

# Programmazione 1

## Modulo C

Marco Beccuti

*Università degli Studi di Torino*  
*Dipartimento di Informatica*

Settembre 2020



# Qualche informazione essenziale

## *Docente:*

Marco Beccuti, Ricercatore

## *Ufficio:*

Primo piano, Dipartimento di Informatica  
Corso Svizzera 185  
10149 Torino

## *E-mail:*

beccuti@di.unito.it o marco.beccuti@unito.it

## *Home page:*

<http://www.di.unito.it/~beccuti>



La mia ricerca si applica alla modellizzazione e simulazione computazionale di sistemi complessi.  
In particolare sono interessato:

- alla definizione di linguaggi di modellazione;
- allo sviluppo di tecniche d'analisi esatte o approssimate per studiare il comportamento di sistemi complessi;
- ad applicazioni in ambito della scienza della vita.

Mi occupo anche della definizione di algoritmi bioinformatici per l'analisi di dati "omici" (es. genomici, trascrittomici, proteomici ...) con enfasi particolare sugli aspetti di riproducibilità dei risultati.

# Qualche informazione essenziale

- Il corso consiste in 48 ore (teoria) + 30 ore (esercitazioni);
- Orario:**

## **PROGIC** (BECCUTI Marco)

Periodo	Giorno	Settimana	Orario	Aula
1	LUN	1	9:00 - 11:00	Aula Seminari (p. aule)
1	MAR	1	9:00 - 11:00	Aula Seminari (p. aule)

## **PROGIC T1** (BASILE Valerio)

Periodo	Giorno	Settimana	Orario	Aula
1	LUN	1	14:00 - 17:00	Laboratorio Turing

## **PROGIC T2** (BASILE Valerio)

Periodo	Giorno	Settimana	Orario	Aula
1	VEN	1	11:00 - 14:00	Laboratorio Turing

# Qualche informazione essenziale

- Il corso si terrà completamente a distanza;
- Ogni lezione consisterà in una serie di video preregistrati pubblicati su questa pagina con il procedere del corso;
- Ci saranno anche momenti interattivi in cui pro porrò esercizi e soluzioni, e risponderò alle vostre domande;
- Questo accadrà approssimativamente una volta alla settimana, per un'ora in collegamento webex, tendenzialmente al Lunedì dalle 10 alle 11 (cioé, durante l'ultima ora della lezione)
- Il link all'aula virtuale sarà disponibile sul moodle del corso.
- Si consiglia l'accesso all'aula virtuale tramite apposito client, reperibile su <https://www.webex.com/>

# Qualche informazione essenziale

- **Ricevimento:** sempre, ma su appuntamento;
- **Testo:** Walter Savitch, Programmazione di base e avanzata con Java, Pearson 2018,
- **Dispense e luci:** su moodle  
<https://informatica.i-learn.unito.it/course/view.php?id=2051>

# Obiettivi principali del corso

- Fornire i concetti di base della programmazione imperativa strutturata di alto livello, usando il linguaggio di programmazione **Java**;
- Capacità di:
  - ▶ **formalizzare** la soluzione di problemi computazionali di base per mezzo di **metodi (= procedure o funzioni)** iterativi e ricorsivi;
  - ▶ **valutare la correttezza e la terminazione** dei programmi per mezzo di semplici dimostrazioni formali;
  - ▶ **rappresentare** gli stati assunti dalla memoria durante l'esecuzione di un programma

# Argomenti principali

- Struttura di base di un calcolatore;
- Terminologia di base relativa alla programmazione;
- Algoritmi iterativi e ricorsivi;
- Java: variabili, tipi di dati fondamentali e array, assegnazione e controllo del flusso, metodi, modello di gestione della memoria;
- Correttezza parziale e terminazione.

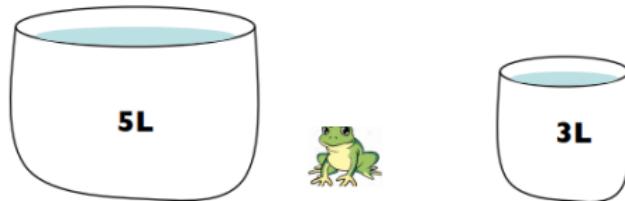
## *Dal problema algoritmico al programma*

# Problema algoritmico

Un problema viene definito algoritmico se valgono i seguenti due requisiti:

- è definito in modo effettivo l'insieme degli input legali per quel problema;
- è data una caratterizzazione effettiva dell'output come funzione dell'input.

# Un problema di “programmazione” (Die Hard)



Dati due recipienti, **G** da 5 litri e **p** da 3 litri, e una disponibilità illimitata di acqua, trovare un modo per avere esattamente 4 litri di acqua in **G** usando solo le seguenti operazioni:

riempire **G**

riempire **p**

vuotare (anche in parte) il contenuto di **G** in **p** (**G** → **p**)

vuotare (anche in parte) il contenuto di **p** in **G** (**p** → **G**)

vuotare **G**

vuotare **p**

# Un problema di “programmazione” (Die Hard)

## Una soluzione

$G = 0, p = 0$

↓      riempি G

$G = 5, p = 0$

↓      G → p

$G = 2, p = 3$

↓      vuota p

$G = 2, p = 0$

↓      G → p

$G = 0, p = 2$

↓      riempি G

$G = 5, p = 2$

↓      G → p

$G = 4, p = 3$

↓      vuota p

$G = 4, p = 0$

# Un problema di “programmazione” (Die Hard)

## Una soluzione

$G = 0, p = 0$   
↓  
riempি G

$G = 5, p = 0$   
↓  
 $G \rightarrow p$

$G = 2, p = 3$   
↓  
vuota p

$G = 2, p = 0$   
↓  
 $G \rightarrow p$

$G = 0, p = 2$   
↓  
riempি G

$G = 5, p = 2$   
↓  
 $G \rightarrow p$

$G = 4, p = 3$   
↓  
vuota p

$G = 4, p = 0$

## Altra soluzione

$G = 0, p = 0$   
↓  
riempি p

$G = 0, p = 3$   
↓  
 $p \rightarrow G$

$G = 3, p = 0$   
↓  
riempি p

$G = 3, p = 3$   
↓  
 $p \rightarrow G$

$G = 5, p = 1$   
↓  
vuota G

$G = 0, p = 1$   
↓  
 $p \rightarrow G$

$G = 1, p = 0$   
↓  
riempি p

$G = 1, p = 3$   
↓  
 $p \rightarrow G$

$G = 4, p = 0$

## Alcune domande collegate

- Si possono ottenere 4 litri di acqua in G quando:
  - a)  $G = 8, p = 3 ?$
  - b)  $G = 7, p = 3 ?$
  - c)  $G = 6, p = 3 ?$

## Risolubilità

## Alcune domande collegate

- Nel caso in cui una soluzione esiste, qual è il numero minimo di operazioni per ottenerla?

