo deve essere concepito tenendo conto di:

- leggibilità: deve essere quanto più facilmente comprensibile possibile
- modificabilità: deve essere facilmente modificabile, a fronte di (piccole) modifiche nelle specifiche del problema risolto dall'algoritmo
- riusabilità: la possibilità di riutilizzare l'algoritmo come mattone elementare per la soluzione di problemi più complessi

# Esempi di algoritmo: calcolo del massimo tra un insieme di numeri

problema: trovare il massimo tra un insieme di numeri

dati di ingresso: il vettore $A$ contenente $n$ valori $a_i$, con $1 \leq i \leq n$

dati di uscita: il valore massimo M

pre-condizione: deve essere verificato che il vettore contenga effettivamente valori numerici confrontabili tra loro

specifica dell'algoritmo:

- porre $M = a_1$
- per ciascun numero $a_i$, con $i$ da 2 a $n$:
- se $a_i > M$ allora porre $M = a_i$

come cambia l'algoritmo se invece del valore di $M$ mi interessa l'indice $i$ del valore massimo $a_i$?

# Esempi di algoritmo: prodotto di matrici

problema: calcolare il prodotto di due matrici $A$ e $B$ di dimensione rispettivamente $m \times n$ e $n \times p$.

dati di ingresso: gli $m \cdot n$ valori che compongono $A$, che saranno indicati con $a_{ij}$; gli $n \cdot p$ valori che compongono $B$, che saranno indicati con $b_{ij}$.

dati di uscita: una matrice $C$ di dimensione $m \times p$.

$$C = A \times B$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}$$

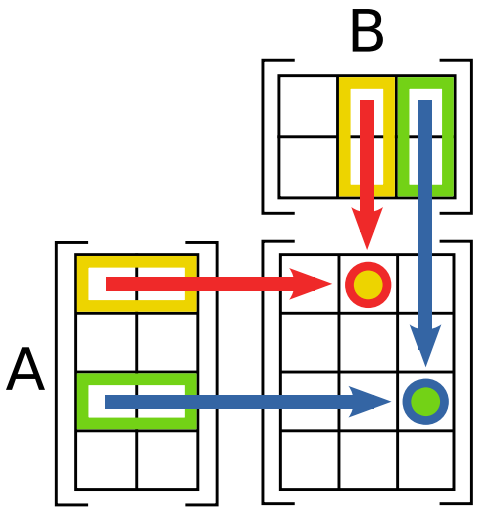
# Esempi di algoritmo: prodotto di matrici

pre-condizione: deve essere verificato che il numero di colonne di $A$ deve essere uguale al numero di righe di $B$.

specifica dell'algoritmo:

- l'elemento $c_{ij}$ della matrice $C$ viene calcolato come

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \ldots + a_{in} b_{nj}$$




fonte immagine: http://it.wikipedia.org/wiki/Moltiplicazione_di_matrici

# Il programma

**Programma**

consiste nell'implementazione mediante un linguaggio di programmazione della sequenza di azioni che devono essere eseguite per realizzare il compito desiderato

il termine programma è spesso usato erroneamente in modo intercambiabile con altri termini, come software o applicazione (una applicazione può essere composta da diversi programmi)

# La programmazione

**Programmazione**

la programmazione è l'insieme delle attività che il programmatore svolge per creare un programma

il processore si occupa di eseguire le istruzioni del programma

# Il programma

le operazioni elementari per il funzionamento di un programma sono 4:

1. trasferimento di informazioni: acquisizione dati, visualizzazione risultati intermedi, scrittura risultati finali
2. esecuzione di calcoli
3. assunzione di decisioni: scelta della successiva operazione da compiere sulla base di risultati intermedi
4. esecuzione di iterazioni: ripetizione di sequenze di operazioni

## Il programma

per descrivere un algoritmo non è possibile utilizzare il linguaggio naturale che può presentare ambiguità che potrebbero causare interpretazioni false o errate

si utilizzano linguaggi sintetici e standardizzati in modo da consentire all'esecutore una interpretazione univoca

# Esecuzione del programma

è la fase con la quale le istruzioni rappresentate in linguaggio macchina vengono messe in esecuzione dal processore

le tipiche operazioni compiute sono

- caricamento in memoria, tipicamente a partire da una periferica di memoria di massa, come un disco rigido
- identificazione del "punto d'ingresso" del programma
- esecuzione sequenziale delle istruzioni (fetch + esecuzione)

# Errori e debug

l'errore di programmazione viene universalmente chiamato bug

esistono errori di tipo sintattico, semantico e logico

"fare il debug" di un programma significa ricercare e correggere gli errori

# Tipologie di errori

- errori di sintassi
- errori semantici
- errori logici

# Correttezza di un programma

l'area di un triamgolo e base per altezza

- **errore sintattico**
- contiene una parola che non è di senso compiuto

# Correttezza di un programma

l'area di un triangolo e base per altezza

- **errore semantico**
- corretta sintatticamente: non contiene parole o costrutti non validi
- è composta da due frasi di senso compiuto unite dalla congiunzione "e"
- non è corretta dal punto di vista semantico: sostanzialmente non significa nulla

