Linguaggi di Programmazione: Storia ed Evoluzione

24/9/2011

Michele DI SANTO

Una premessa importante …

L'informatica è la
"scienza dell'elaborazione (automatica)
dell'informazione"

Dal francese: *Information automatique*
(P. Dreyfus, 1962)

« Computer science is no more about computers than astronomy is about telescopes. »

Edgser W. Dijkstra
(1930–2002)

Introduzione ai linguaggi di programmazione

- ✓ **Che cosa sono i linguaggi di programmazione?**
- ✓ **Perché è importante studiare i linguaggi di programmazione?**
- ✓ **Quanti sono i linguaggi di programmazione esistenti?**
- ✓ **Per quali motivi esistono così tanti linguaggi di programmazione?**
- ✓ **Quali sono i linguaggi di programmazione più popolari?**
- ✓ **Come è possibile classificare i linguaggi e cosa sono i paradigmi di programmazione?**

Che cosa sono i linguaggi di programmazione?

"... there is no agreement on what a programming language really is and what its main purpose is supposed to be.

Is a programming language a tool for instructing machines?

A means of communicating between programmers?

A vehicle for expressing high-level designs?

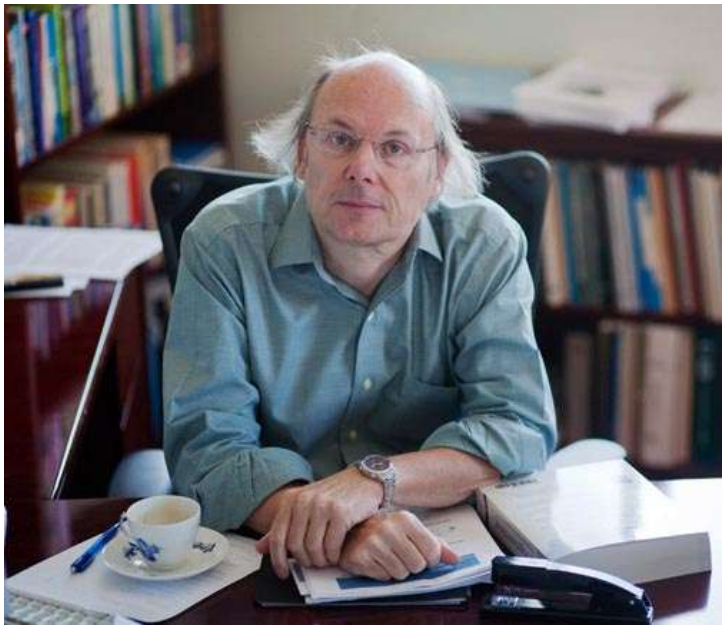
A notation for algorithms?

A way of expressing relationships between concepts?

A tool for experimentation?

A means of controlling computerized devices?

My view is that a general-purpose programming language must be all of those to serve its diverse set of users."

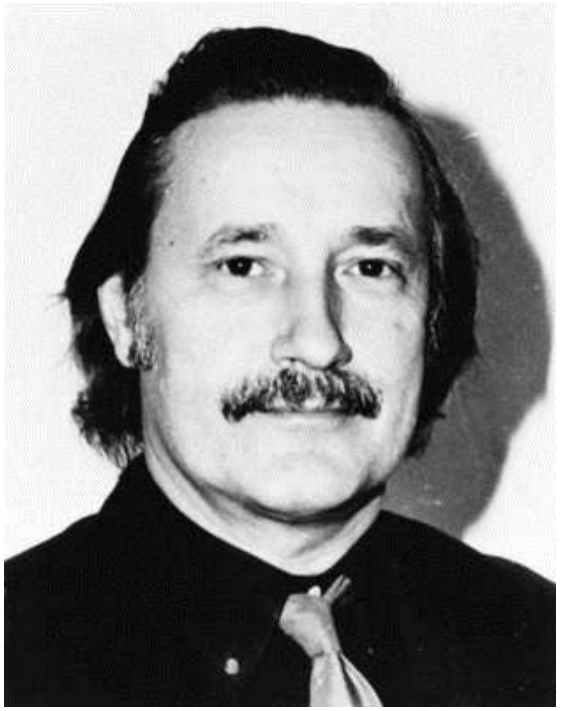


Bjarne Stroustrup (1950-),
The Design and Evolution of C++

Perché è importante studiare i linguaggi di programmazione? (1)

"I always worked with programming languages because it seemed to me that until you could understand those, you really couldn't understand computers.

Understanding them doesn't really mean only being able to use them. A lot of people can use them without understanding them."



Christopher Strachey
(1916-1975)

"The tools we use have a profound (and devious!) influence on our thinking habits, and therefore, on our thinking abilities."

Edsger Wybe Dijkstra (1930-2002),
How do we tell truths that might hurt?

Perché è importante studiare i linguaggi di programmazione? (2)

- I linguaggi influenzano il nostro modo di pensare e danno forma ai nostri pensieri
  - Nella sua forma più radicale, questa è la **controversa ipotesi linguistica**, detta di "Sapir-Whorf", secondo cui la visione del mondo di un individuo è sostanzialmente determinata dal vocabolario e dalla sintassi del proprio linguaggio
- Le caratteristiche del linguaggio di programmazione possono avere una profonda influenza sul modo di impostare la soluzione di un problema
  - Certi costrutti linguistici facilitano modi di pensare che conducono più "naturalmente" a soluzioni utili (ad esempio, la ricorsività)
    - Anche se non è vero che l'uso di un particolare linguaggio può impedire la risoluzione di un dato problema

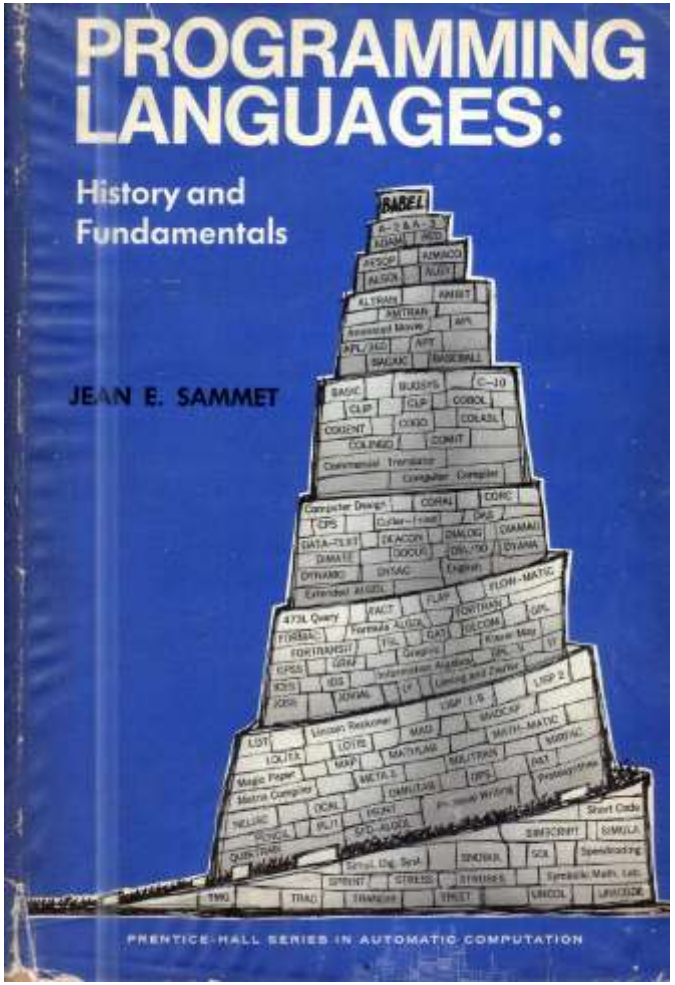
Perché è importante studiare i linguaggi di programmazione? (3)