### Complessità in tempo

- Come cambia la situazione se si hanno a disposizione più recipienti, per esempio:
  - a)  $G = 5, p = 3, q = 3$  ?
  - b)  $G = 6, p = 3, q = 3$  ?

### tradeoff spazio e tempo

# Alcune domande collegate

Immaginiamo il gioco solitario in cui ogni posizione è una fila di pedine bianche (B) e nere (N). Ogni mossa è fatta in accordo con una delle seguenti regole:

- (1) rimpiazzare una pedina bianca ed una nera consecutive con due pedine bianche consecutive, in simboli  $\text{BN} \rightarrow \text{BB}$ ;
- (2) rimpiazzare due pedine bianche consecutive con una pedina nera, in simboli  $\text{BB} \rightarrow \text{N}$ ;
- (3) rimpiazzare una pedina nera ed una bianca consecutive con due pedine bianche ed una nera consecutive, in simboli  $\text{NB} \rightarrow \text{BBN}$ .

## Domande

1. è possibile, utilizzando mosse di tipo (1), (2) e (3), passare da una posizione BBB ad una posizione NNN?
2. è possibile, utilizzando mosse di tipo (1), (2) e (3), passare da una posizione BBBB ad una posizione NNNN?

## Alcune domande collegate

Immaginiamo il gioco solitario in cui ogni posizione è una fila di pedine bianche (B) e nere (N). Ogni mossa è fatta in accordo con una delle seguenti regole:

- (1) rimpiazzare una pedina bianca ed una nera consecutive con due pedine bianche consecutive, in simboli  $\text{BN} \rightarrow \text{BB}$ ;
- (2) rimpiazzare due pedine bianche consecutive con una pedina nera, in simboli  $\text{BB} \rightarrow \text{N}$ ;
- (3) rimpiazzare una pedina nera ed una bianca consecutive con due pedine bianche ed una nera consecutive, in simboli  $\text{NB} \rightarrow \text{BBN}$ .

### Domande

1. è possibile, utilizzando mosse di tipo (1), (2) e (3), passare da una posizione BBB ad una posizione NNN? **NO**
2. è possibile, utilizzando mosse di tipo (1), (2) e (3), passare da una posizione BBBB ad una posizione NNNN? **SI**

# Alcune domande collegate

- (1)  $BN \rightarrow BB$
- (2)  $BB \rightarrow N$
- (3)  $NB \rightarrow BBN$

Le sequenze di passi possibili sono le seguenti:

- $\underline{BBB} \rightarrow (2) \underline{NB} \rightarrow (3) \underline{BBN} \rightarrow (2) NN$
- $\underline{BBB} \rightarrow (2) \underline{NB} \rightarrow (3) \underline{BBN} \rightarrow (1) BBB$
- $\underline{BBB} \rightarrow (2) BN \rightarrow (1) BB \rightarrow (2) N$

Nessuna di queste può portare BBB in NNN. Ma la seguente derivazione mostra che c'è una derivazione che porta BBBB in NNNN:

- $$\begin{aligned} BBBB &\rightarrow (2) NBB \rightarrow (3) BB\underline{NB} \rightarrow (3) \underline{BBBBN} \rightarrow (2) \underline{NNBN} \\ &\quad \rightarrow (3) B\underline{BNBN} \rightarrow (3) \underline{BBBBNN} \\ &\quad \rightarrow (2) \underline{NBBNN} \\ &\quad \rightarrow (2) NNNN \end{aligned}$$

# Algoritmo

- Un algoritmo è un procedimento di calcolo, ossia un procedimento ben definito che può essere eseguito meccanicamente da **un essere umano o da una macchina** per ottenere un dato risultato (**output**) a partire da certi dati di partenza (**input**);

- Gli algoritmi sono stati inventati ben prima della nascita dei calcolatori:

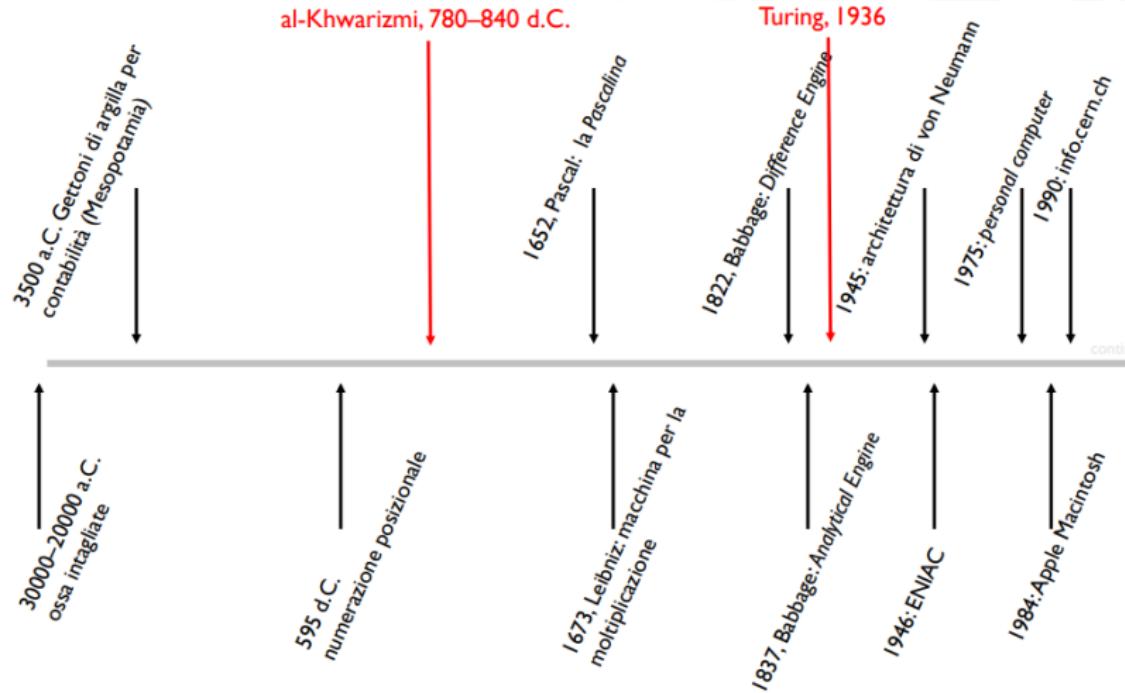
*la parola algoritmo deriva dal nome del matematico persiano al-Khwarismi (vissuto intorno all'anno 800), il cui libro “Calcoli con i numerali indiani” descriveva gli algoritmi di calcolo per le operazioni aritmetiche con il sistema di numerazione indiano, cioè quelli che ancora oggi studiamo nella scuola elementare.*

# Programma

- È la descrizione di un algoritmo scritta in un **linguaggio di programmazione**;
- Linguaggio di programmazione: in informatica, insieme di parole e di regole, definite in modo formale, per consentire la programmazione di un elaboratore affinchè esegua compiti predeterminati;
- I linguaggi di programmazione per esprimere un algoritmo:
  - ▶ **Linguaggi a basso livello** (es. Linguaggio macchina, assembly, ...)
  - ▶ **Linguaggi ad alto livello** vicini al linguaggio naturale (es. C, Java, Visul Basic, ...)