# Correttezza di un programma

una frase corretta sia sintatticamente che semanticamente:

> l'area di un triangolo è base per altezza

- **errore logico**
- la frase è corretta sia sintatticamente che semanticamente
- non è corretta dal punto di vista logico: in questo caso l'affermazione è falsa

## Correttezza di un programma

una frase corretta dal punto di vista sintattico, semantico e logico è la seguente:

> l'area di un rettangolo è base per altezza

## Correttezza di un programma

anche la frase

> l'area di un triangolo è base per altezza diviso due

è corretta dal punto di vista sintattico, semantico e logico

> non è possibile correggere in modo automatico un errore logico, poichè in generale non sono noti al "correttore" gli scopi del programma

## Errori di sintassi

- infrangono delle regole ben definite relative a **come si scrive il codice** in modo che sia comprensibile dal traduttore o esecutore
- sono relativamente semplici da trovare
- sono segnalati in modo automatico dagli strumenti usati per lo sviluppo di un programma

es.

- dimenticare di chiudere una parentesi precedentemente aperta: es. `(((a + b) * c) - 1`
- usare un numero come prima lettera per un identificatore: es. `a + 2b`

## Errori semantici

- infrangono regole relative **all'uso coerente degli elementi del linguaggio**, che deriva dal loro significato
- in genere è possibile indentificarli in modo automatico
- sono ricercati su programmi sintatticamente corretti

es.

- richiamare una funzione che non esiste
- confrontare un numero intero con una stringa (testo)
- assegnare un valore ad una costante (3 = x)
- assegnare un valore ad una espressione (x + y = 3)

# Errori logici

- sono collegati alla logica di funzionamento del programma
- sono più difficili da identificare
- è difficile rilevarli per mezzo di procedure automatiche
- spesso dipendono dai dati in ingresso
- si manifestano in fase di esecuzione del programma, cosa che complica ulteriormente il debugging

es.

- effettuare un ciclo per un numero di volte errato
- combinare in modo errato più test nelle istruzioni condizionali
- dividere un numero per zero (es. x = a / b, dove b vale 0)

## Errori di design

- il programma è corretto sintatticamente, semanticamente e anche dal punto di vista logico
- quanto eseguito, produce i risultati corretti a fronte del valore dei dati in ingresso
- il programma però non risolve il problema di partenza, in quanto quest'ultimo non è stato compreso (e/o specificato) correttamente

es.

- realizzare un programma che restituisca il numero di studenti iscritti all'Università di Pavia
- non viene specificato il significato di "iscritti", che può o meno comprendere gli studenti già laureati
- una soluzione che non tenga conto di questo aspetto potrebbe non fornire il risultato desiderato anche svolgendo correttamente tutte le elaborazioni

## Testing

viene effettuato su un programma sintatticamente e semanticamente corretto (quindi eseguibile)

il programma viene collaudato per verificarne la correttezza logica

## Testing

si verifica che l'algoritmo implementato svolga correttamente le operazioni previste a fronte di determinati dati in ingresso

si realizza fornendo in ingresso al programma opportuni valori di input per verificare che l'output corrispondente sia corretto

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." – Brian W. Kernighan.

## Esempio di testing

si dispone della funzione

$$sqrt(x) \Rightarrow val$$

che restituisce *val*, la radice quadrata di un valore numerico $x$ compreso nell'intervallo $[0, 2^{32} - 1]$

- $sqrt(9) \stackrel{?}{=} 3.0$
- $sqrt(0) \stackrel{?}{=} 0.0$
- $sqrt(2^{32} - 1) \stackrel{?}{=} 65535.999992370605468305 91$
- $sqrt(2^{32}) \stackrel{?}{=} ERROR$
- $sqrt(-1) \stackrel{?}{=} ERROR$
- $sqrt(\text{"}testo\text{"}) \stackrel{?}{=} ERROR$

## Manutenzione

spesso il programmatore non deve sviluppare ex novo un programma, ma si trova a dover modificare un programma

la manutenzione è un aspetto talvolta trascurato del ciclo di vita di un programma

- spesso il programma è stato scritto da altri programmatori, oppure
- è stato scritto dallo stesso programmatore, ma
- è passato un periodo di tempo sufficiente da far dimenticare i dettagli dell'implementazione

1. utilizzare uno stile di programmazione chiaro e coerente
2. commentare il codice

# Cos'è un linguaggio di programmazione

il programma viene realizzato scrivendo del codice sorgente utilizzando un linguaggio di programmazione

esistono molti diversi linguaggi di programmazione, ciascuno dei quali ha caratteristiche specifiche che lo rendono adatto a compiti specifici