- La conoscenza dei principi che sono alla base dei moderni linguaggi di programmazione e dei relativi aspetti implementativi
  - **Agevola l'apprendimento** di nuovi linguaggi
  - **Consente di scegliere il linguaggio** di volta in volta più appropriato al problema
  - **Consente di fare un uso migliore e più efficiente** di un dato linguaggio e aiuta a predirne i costi
  - **Migliora la capacità di progettare** nuovi linguaggi
    - Praticamente ogni programma ha un'interfaccia utente che è in effetti un linguaggio
      - Il progettista dell'interfaccia si trova ad affrontare problemi che s'incontrano nella progettazione dei linguaggi di programmazione

Quanti sono i linguaggi di programmazione esistenti? (1)

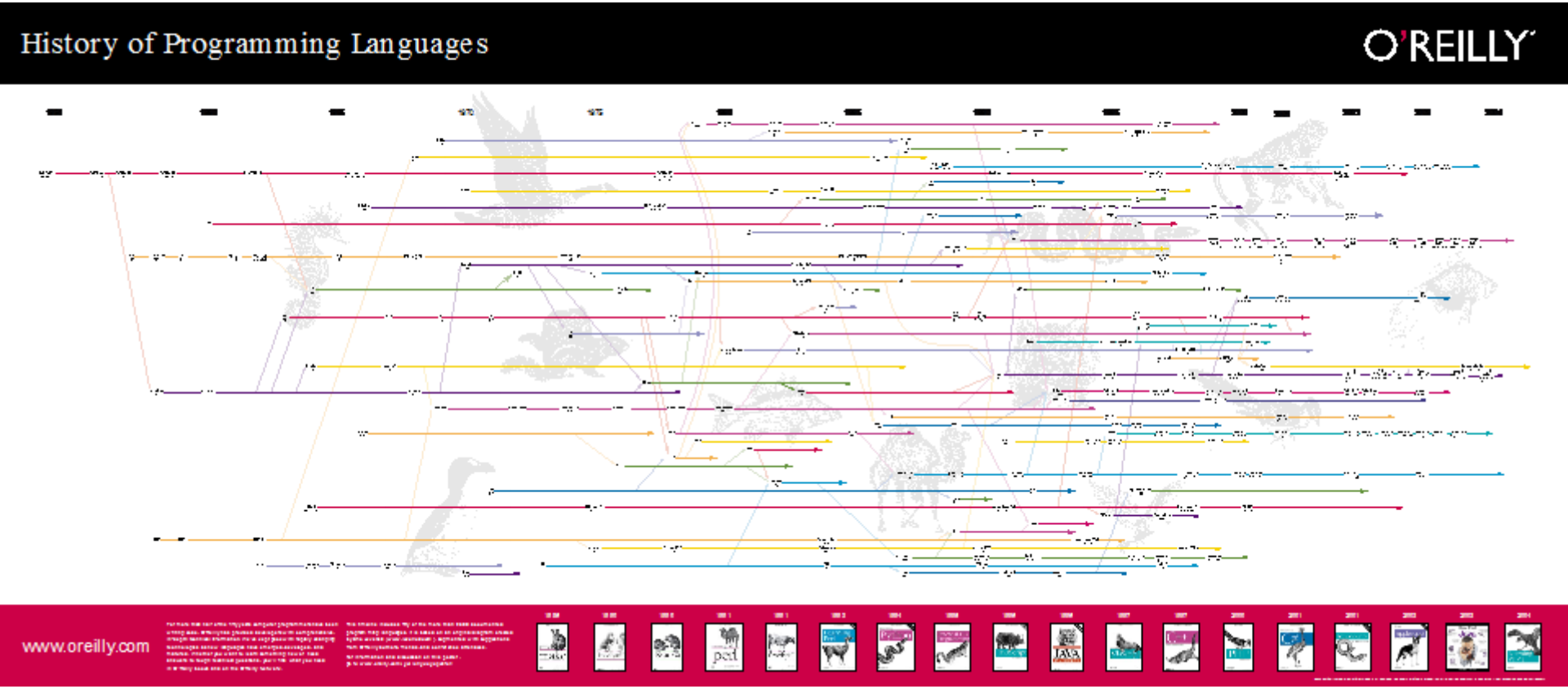
“La torre di Babele”,

Peter Bruegel - 1563



Jean Sammet, “Computer Languages: History and Fundamentals” - 1969

Quanti sono i linguaggi di programmazione esistenti? (2)



HOPL: an interactive Roster of Programming Languages (2006)

http://hopl.murdoch.edu.au/

Elenca 8512 linguaggi di programmazione

Per quali motivi esistono così tanti linguaggi di programmazione? (1)

Una lingua riempie una cella nell'alveare delle percezioni e delle interpretazioni possibili. Articola una gerarchia di valori, di significati e di supposizioni che non corrisponde esattamente a quella di nessun altra lingua. [...]

Parlando, creiamo mondi.

[G. Steiner, Errata, *cap. 7, 1997]*

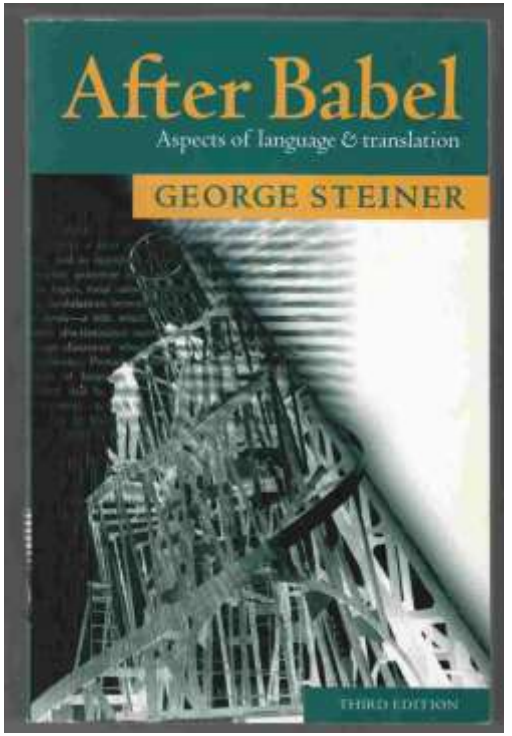
Babele è stata il contrario di una maledizione. Il dono delle lingue non è una metafora vuota, è proprio un dono e una benedizione immensa.

[G. Steiner, Errata, *cap. 7, 1997]*

Parlando, creiamo mondi.

[G. Steiner, Errata, *cap. 7, 1997]*

Per quali motivi esistono così tanti linguaggi di programmazione? (2)



George Steiner (Parigi, 1929) è figura di primo piano della cultura internazionale

- ✓ **Importante critico letterario comparatista, ha insegnato a Cambridge, Ginevra, Oxford, Harvard, Stanford, Princeton**
- ✓ **Perfettamente trilingue, ha studiato a fondo la natura del linguaggio e i problemi della traduzione, anche all'interno della stessa lingua**

Per quali motivi esistono così tanti linguaggi di programmazione? (3)

A good programming language is a conceptual universe for thinking about programming.

A language that doesn't affect the way you think about programming, is not worth knowing.

There will always be things we wish to say in our programs that in all known languages can only be said poorly.