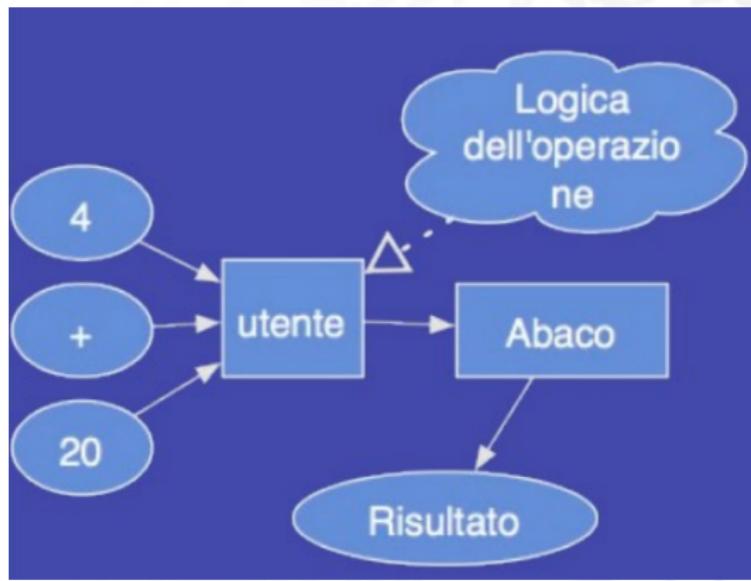
# *Storia dell'informatica ...*

# Storia dell'informatica ...



# L'abaco

- È la prima “macchina” di calcolo nota;
- È uno strumento per tenere traccia di quanto già fatto;
- La logica e la correttezza dell'operazione dipendono interamente dall'utente.



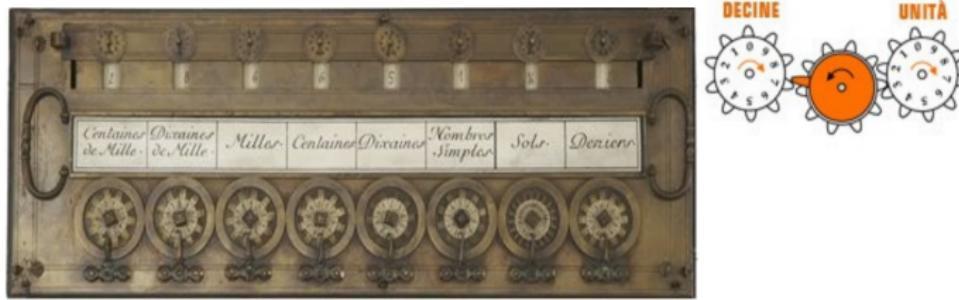
# Pascalina

- Bisogna aspettare fino al XVII secolo d.C. per avere la prima vera innovazione rispetto all'abaco;
- Inventata nel 1642 dal filosofo e matematico francese Blaise Pascal;
- E' considerata la prima addizionatrice meccanica, consente di addizionare e sottrarre fino a otto cifre, tenendo conto del riporto.



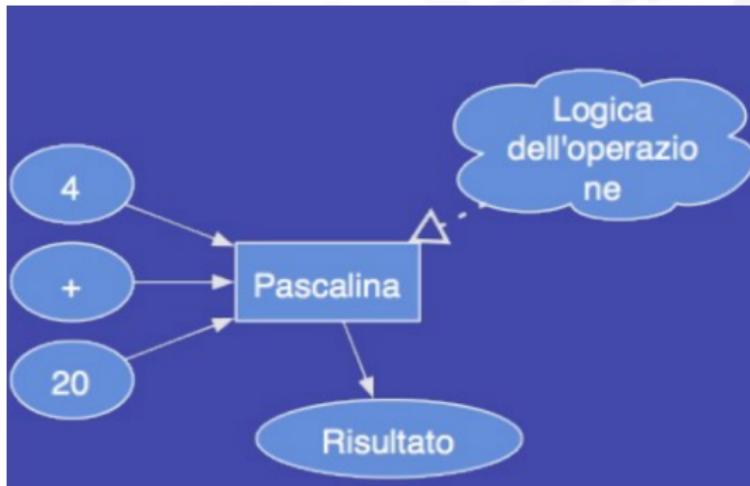
# Pascalina

- Su ogni ingranaggio sono rappresentate le cifre da 0 a 9.
- La macchina funziona come un abaco ma, effettua il riporto dell'addizione mediante un'opportuna leva inserita tra due ingranaggi successivi.



# Pascalina

- La logica dell'operazione è controllata dalla macchina.



Come aggiungere nuove operazioni (es. divisione e moltiplicazione)?

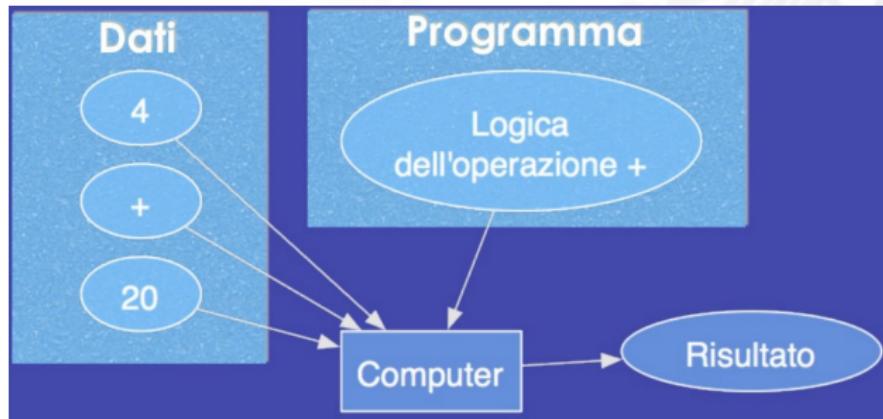
# Come aggiungere nuove operazioni?

- Si può pensare di affrontare il problema modificando la macchina in modo da introdurre la divisione e la moltiplicazione;
- Quante sono le funzioni definibili?
- Il problema vero è che la logica che governa le operazioni è **cablata** nella macchina calcolatrice.