è importante selezionare adeguatamente il linguaggio adatto per l'applicazione specifica

# Classificazione dei linguaggi

- linguaggi interpretati vs compilati
- linguaggi di basso livello vs alto livello
- linguaggi procedurali
- linguaggi funzionali
- linguaggi dichiarativi
- linguaggi ad oggetti
- linguaggi di scripting

le tipologie di linguaggi di programmazione
non sono esclusive
(es. linguaggio compilato, di alto livello, funzionale)

# Linguaggi di basso vs alto livello

per livello di un linguaggio si intende la sua vicinanza al modo di rappresentare il codice rispetto alla macchina che deve eseguirlo piuttosto che al programmatore che deve scriverlo

linguaggi di basso livello

- linguaggi vicini alla rappresentazione usata dalla macchina
- tipicamente più complicato da scrivere e/o comprendere

linguaggi di alto livello

- linguaggi vicini alla rappresentazione umana
- tipicamente più "descrittivi" e facili da scrivere e/o leggere

# Linguaggi interpretati vs compilati

differenzia lo stadio al quale viene analizzato il codice sorgente

linguaggi interpretati

- il codice sorgente viene interpretato ed eseguito direttamente da un apposito programma chiamato interprete
- generalmente più lenti in quanto c'è l'overhead dovuto all'esecuzione dell'interprete

linguaggi compilati

- richiedono che il codice sorgente, una volta terminato, sia processato da un compilatore che lo converte in linguaggio macchina e ne permette l'esecuzione da parte della CPU

# Linguaggi interpretati vs compilati

linguaggi interpretati

- es. BASIC, Perl, Python, MATLAB

linguaggi compilati

- es. C, Pascal

# Linguaggi procedurali

l'organizzazione di un programma è basato su blocchi logico-funzionali chiamati procedure o funzioni

- la funzione raggruppa un insieme di istruzioni e/o chiamate ad altre funzioni che implementano funzionalità specifiche e ben definite
- la divisione di un programma in funzioni rende più chiara la stesura del codice
- una stessa funzione può essere richiamata varie volte nel corso del programma

es. C, ...

# Linguaggi dichiarativi

non viene implementato direttamente un algoritmo

- il programmatore non specifica come deve essere ottenuto il risultato, ovvero non implementa un algoritmo
- il programmatore indica quali sono i dati coinvolti nel calcolo del risultato e qual è il risultato desiderato
- le azioni per mettere in relazione i dati al fine di ottenere il risultato desiderato sono individuate e compiute da un interprete

es. Prolog, make

## Linguaggi a oggetti

evoluzione della programmazione procedurale con l'introduzione degli oggetti

gli oggetti sono caratterizzati da:

1. incapsulamento, cioè l'oggetto incorpora sia i dati che le funzioni che operano sui dati
2. ereditarietà
3. polimorfismo

es. C++, Java

# Linguaggi di scripting

servono ad automatizzare l'esecuzione di lunghe attività sequenziali

- sono linguaggi nati per descrivere una sequenza di esecuzione di altri programmi o comandi (esecuzione batch)
- in seguito sono state aggiunte funzionalità quali l'esecuzione di cicli e l'uso di variabili
- sono tutti linguaggi interpretati

es. PHP, Perl, JavaScript, Python, shell Unix

# Evoluzione del linguaggio C

1966 : Martin Richards sviluppa il BCPL, pensato per scrivere sistemi operativi e software di sistema

- alcune caratteristiche del BCPL sono ereditate dal linguaggio B, anch'esso sviluppato con lo stesso scopo da Ken Thompson nel 1970 per il primo sistema UNIX

1972 : Dennis Ritchie progettava e realizzava, presso i Bell Laboratories, la prima versione del linguaggio C

- gli stessi Ritchie e Thompson riscrissero in C il codice di UNIX
- inizialmente UNIX viene utilizzato solo nei Laboratori Bell (come ambiente di sviluppo del s/w), quindi nell'università californiana di Berkeley (UCB). In questi due ambienti UNIX si sviluppa fino a diventare uno dei sistemi più completi sul mercato. Il C si sviluppa e si diffonde parallelamente a UNIX

# Evoluzione del linguaggio C

1983 : l'Istituto Nazionale Americano per gli Standard (ANSI), costituisce un comitato per definire in modo completo il linguaggio e l'insieme minimo di funzioni di libreria che un compilatore deve implementare

1989 : è approvato lo standard ANSI o ANSI C

1995 : adottato l'Emendamento 1 al C Standard che, fra le altre cose, ha aggiunto nuove funzioni alla libreria standard del linguaggio

1998 : viene ratificato il C++ come standard ISO, sviluppato da Bjarne Stroustrup a partire dal C89 con l'Emendamento 1, e unendovi l'uso delle classi di Simula

1999 : promulgazione dello standard ISO C99 (codice ISO 9899)