Alan J. Perlis, famoso per i suoi aforismi, è stato un grande pioniere

- ✓ **Primo direttore del primo dipartimento di informatica (quello di Carnegie-Mellon University)**
- ✓ **Primo presidente della ACM (Association for Computing Machines)**
- ✓ **Primo premio Turing (1966)**

Per quali motivi esistono così tanti linguaggi di programmazione? (4)

Quali sono i linguaggi di programmazione più popolari? (1)

L'indice TIOBE è un indicatore di popolarità dei linguaggi di programmazione

"The Importance Of Being Earnest" **+"<language> programming"**

Position Sep 2011	Position Sep 2010	Delta in Position	Programming Language	Ratings Sep 2011	Delta Sep 2010
1	1	=	Java	18.761%	+0.85%
2	2	=	C	18.002%	+0.86%
3	3	=	C++	8.849%	-0.96%
4	6	↑↑	C#	6.819%	+1.80%
5	4	↓	PHP	6.596%	-1.77%
6	8	↑↑	Objective-C	6.158%	+2.79%
7	5	↓↓	(Visual) Basic	4.420%	-1.38%
8	7	↓	Python	4.000%	-0.58%
9	9	=	Perl	2.472%	+0.03%
10	11	↑	JavaScript	1.469%	-0.20%
11	10	↓	Ruby	1.434%	-0.47%
12	12	=	Delphi/Object Pascal	1.313%	-0.27%
13	24	↑↑↑↑↑↑↑↑↑↑	Lua	1.154%	+0.60%
14	13	↓	Lisp	1.043%	-0.04%
15	15	=	Transact-SQL	0.860%	+0.09%
16	14	↓↓	Pascal	0.845%	+0.06%
17	20	↑↑↑	PL/SQL	0.720%	+0.08%
18	19	↑	Ada	0.682%	+0.01%
19	17	↓↓	RPG (OS/400)	0.666%	-0.05%
20	30	↑↑↑↑↑↑↑↑↑↑	D	0.609%	+0.20%

La posizione è definita contando gli hit dei più popolari motori di ricerca alla richiesta in verde

Position	Programming Language	Ratings
21	Assembly	0.590%
22	MATLAB	0.543%
23	F#	0.512%
24	SAS	0.504%
25	COBOL	0.471%
26	Logo	0.448%
27	Scheme	0.400%
28	R	0.385%
29	C shell	0.383%
30	Fortran	0.372%
31	ActionScript	0.370%
32	Go	0.358%
33	Scratch	0.327%
34	NXT-G	0.327%
35	Haskell	0.325%
36	ABAP	0.320%
37	Forth	0.317%
38	Erlang	0.315%
39	Visual Basic .NET	0.309%
40	Prolog	0.282%
41	APL	0.272%
42	PL/I	0.268%
43	ML	0.263%
44	Smalltalk	0.255%
45	MOO	0.236%
46	Alice	0.235%
47	Ladder Logic	0.223%
48	Q	0.216%
49	Awk	0.212%
50	Scala	0.209%

Quali sono i linguaggi di programmazione più popolari? (2)

Programming Language	Position Sep 2011	Position Sep 2006	Position Sep 1996	Position Sep 1986
Java	1	1	5	-
C	2	2	1	1
C++	3	3	2	5
C#	4	8	-	-
PHP	5	5	-	-
Objective-C	6	39	-	-
(Visual) Basic	7	4	3	6
Python	8	7	23	-
Perl	9	6	6	-
JavaScript	10	9	21	-
Lisp	14	14	13	3
Ada	18	19	10	2

Year	Winner
2010	Python
2009	Go
2008	C
2007	Python
2006	Ruby
2005	Java
2004	PHP
2003	C++

Quali sono i linguaggi di programmazione più popolari? (3)

Category	Ratings Sep 2011	Delta Sep 2010
Object-Oriented Languages	56.2%	+1.7%
Procedural Languages	37.7%	-2.5%
Functional Languages	4.3%	+0.5%
Logical Languages	1.8%	+0.3%

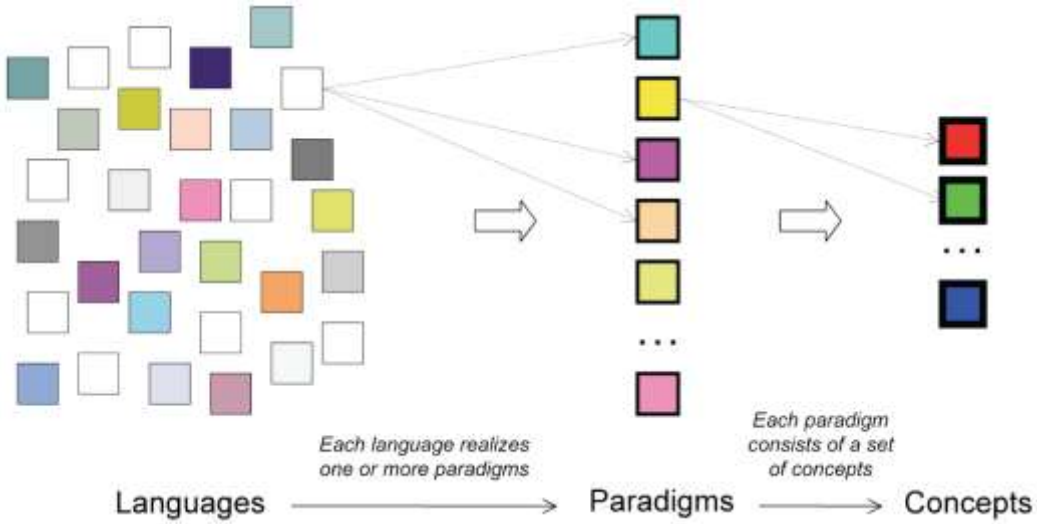
Category	Ratings Sep 2011	Delta Sep 2010
Statically Typed Languages	66.1%	+3.4%
Dynamically Typed Languages	33.9%	-3.4%

Come è possibile classificare i linguaggi e cosa sono i paradigmi di programmazione?

- I linguaggi di programmazione possono essere caratterizzati in base allo stile di programmazione che supportano, detto anche **paradigma di programmazione**
  - Un paradigma caratterizza il modo in cui il programma si presenta, come sono espresse le computazioni e come il programma è organizzato in parti
- I paradigmi dei linguaggi di programmazione possono essere distinti in due classi fondamentali
  - **Paradigmi computazionali**
    - Caratterizzano i modelli di calcolo definiti dai linguaggi
  - **Paradigmi strutturali**
    - Classificano i linguaggi in funzione delle modalità che essi offrono per dare una struttura ai programmi, scomponendoli in unità logiche

Ancora sui paradigmi di programmazione

- Ogni linguaggio di programmazione realizza uno o più paradigmi
- Ogni paradigma è definito da un insieme di concetti di programmazione
- I paradigmi computazionali e strutturali consentono di classificare i linguaggi di programmazione, e quindi di meglio coglierne le caratteristiche distintive principali



- Malgrado il numero dei paradigmi sia enormemente inferiore a quello dei linguaggi, **Peter Van Roy** dell'Université catholique de Louvain elenca 27 diversi paradigmi in uso

Programming Paradigms for Dummies: What Every Programmer Should Know
http://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf

Il paradigma computazionale imperativo (1)