**La soluzione è di trattare tale logica come parte dell'input della macchina.**

# Logica delle operazioni come parte dell'input?

- Individuiamo un insieme **base** di operazioni e combiniamole per rappresentare quelle più complesse;
- L'ordine delle operazioni base e i loro argomenti possono essere specificati usando solo numeri;
- Siamo quindi in grado di descrivere la logica di operazioni complesse usando il linguaggio (i numeri) usato per l'input delle macchine di calcolo



# Macchina Analitica

- Introdotta da Charles Babbage intorno al 1840, è il primo esempio di macchina di calcolo **programmabile**
- Linguaggio di programmazione **simile** all'assembly



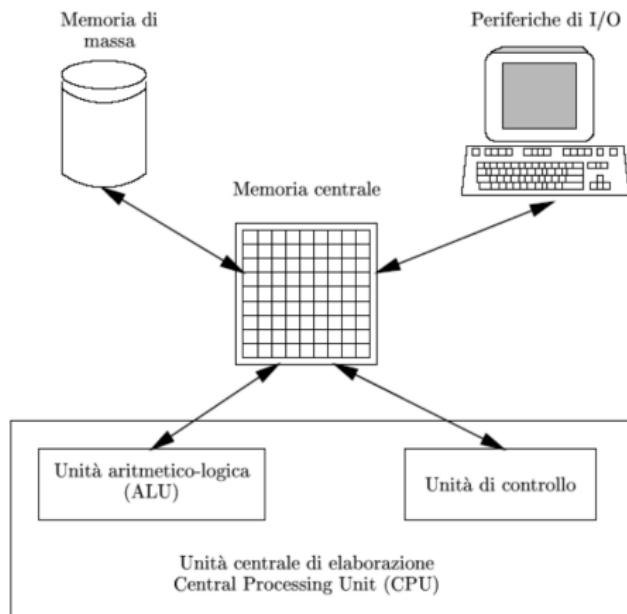
# Da Babbage ai giorni nostri

- Le idee di Babbage erano troppo avanzate per la sua epoca storica;
- Le idee di Babbage vennero riscoperte nella prima metà del'900 da **Alan Turing** e da **John von Neumann**.



# John von Neumann

- Matematico ungherese, creatore del modello di calcolatore a cui si ispirano i calcolatori moderni.



# Alan Turing

- Matematico inglese, considerato uno dei padri dell'informatica;
- Introduce il concetto di **macchina universale**: un modello astratto di calcolatore;
- È usato da Turing al fine di analizzare l'insieme delle funzioni che possono essere calcolate in modo automatico;
- Nasce uno dei pilastri dell'informatica teorica: **la teoria della calcolabilità**;
- È universale perché può calcolare tutte le funzioni **calcolabili**.



# La macchina di Turing

- Si può dimostrare che **non** tutte le funzioni definibili in matematica sono calcolabili mediante una macchina di Turing;
- Non è un problema specifico della macchina di Turing: esistono problemi che **semplicemente non possono essere risolti in modo automatico**



# La macchina di Turing

- Il calcolo è normalmente effettuato scrivendo certi **simboli** su carta;
- Possiamo supporre che la carta sia divisa in **quadretti** come i quaderni di aritmetica dei bambini;
- Nell'aritmetica elementare si sfrutta talvolta il carattere bidimensionale della carta, ma questo non è essenziale per il calcolo. Assumo quindi che il calcolo sia effettuato su carta ad una dimensione, cioè su di un nastro **suddiviso in caselle**;
- Assumo anche che il numero di simboli che possono essere scritti sia **finito**.

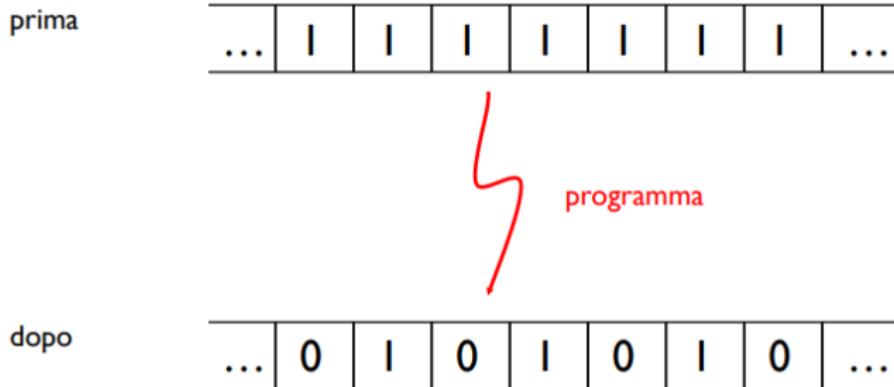


# La macchina di Turing

- Il comportamento del calcolatore ad ogni istante è determinato dal **simbolo che sta osservando** e dal suo **stato mentale** a quell'istante.
- Possiamo supporre che esista un limite superiore al numero di simboli o caselle che il calcolatore sta osservando ad un dato momento;
- Supporremo anche che il numero di stati mentali che possono essere considerati sia finito.

# La macchina di Turing

- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1



# La macchina di Turing

- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

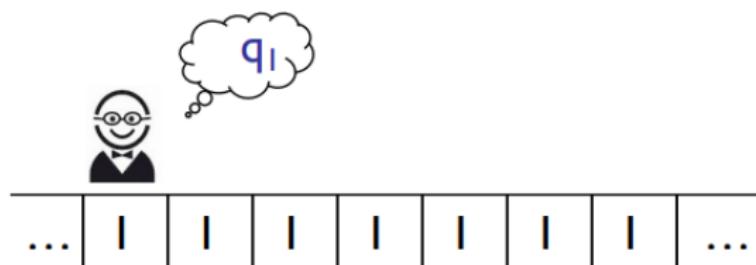
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

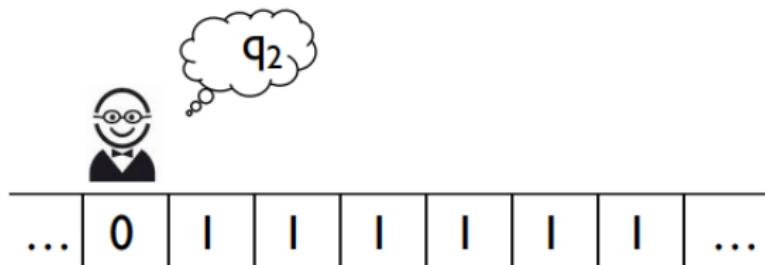
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 \textcolor{red}{0} q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

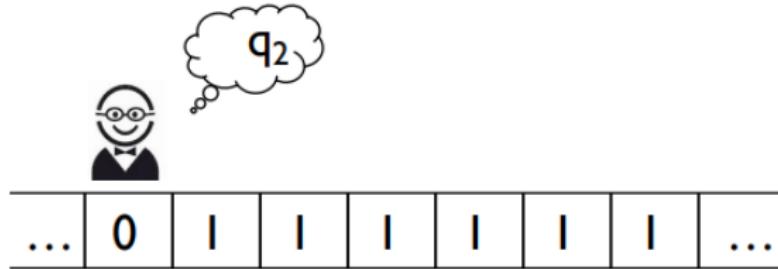
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

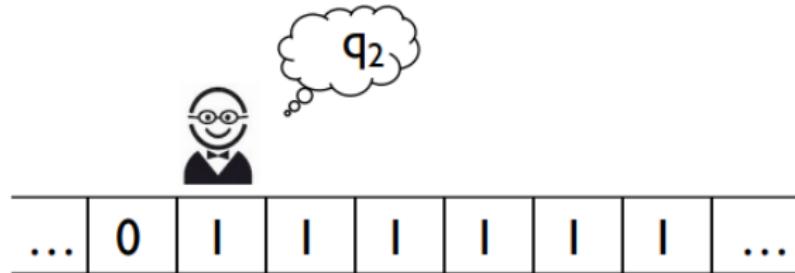
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

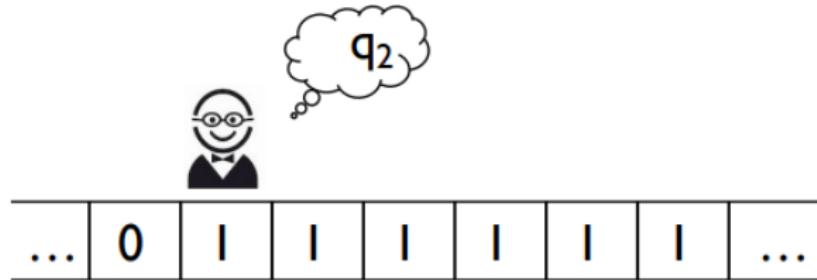
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

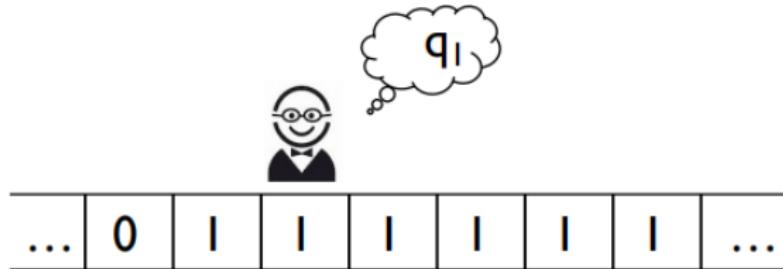
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

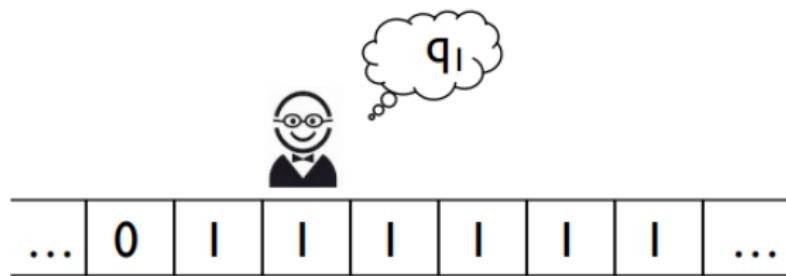
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 \mid 0 q_2$

$q_2 0 R q_2$

$q_2 \mid R q_1$



# La macchina di Turing

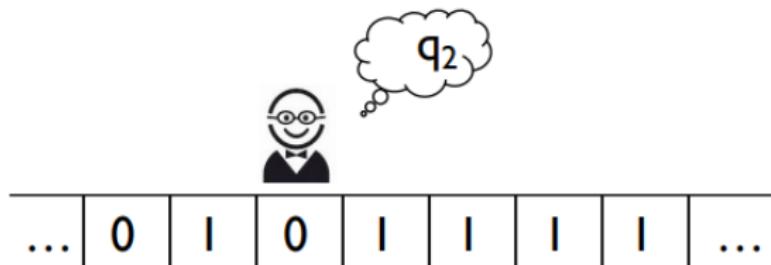
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 I 0 q_2$

$q_2 0 R q_2$

$q_2 I R q_1$



# La macchina di Turing

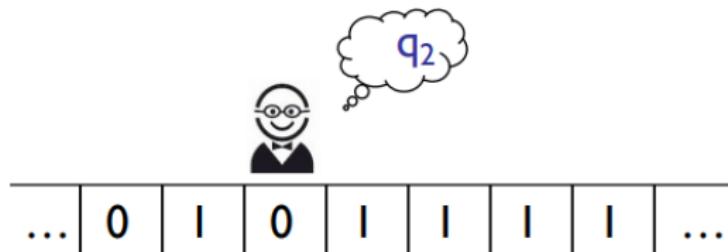
- **Problema:** scrivere un programma per trasformare una sequenza di simboli 1 in una sequenza alternante di simboli 0 e 1
- Gli stati mentali sono:  $\{q_1, q_2\}$

$q_1 0 R q_1$

$q_1 1 0 q_2$

$q_2 0 R q_2$

$q_2 1 R q_1$



# La macchina di Turing

- È universale perché può calcolare tutte le funzioni **calcolabili**;
- le funzioni calcolabili corrispondono alle **funzioni ricorsive** (Tesi di Church-Turing).

Es.:

**Funzioni ricorsive primitive:** es. funzioni costanti, funzione successore e funzioni selettive (o proiettive);

**Funzioni ricorsive** ottenute applicando un numero finito di volte la ricorsione e la composizione a partire da funzioni ricorsive primitive (es. somma, sottrazione, moltiplicazione, fattoriale, ... ).

# Funzioni non calcolabili

- Una funzione non calcolabile è: quella che risolve il **Problema della terminazione**:

*in termini algoritmici, non esiste un algoritmo **ALT** che decide se un altro algoritmo arbitrario  $A$ , operando su dati arbitrari  $D$ , termina il suo calcolo in un tempo finito.*

$$ALT(A, D) = \text{true}, \quad \text{se } A(D) \text{ termina}$$

$$ALT(A, D) = \text{false}, \quad \text{se } A(D) \text{ non termina}$$

# Funzioni non calcolabili

- Considera algoritmi che indagano sulle proprietà di altri algoritmi, che sono trattati come dati;
- È legittimo: gli algoritmi sono rappresentabili con sequenze di simboli, che possono essere presi dallo stesso alfabeto usato per codificare i dati di input;
- Una stessa sequenza di simboli può essere quindi interpretata sia come un programma, sia come un dato di ingresso di un altro programma;

# Funzioni non calcolabili

- Un algoritmo A, comunque formulato, può operare sulla rappresentazione di un altro algoritmo B  $\rightarrow A(B)$ ;
- In particolare può avere senso calcolare A(A).