- Il **paradigma computazionale** di gran lunga prevalente è quello **imperativo**, il cui modello di calcolo è il cosiddetto "***modello di Von Neumann***"
  - In altre parole, i linguaggi imperativi non sono altro che astrazioni della macchina di Von Neumann, il cui modello di calcolo consiste nell'esecuzione sequenziale, passo-passo, di istruzioni che modificano lo stato della memoria della macchina
    - I linguaggi imperativi operano su variabili, intese come astrazioni di celle di memoria il cui contenuto viene modificato nel corso dell'esecuzione, e forniscono istruzioni per l'esecuzione condizionale (tipo if…then…else) e per le iterazioni (tipo while…do …)
      - A questa classe appartengono linguaggi ormai obsoleti, ma tuttora in uso, come FORTRAN e COBOL, ma anche linguaggi più moderni come C, C++, Java e C#

Il paradigma computazionale imperativo (2)

- Più recentemente, si è passati dal modello di calcolo della singola macchina di Von Neumann a quello di un insieme di macchine di Von Neumann operanti in parallelo
  - Ciò è stato reso possibile, sia dalla definizione di meccanismi per "multi-programmare" i computer che dalla possibilità di interconnettere e coordinare più computer, realizzando sistemi fisicamente paralleli
    - Linguaggi moderni, come Java o C#, forniscono un supporto linguistico per la programmazione concorrente e distribuita
      - Consentono di definire sia più unità concorrenti che operano all'interno di un singolo nodo di elaborazione, sia unità che possono essere allocate su nodi diversi di una rete
      - Anche se concorrenza e distribuzione delle computazioni rende più complesso il modello computazionale complessivo di linguaggi tipo Java o C#, i singoli nodi dell'architettura soggiacente il linguaggio seguono sempre il paradigma di Von Neumann

Il paradigma computazionale funzionale

- Nel **paradigma computazionale funzionale** i programmi sono collezioni di definizioni e applicazioni di funzioni
  - Il calcolo effettuato da un programma è espresso come una funzione matematica
  - Le sue basi concettuali risiedono nella matematica, e in particolar modo nella teoria delle funzioni ricorsive
    - Funzioni "higher-order" e mancanza di effetti collaterali sono spesso associati a questi linguaggi
- I linguaggi funzionali consentono di esprimere in maniera sintetica ed elegante il procedimento di risoluzione di problemi anche complessi
  - Il capostipite dei linguaggi funzionali è il LISP
  - Altri linguaggi funzionali sono ML, Scheme, Haskell, F#

Il paradigma computazionale logico (1)

- Nel **paradigma computazionale logico** i programmi sono collezioni di formule espresse in un sottoinsieme del calcolo dei predicati del I ordine
- Anche in questo caso, si tratta dunque di un paradigma fortemente radicato nella matematica
- Il più noto linguaggio logico è il Prolog
- Il modello computazionale dei linguaggi logici si basa sulle regole logiche di inferenza per la loro valutazione

Il paradigma computazionale logico (2)

- Un programma logico non contiene istruzioni, ma dà informazioni su di un determinato contesto ed è normalmente visto come una base di conoscenza del sistema a cui siamo interessati
- Le informazioni sono specificate attraverso formule logiche che esprimono fatti e regole
  - I **fatti**
    - Sono asserzioni vere nel contesto che stiamo descrivendo
  - Le **regole**
    - Ci danno gli strumenti per dedurre nuovi fatti da quelli esistenti
- Un programma logico non si esegue ma si interroga
  - Ci si chiede: questi fatti sono veri?
  - Il Programma risponderà solo Sì o No alla domanda

I paradigmi strutturali

- I **paradigmi strutturali** classificano i linguaggi in funzione delle modalità che offrono per dare una struttura ai programmi, scomponendoli in unità logiche
- I principali paradigmi strutturali sono quello **procedurale** e quello **orientato agli oggetti** che prevedono entrambi la scomposizione dei programmi in parti
- Entrambi i paradigmi rispondono al classico principio ***"divide et impera"*** e costituiscono degli strumenti per la modularizzazione di un programma
  - La scomposizione in moduli fa sì che il programma complessivo risulti più facile da leggere, da scrivere e da manutenere
  - La possibilità di sviluppo separato e indipendente delle diverse parti consente di ridurre i tempi complessivi di sviluppo delle applicazioni

Il paradigma procedurale

- Secondo il **paradigma procedurale**, un programma è scomposto in **sottoprogrammi** che consentono di definire e dare nomi a operazioni complesse utilizzabili tutte le volte che occorra grazie all'operazione di chiamata a sottoprogramma
  - Tipici esempi sono le SUBROUTINE e FUNCTION del Fortran, le procedure del Pascal, le function del C
  - Per esempio, una funzione ***sort*** per ordinare un insieme di valori o una funzione ***draw_map*** per disegnare una mappa
- Nella modularizzazione che si ottiene risultano ben isolate le parti di programma che utilizzano le operazioni complesse da quelle che le definiscono
  - Eventuali modifiche interne a ciascun sottoprogramma non hanno riflessi imprevisti sulle altre parti del programma e possono essere da queste ignorate

Il paradigma orientato agli oggetti (1)

- Secondo il **paradigma orientato agli oggetti**, un programma è scomposto in **classi** di oggetti, ciascuna definita da un insieme di operazioni attraverso le quali gli oggetti della classe possono essere manipolati
  - Una classe raggruppa
    - La definizione della struttura dati che fornisce la rappresentazione concreta degli oggetti della classe
      - Il cui stato è generalmente modificabile da parte di alcune operazioni della classe
    - Le definizioni dei sottoprogrammi che implementano le operazioni
      - Inclusa quella che consente di creare un numero qualsiasi di oggetti della classe
  - Una classe si comporta come un tipo di dato dei convenzionali linguaggi procedurali
    - Pertanto, il costrutto di classe consente di definire **tipi di dato astratti**

Il paradigma orientato agli oggetti (2)

- I vantaggi in termini di strutturazione e modularità dei linguaggi object-oriented rispetto ai linguaggi procedurali sono notevoli
  - La possibilità di associare una struttura di dati alle operazioni che ne consentono la manipolazione e di renderla invisibile all'esterno del costrutto di classe consente di ottenere sia una più razionale suddivisione in moduli che una più facile modificabilità dei programmi
    - Eventuali modifiche della struttura di dati che realizza lo stato degli oggetti non hanno effetto sui moduli clienti
      - La modifica della struttura di dati resta nascosta (***incapsulata***) all'interno del costrutto di classe

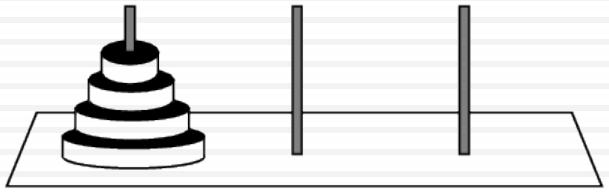
Il paradigma orientato agli oggetti (3)

- Un ulteriore potente strumento linguistico a supporto della evoluzione del software è il meccanismo di **sottoclasse** con cui è possibile definire **classi derivate** di oggetti che specializzano il comportamento della classe originaria
  - La sottoclasse definisce solo le nuove funzionalità che gli oggetti della sottoclasse forniscono, in quanto tutte le altre risultano ***ereditate*** dalla classe
  - La sottoclasse può anche ridefinire alcune delle funzionalità fornite dalla classe
  - I moduli di programma esistenti continuano ad operare correttamente anche se gli oggetti su cui operano sono istanze delle classi derivate
    - Tecnicamente, questo importante risultato è dovuto a due caratteristiche dei linguaggi object-oriented: il **binding dinamico** e il **polimorfismo**