# Funzioni non calcolabili

- Consideriamo un nuovo algoritmo  $P$  che impiega  $\text{ALT}$  e lavora su una sequenza  $A$  che rappresenta un algoritmo arbitrario:

$P(A)$  se( $\text{ALT}(A, A) = \text{true}$ ) ripeti il test

- $P(A)$  termina se e solo se  $A(A)$  non termina
- dunque  $P(P)$  termina se e solo se  $P(P)$  non termina !!!

**il punto debole è l'ammissione che  $\text{ALT}$  esista**

# Funzioni non calcolabili

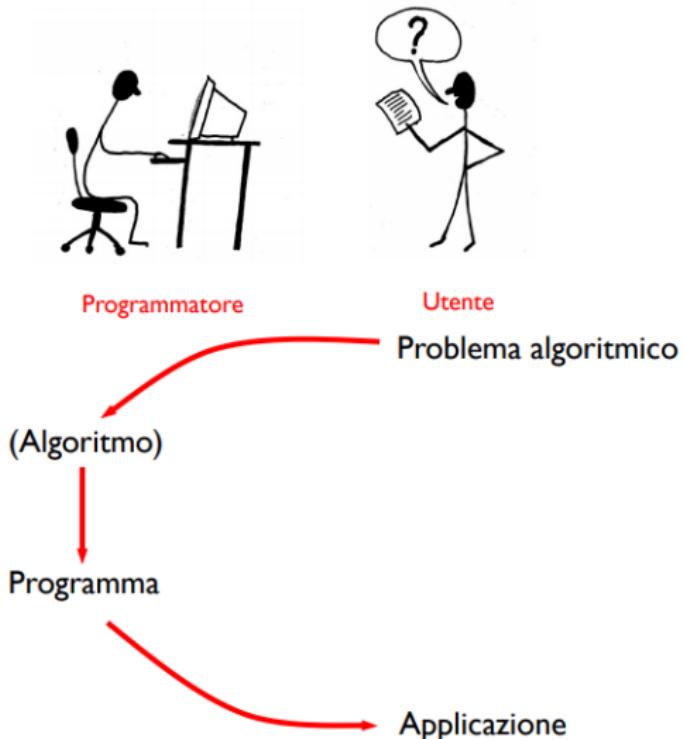
- Per trasformazione dal problema della terminazione è possibile dimostrare che molti altri problemi non sono calcolabili.

Per esempio:

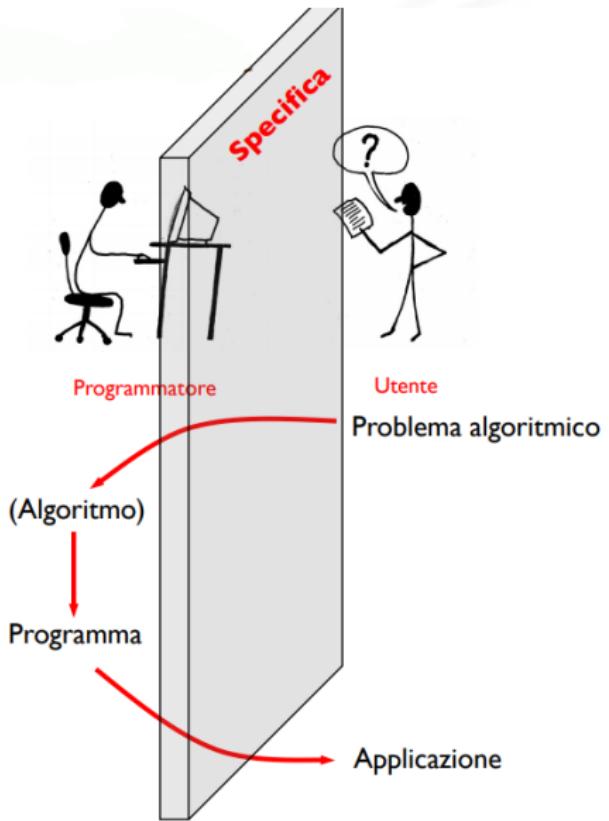
- ▶ decidere se due programmi arbitrari sugli stessi dati arbitrari generano gli stessi risultati;
- ▶ decidere se un'equazione polinomiale arbitraria ha soluzione intera;
- ▶ .....

# *Programmazione e pensiero computazionale*

# L'attività di programmazione



# L'attività di programmazione



# Alcune definizioni

- **Specifico:** Contratto tra Utente e Programmatore:

*se i dati in ingresso soddisfano la condizione di ingresso, allora i dati in uscita soddisfano la condizione di uscita*

Quindi una specifica definisce: dati in ingresso; dati in uscita; condizione di ingresso (requisito sui dati in ingresso); condizione di uscita (relazione che lega dati in ingresso e dati in uscita)

## Esempio divisione intera

dati in ingresso: interi  $\geq 0$  come dividendo  $N$ ; interi  $> 0$  come divisore  $D$ ;

dati in uscita: interi  $q \geq 0$  come quoziente  $q$ ; interi  $r \geq 0$  come resto;

condizione di ingresso: nessuna;

condizione di uscita:  $N = qD + r, r < D$

# Alcune definizioni

- **Algoritmo:** Piano di azione che prescrive le azioni da compiere per risolvere un problema. Le azioni devono poter essere eseguite in modo meccanico:

## Esempio divisione intera di N per D

```
inizialmente q = 0;  
inizialmente r = N;  
fino a quando r ≥ D  
    sottrai D a r;  
    aumenta q di 1
```

- **Correttezza (parziale) di un algoritmo rispetto ad una specifica:**  
Per ogni dato di ingresso che soddisfa la condizione di ingresso, **se** l'esecuzione termina **allora** i dati in uscita soddisfano la condizione di uscita
- **Terminazione di un algoritmo:** Per ogni dato di ingresso che soddisfa la condizione di ingresso, l'algoritmo restituisce il risultato dopo un numero finito di passi.
- **Correttezza (totale) di un algoritmo rispetto ad una specifica:**  
Correttezza parziale + terminazione

# Alcune definizioni

- **Linguaggio di programmazione:** collezione di costrutti componibili mediante regole di **sintassi** per formare, in particolare:
  - espressioni che possono essere **valutate** (da un calcolatore) per ottenere il loro valore;
  - istruzioni che possono essere **eseguite** (da un calcolatore) per modificare lo stato della sua **memoria**.
- Sono classificati in:
  - ▶ **Linguaggi a basso livello** (es. linguaggio macchina, assembly, ...)
  - ▶ **Linguaggi ad alto livello** più vicini al linguaggio naturale (es. C, Java, ...)

# Alcune definizioni

- Esempio: Stampa sullo schermo la somma fra C ed il prodotto di A e B:

*Linguaggio ad alto livello*

```
print(C+A*B)
```

*Linguaggio Assembly:*

```
mov eax,A  
mul B  
add eax,C  
call WriteInt
```

*Linguaggio macchina:*

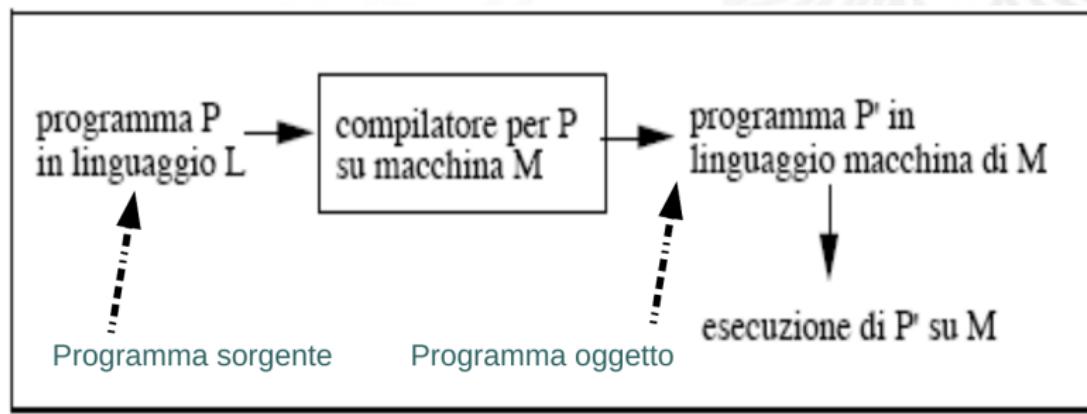
```
A1 00000000  
F7 25 00000004  
03 05 00000008  
E8 00500000
```

## Alcune definizioni

- Un programma scritto in un linguaggio di programmazione ad alto livello può essere eseguito direttamente?
- **Compilatore:** Programma che traduce costrutti di un linguaggio di programmazione in costrutti di un altro linguaggio di programmazione di più basso livello.
- **Interprete:** Programma che esegue le istruzioni e valuta le espressioni di un programma.

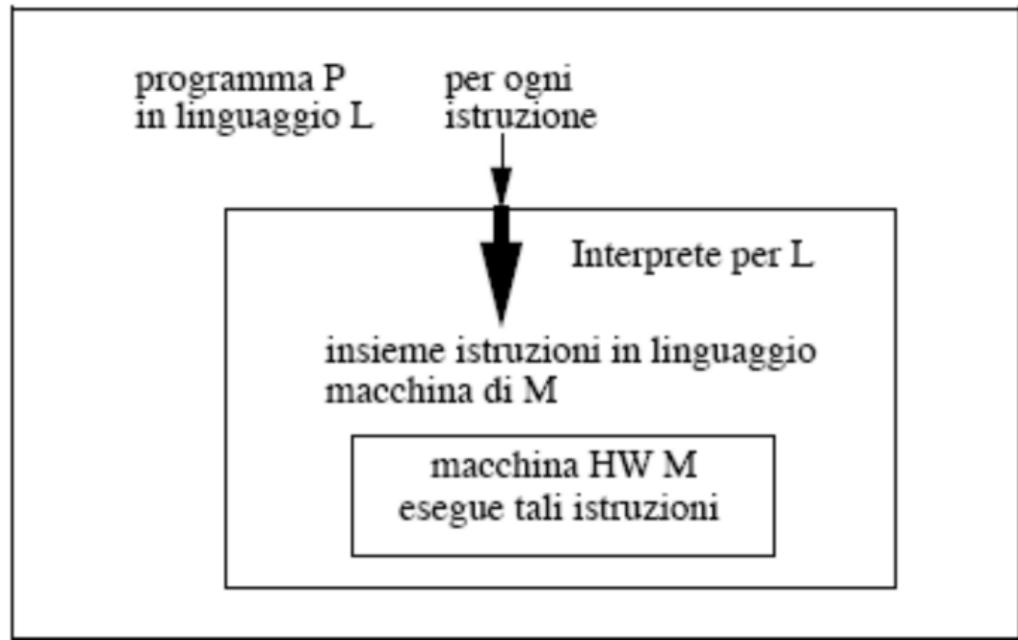
# Alcune definizioni

- Un programma scritto in un linguaggio ad alto livello è detto **programma sorgente**
- Il compilatore è un programma che esegue la traduzione, producendo il **programma oggetto**.
- Il compilatore segnala anche eventuali **errori di sintassi** nella scrittura del programma sorgente



## Alcune definizioni

- Un interprete è un programma che non produce alcun programma oggetto, ma legge ogni istruzione del programma sorgente e genera al volo le istruzioni macchina corrispondenti, che vengono passate al hardware per l'esecuzione.



# Alcune definizioni

- L'esecuzione di un programma compilato risulta più efficiente, ma il programma può essere eseguito solo per la specifica architettura per cui è stato compilato.
- Un programma interpretato, può essere eseguito su ogni architettura cambiando solo l'interprete.

## PERFORMANCE VS PORTABILITÀ

# Paradigmi di programmazione:

- **Programmazione imperativa:** un programma consiste di istruzioni che realizzano trasformazioni dello stato della memoria di un calcolatore, dove uno stato è identificato dai valori assunti ad un certo istante da un insieme di variabili;

## Programmazione 1

- **Programmazione funzionale:** Un programma è un'espressione che viene valutata per ottenere il suo risultato;  
Esempio  $f(5)$  where  $f(x) = x * 2$  ha valore 10