Altre forme di classificazione

- A seconda
  - Dello stile **testuale** o **visuale** mediante il quale si definiscono i programmi nel linguaggio
  - Della modalità prevalente **compilata** o **interpretata** con cui il linguaggio è implementato
    - Tra i linguaggi interpretati vi sono quelli di script (Python, Ruby)
    - La maggior parte dei linguaggi sono compilati
      - Sono però comuni situazioni ibride come quella di Java, che di solito è compilato in un linguaggio intermedio, detto Bytecode, che è poi interpretato
  - Della modalità di tipizzazione **statica** o **dinamica** che consente di distinguere i linguaggi in cui ogni variabile ha associato in maniera statica un tipo rispetto a quelli in cui questo tipo può cambiare dinamicamente
    - I linguaggi della prima categoria supportano una maggiore affidabilità dei programmi, potendo garantire a priori l'assenza di certi errori durante l'esecuzione

```
void MoveTower(int n, char start, char finish, char temp) {
   if (n == 1) MoveSingleDisk(start, finish);
   else      { MoveTower(n - 1, start, temp, finish);
               MoveSingleDisk(start, finish);
               MoveTower(n - 1, temp, finish, start); }
}
```



I precursori dei linguaggi di programmazione

- ✓ Algoritmi e linguaggi di programmazione
- ✓ L'origine della parola algoritmo e i primi algoritmi noti
- ✓ L'importanza attuale degli algoritmi
- ✓ Dall'antichità al '700
- ✓ La rivoluzione industriale e le macchine: il telaio Jacquard
- ✓ Charles Babbage e la Analitycal Engine
- ✓ Ada Augusta Byron, lady Lovelace
- ✓ La nascita dell'informatica "moderna": Alan Turing e la sua macchina
- ✓ La realizzazione del calcolatore universale

Algoritmi e Linguaggi di programmazione

- Centrale all'informatica è lo studio dei **procedimenti di calcolo per risolvere problemi**
  - Sia quelli eseguiti dall'uomo che quelli eseguiti automaticamente dagli strumenti di calcolo
- In informatica, i procedimenti di calcolo sono comunemente denominati **algoritmi**

Un algoritmo è una procedura di calcolo atta a risolvere un determinato problema, descritta in modo preciso

- Poiché i linguaggi di programmazione sono anche una *notazione per esprimere **algoritmi***, possiamo certamente dire che i primi linguaggi di programmazione, così come i primi algoritmi, risalgono agli albori della civiltà

L’origine della parola algoritmo (1)

- La parola **algoritmo** ha origine nel Medio Evo e proviene dal nome latinizzato del matematico persiano del IX secolo

MUHAMMAD IBN MŪSĀ AL-KHWĀRIZMĪ

L’origine della parola algoritmo (2)

- Dal titolo del suo principale libro

  AL-KITĀB AL-MUKHTAṢAR FĪ HĪSĀB AL-ĞABR WA’L-MUQĀBALA

  deriva la parola **algebra**

L’origine della parola algoritmo (3)

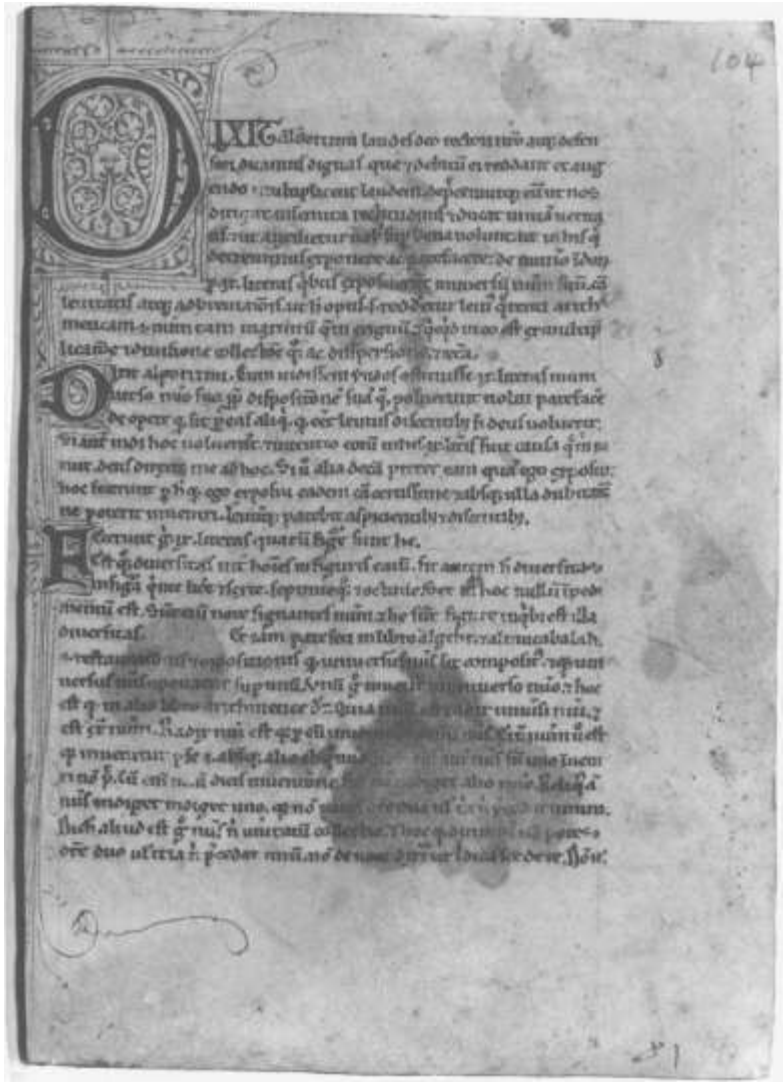
- L’opera Kitāb al-Jam’ wa-l-Tafrīq bi-l hisāb al-Hind,

  in cui veniva descritto il **sistema posizionale decimale inventato dagli indiani**, venne tradotta in latino attorno al 1120 con il titolo

  Algoritmi de numero indorum

- La traduzione latina iniziava così:

  Dixit Algorizmi….



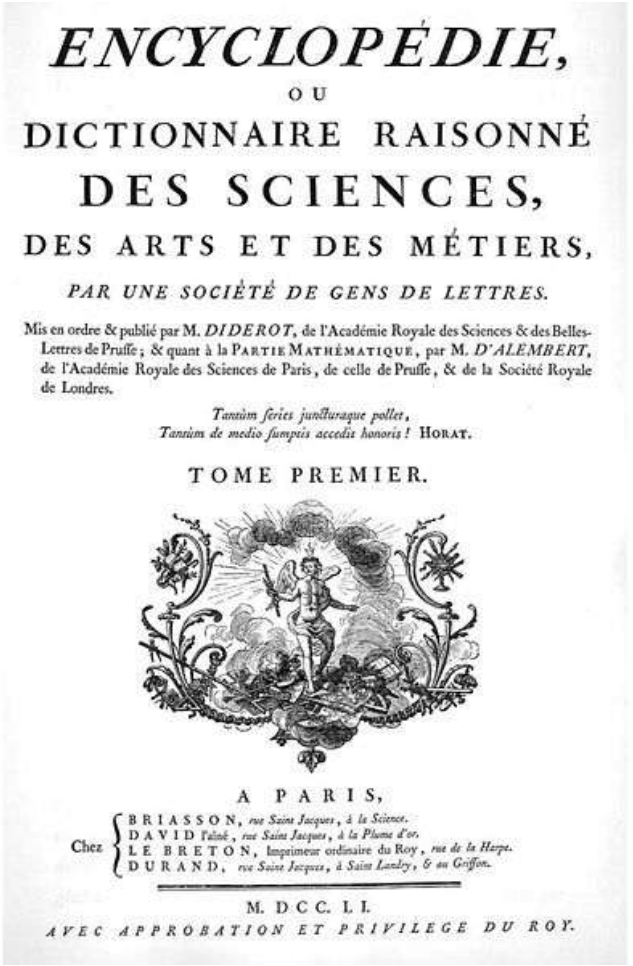
L'origine della parola algoritmo (4)

- A poco a poco, la parola **algorismus** cominciò ad essere usata per indicare i procedimenti di calcolo delle operazioni aritmetiche nel sistema decimale
  - In contrapposizione al sistema di numerazione romano e all'uso dell'abaco (vedi competizione tra algorista e abacista)

- Tale uso portò quasi a dimenticare il collegamento tra il termine algorismus e il matematico arabo
  - Collegamento che fu riscoperto solo nella metà del XIX secolo, quando le opere del matematico arabo furono ritrovate dopo un lungo oblio

L'origine della parola algoritmo (5)

- Successivamente, il termine algoritmo ha finito per indicare non solo i metodi per eseguire le quattro operazioni aritmetiche, ma un qualunque procedimento atto a risolvere un problema di calcolo

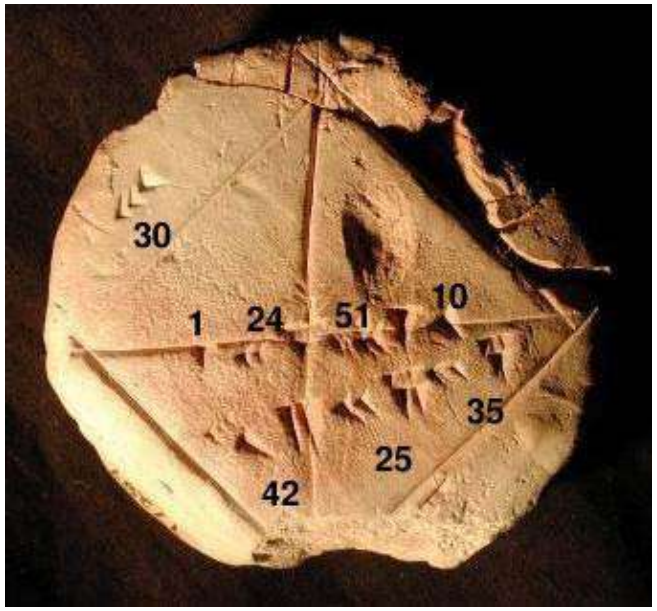

ENCYCLOPÉDIE,
OU
DICTIONNAIRE RAISONNÉ
DES SCIENCES,
DES ARTS ET DES MÉTIERS,
PAR UNE SOCIÉTÉ DE GENS DE LETTRES.
Mis en ordre & publié par M. DIDEROT, de l'Académie Royale des Sciences & des Belles-Lettres de Prusse; & quant à la PARTIE MATHÉMATIQUE, par M. D'ALEMBERT, de l'Académie Royale des Sciences de Paris, de celle de Prusse, & de la Société Royale de Londres.
Tantùm series juncturaque pollet,
Tantùm de medio sumptis accedit honoris! HORAT.
TOME PREMIER.
A PARIS,
Chez BRIASSON, rue Saint Jacques, à la Science.
DAVID l'aîné, rue Saint Jacques, à la Plume d'or.
LE BRETON, Imprimeur ordinaire du Roy, rue de la Harpe.
DURAND, rue Saint Jacques, à Saint Landry, & au Griffon.
M. DCC. LI.
AVEC APPROBATION ET PRIVILEGE DU ROY.

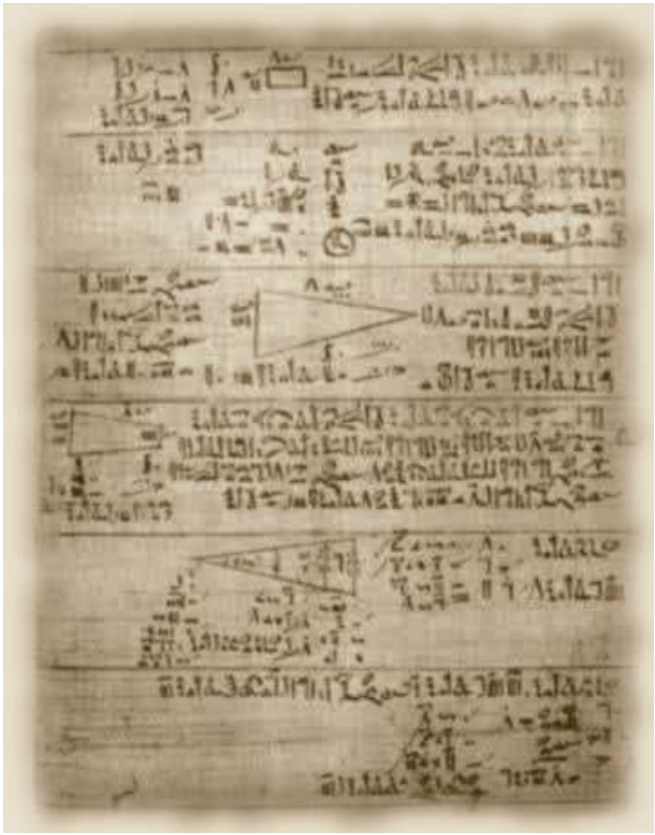
- Diderot e d'Alembert definiscono così il termine nell'Encyclopédie (1751-1772):

Termine arabo, usato da molti autori, e particolarmente dagli spagnoli per indicare la pratica dell'algebra. Può anche indicare l'aritmetica delle cifre … La stessa parola può in generale significare il metodo e la notazione per tutti i tipi di calcolo. In questo senso, parliamo di algoritmo del calcolo integrale, di algoritmo del calcolo esponenziale, di algoritmo dei seni, ecc.

I primi algoritmi (1)

- I primi algoritmi di cui si abbia traccia certa risalgono ai babilonesi e agli egiziani (2000-1660 a. C.)
- Caratteristiche di questi algoritmi sono:
  - L'illustrazione mediante sequenze di esempi
  - L'assoluta mancanza di dimostrazioni



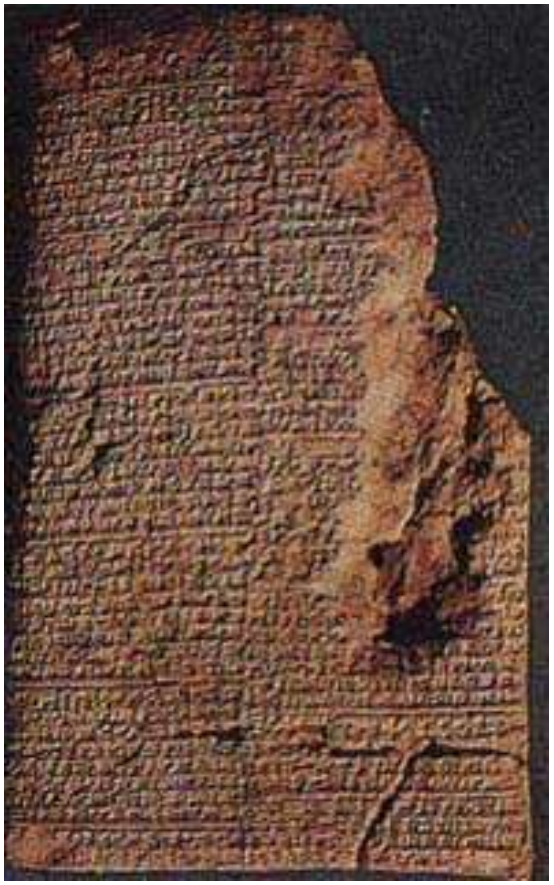


D. E. Knuth, Ancient Babylonian algorithms, Twenty-fifth anniversary of the Association for Computing Machinery, Comm. ACM 15 (7) (1972), 671-677.