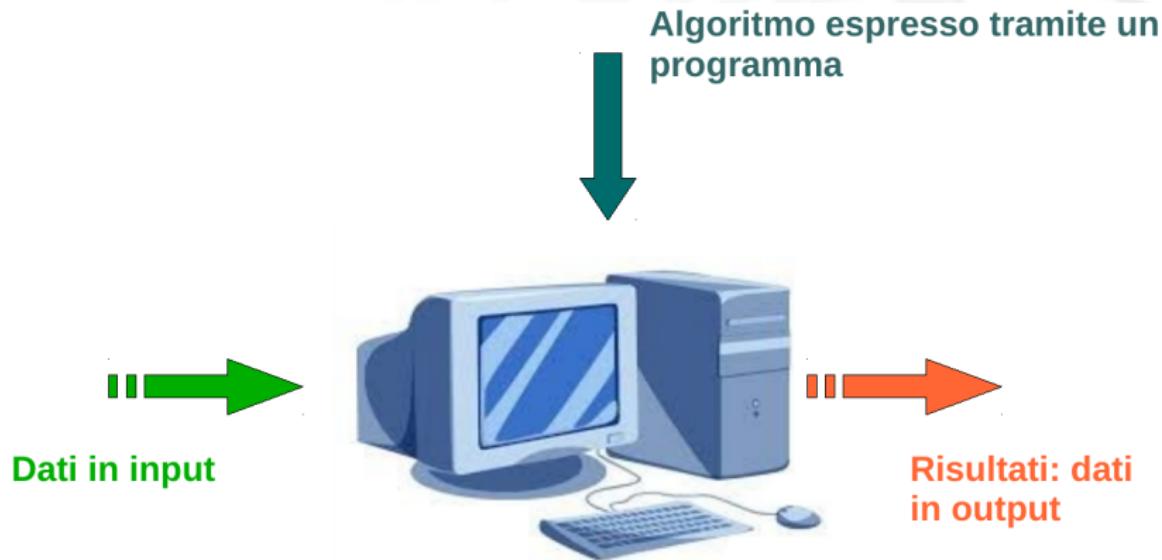
## Linguaggi e paradigmi di programmazione (3 anno)

- **Programmazione orientata agli oggetti:** Un programma descrive la dinamica di una collezione di oggetti - esemplari di **classi** - che comunicano scambiandosi messaggi (- invocando **metodi**).

## Programmazione 2

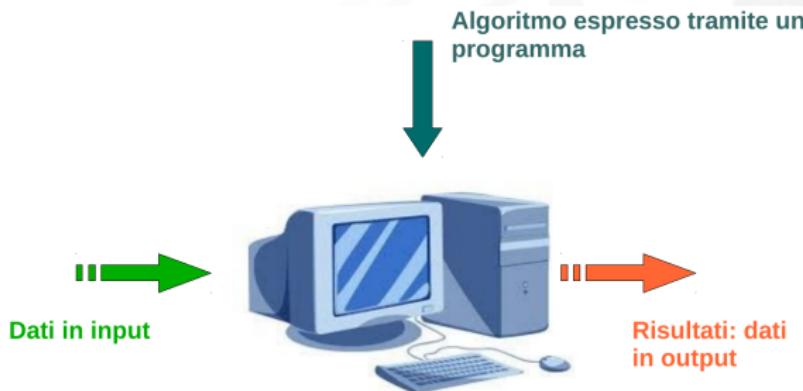
# Il calcolatore: versione “introversa”

- Il calcolatore:



# Il calcolatore: versione “introversa”

- Il calcolatore:



- HARDWARE** {
- congegni per la comunicazione dei dati (periferiche);
  - strumenti fisici per l'elaborazione (registri, bus dei dati e degli indirizzi, clock,...)
- SOFTWARE** {
- codifiche dei dati (bit, rappresentazione binaria dei numeri naturali, relativi e reali, rappresentazione dei caratteri:ASCII e UNICODE);
  - rappresentazione del flusso di controllo: programmi

## Il calcolatore: versione “estroversa”

- come fare comunicare i calcolatori in rete;
- come strutturare le reti di calcolatori;
- come realizzare la cooperazione e la risoluzione dei conflitti tra calcolatori che condividono risorse



# bit

- Segnale binario: segnale discreto su due valori;
- bit: binary digit (cifra binaria);
- Elemento di base per rappresentare le informazioni



- Perché il sistema binario → È semplice;
- Come viene realizzato un bit:
  - ▶ direzione di magnetizzazione;
  - ▶ presenza/ assenza di corrente/tensione;
  - ▶ passaggio/non passaggio di luce.
  - ▶ ...

# Rappresentazione dell'informazione con bit

- Un bit rappresenta 2 possibili informazioni

Es.: si/no, on/off, su/giù, vero/falso

- Combinando più bit si rappresentano più informazioni.

Es.: 2 bit possono rappresentare 4 informazioni

Un esame con 4 possibili esiti: insufficiente (00), sufficiente (01), buono (10), ottimo (11)

La corrispondenza concetto/configurazione di bit è una convenzione!

# Rappresentazione dell'informazione con bit

- Con 1 bit si rappresentano 2 informazioni ( $2^1$ )
- Con 2 bit si rappresentano 4 informazioni ( $2^2$ )
- Con 3 bit si rappresentano 8 informazioni ( $2^3$ )
- ...
- Con N bit si rappresentano  $2^N$  informazioni

Potenza	Valore
$2^0$	1
$2^1$	2
$2^2$	$2*2 = 4$
$2^3$	$2*2*2 = 8$
$2^4$	$2*2*2*2 = 16$
$2^5$	$2*2*2*2*2 = 32$
$2^6$	$2*...*2 = 64$
$2^7$	$2*...*2 = 128$
$2^8$	$2*...*2 = 256$
...	...

# Rappresentazione dell'informazione con bit

- Per rappresentare  $K$  informazioni, si deve utilizzare un numero di bit sufficiente per esprimerle tutte, per cui devo scegliere  $N$  in modo che

$$2^N \geq K$$

- Per rappresentare 61 informazioni diverse si devono usare  $N$  bit tali che:

$$2^N \geq 61$$

5 bit non sono sufficienti, infatti

$$2^5 = 32 < 61$$

Occorrono almeno 6 bit, infatti

$$2^6 = 64 \geq 61$$

# Byte

- È stato attribuito un significato particolare ai gruppi di 8 bit;
- byte(B) viene utilizzato - insieme al bit - come unità di misura

$$1\text{Byte} = 8\text{bit} = 2^8 = 256 \text{ informazioni diverse}$$

per esprimere la capacità della memoria, la potenza di un calcolatore, la velocità di trasmissione di una linea

# Unità di misura bit

Tra parentesi la nomenclatura standard ma meno usuale

Valore	Nome	Abbreviazione	Potenza
1	bit	b	$2^0$
1.024	Kilobit (kibibit)	Kb (Kib)	$2^{10}$
1.048.576	Megabit (Mebibit)	Mb (Mib)	$2^{20}$
1.073.741.824	Gigabit (Gibibit)	Gb (Gib)	$2^{30}$
1.099.511.627.776	Terabit (Tebibit)	Tb (Tib)	$2^{40}$

# Unità di misura byte

Valore	Nome	Abbreviazione	Potenza
1	byte	B	$2^0$
1.024	Kilobyte (kibibyte)	KB (KiB)	$2^{10}$
1.048.576	Megabyte (Mebibyte)	MB (MiB)	$2^{20}$
1.073.741.824	Gigabyte (Gibibyte)	GB (GiB)	$2^{30}$
1.099.511.627.776	Terabyte (Tebibyte)	TB (TiB)	$2^{40}$