I primi algoritmi (2)

- Tavolette babilonesi forniscono algoritmi nella forma di ripetuti esempi di sequenze di operazioni
  - Dalla sequenza di esempi si può inferire l'algoritmo
- Ogni esempio è strutturato come segue:
  - Il numero è **x** (Sia **x=y+z**)
  - Quale è il suo inverso?
  - Procedi come segue
    - Forma l'inverso di **y** troverai **y'**
    - Moltiplica **y'** per **z**, troverai **t**
    - Aggiungi **1**, troverai **u**
    - Forma l'inverso di **u** troverai **u'**
    - Moltiplica **u'** per **y'**. Troverai **v**
  - L'inverso è **v**. Questo è il modo di procedere

$$\frac{1}{x} = \frac{1}{y+z} = \frac{1}{z \cdot 1/y + 1} \cdot \frac{1}{y}$$



I primi algoritmi (3)

- Il **papiro di Rhind**, dal nome dell'antiquario scozzese che lo acquistò a Luxor nel 1858 riporta tra gli altri, il seguente algoritmo di moltiplicazione:
  - Per moltiplicare **83** per **37**:
    - Si scrivono su due colonne i numeri **1** e **37** e sotto di essi i numeri che si ottengono per successive moltiplicazioni per **2**, fino a non eccedere nella colonna di sinistra il numero **83**
    - Si cercano nella colonna di sinistra i numeri che sommati danno **83**
    - I numeri corrispondenti nella colonna di destra, sommati danno il prodotto

⟹	1	37
⟹	2	74
	4	148
	8	296
⟹	16	592
	32	1184
⟹	64	2368

37 + 74 + 592 + 2368 = 3071

L'algoritmo di Euclide

- Euclide di Alessandria (350-300 a.C.) ha concepito l'algoritmo per il calcolo del massimo comun divisore
  - *"Siano dati AB e CD non primi tra loro, si deve trovare la più grande misura comune di AB e CD*
    Se CD misura AB allora CD sarà la misura comune …
    Ma se CD non misura AB e se togliamo il minore tra AB e CD dal maggiore resterà qualche numero che misura ciò che rimane…"

```
function MCD(a, b)
   while a ≠ b
      if a > b
         a := a – b
      else
         b := b – a
   return a
```

```
function MCD(a, b)
   if b == 0
      return a
   else
      return mcd(b, a % b)
```

Il crivello di Eratostene (1)

- Eratostene di Alessandria (276-195 a.C.) ha concepito un algoritmo per la determinazione dei numeri primi compresi tra 2 ed n
  1. Considera la lista dei numeri da 2 a n
  2. Il primo numero in lista ancora non considerato è primo
  3. Elimina tutti i multipli di questo numero
     - L'eliminazione può avvenire a partire dal quadrato del numero, poiché i multipli più piccoli sono stati già cancellati
  4. Ripeti i passi 2 e 3 fino al primo numero maggiore della radice quadrata di n
  5. Tutti i numeri ancora presenti nella lista sono primi

Il crivello di Eratostene (2)

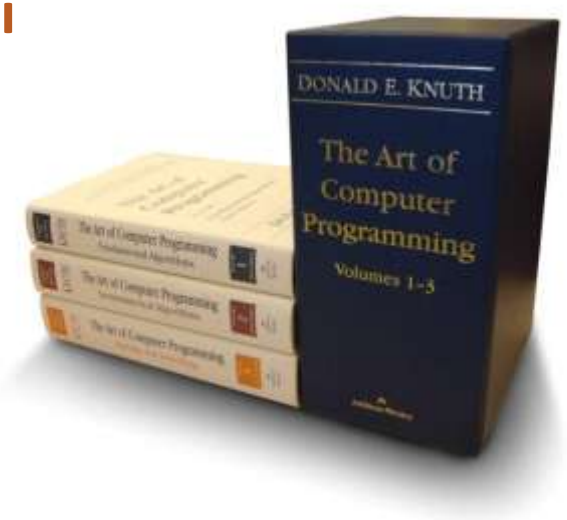
	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

L'importanza attuale degli algoritmi (1)

- **Donald E. Knuth**, uno dei massimi studiosi di informatica viventi ha scritto nel 1977

"Fino a dieci anni fa la parola algoritmo era sconosciuta alla maggior parte delle persone colte e, in verità, era scarsamente necessaria. Lo sviluppo rapidissimo dell'informatica, il cui punto focale è costituito dallo studio degli algoritmi, ha modificato la situazione: attualmente la parola è indispensabile."



L'importanza attuale degli algoritmi (2)

- Dall'uso di sofisticati algoritmi e dalla loro efficienza dipendono in modo cruciale
  - Le prestazioni dei calcolatori
  - La gestione e la comunicazione in reti complesse (Internet, P2P)

  e la quasi totalità delle moderne applicazioni informatiche, come ad esempio:
  - I motori di ricerca di informazioni nel Web
  - I sistemi di grafica interattiva (fondamentali nei videogiochi e nelle interfacce utente)
  - I sistemi di navigazione satellitare
  - I risolutori di complessi sistemi di ottimizzazione
  - Ecc.

L'importanza attuale degli algoritmi (3)

- Grazie alla pervasività delle applicazioni, anche nella percezione comune è ben presente il ruolo degli algoritmi, utilizzati
  - da un navigatore per indicarci un percorso ottimizzato
  - per comprimere e decomprimere informazioni multimediali
  - per analizzare e confrontare il DNA di due esseri umani o altri dati biometrici
  - ecc.
- Inoltre, da tempo, algoritmi sono inglobati in leggi (ad es. la legge che definisce il codice fiscale o la legge per l'adozione della firma digitale) e codici di identificazione, utilizzati da sistemi crittografici più o meno complessi

Dall'antichità al '700

- In un velocissimo excursus ricordiamo che in questi lunghi secoli i progressi fondamentali sono stati compiuti nella progettazione e realizzazione di **strumenti per l'automazione dei calcoli**

Abaco cinese **Calculating clock di W. Schickard (1623)** **Pascalina di B. Pascal (1623)**
Stepped Reckoner di G. W. Leibnitz (1671) **Aritmometro di T. de Colmar (1820)**

La rivoluzione industriale e le macchine

- Nel ’700 andarono maturando due idee importanti:
  - Le macchine possono essere dotate di una propria forza interna grazie ad un motore a vapore o altri dispositivi
  - Le macchine possono eseguire sequenze complesse di azioni grazie a opportuni meccanismi di controllo
- Su queste idee nasce la **macchina tessile** di **Joseph Marie Jacquard** (1752 – 1834)

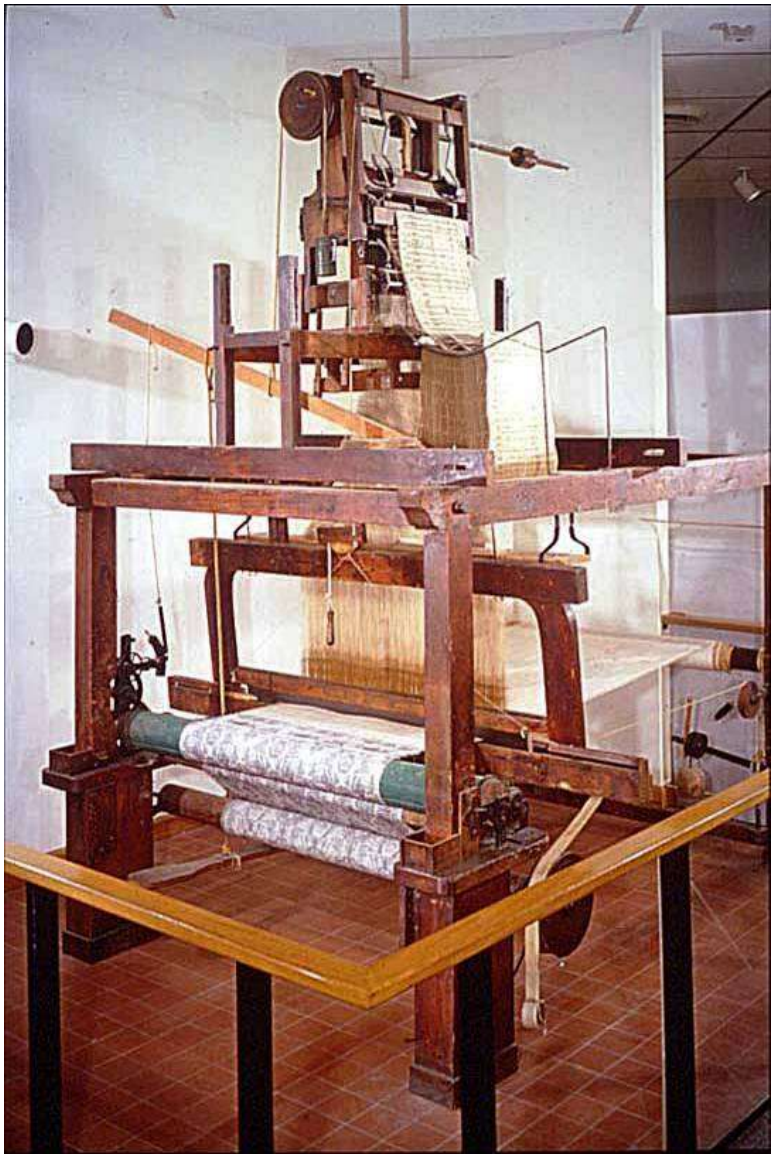
Il telaio Jacquard (1)

- È il primo esempio di macchina programmabile per il controllo di un processo (in questo caso la tessitura)
  - Le operazioni compiute dal telaio sono definite da un insieme di **schede perforate**
    - I fori nelle schede consentono di muovere degli uncini che alzano ed abbassano i fili dell'ordito
- Jacquard scrisse un programma composto da 24000 schede perforate per la tessitura di un arazzo con la sua immagine

Il telaio Jacquard (2)

[Jacquard-card Making.]

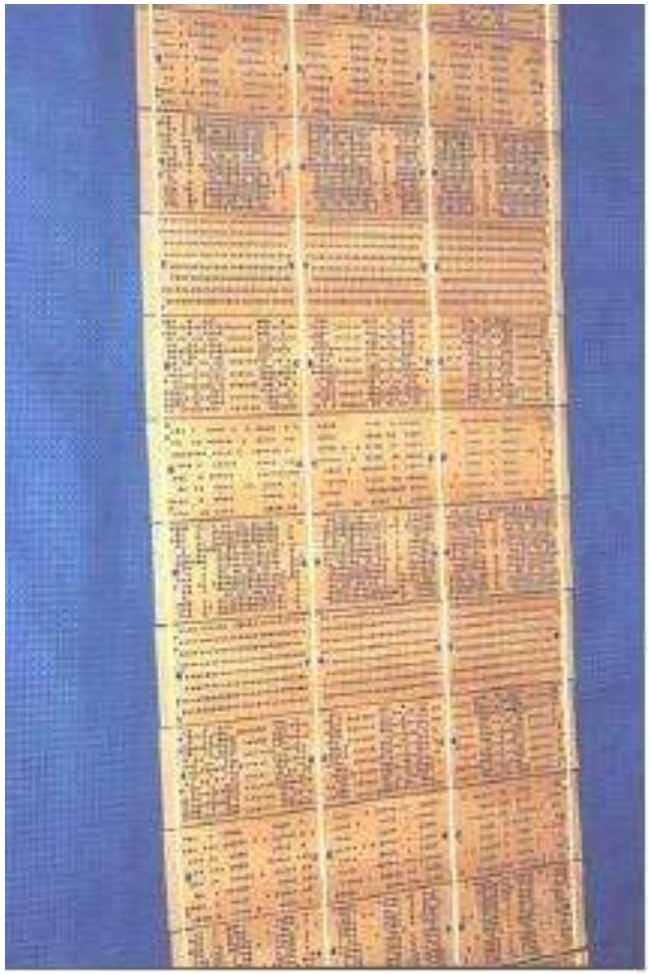
Il telaio Jacquard (3)



Il telaio Jacquard (4)

Primo esempio di software

(istruzioni su schede perforate)



Charles Babbage (1)

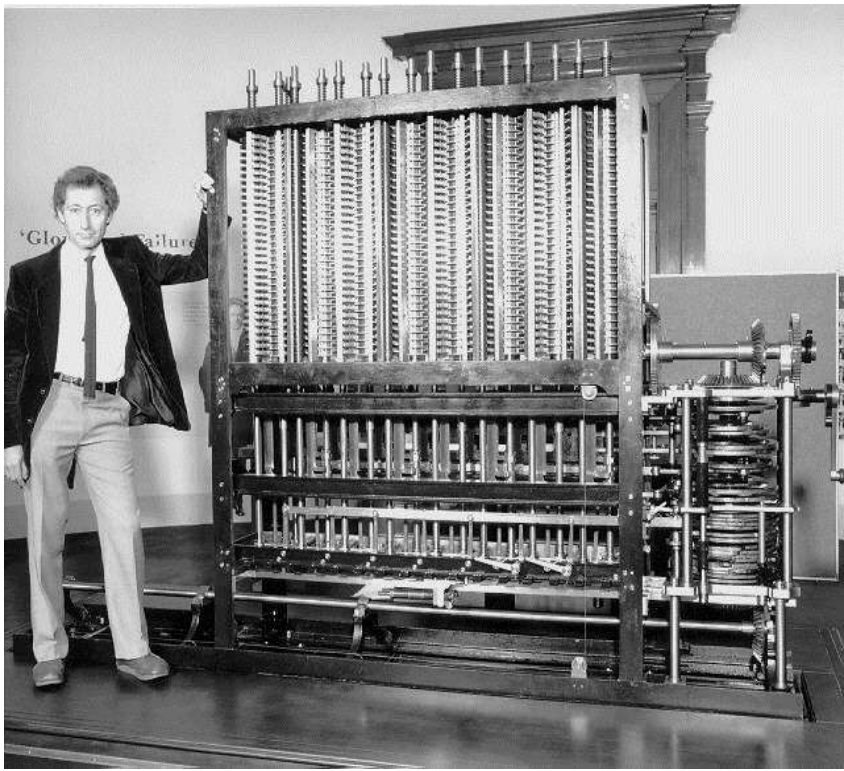
- **Charles Babbage** (1791-1871), inglese, matematico, filosofo, ingegnere meccanico può essere considerato **il padre**, o forse meglio il nonno, **del calcolatore moderno a programma memorizzato**
  - Tra l'altro, inventore del tachimetro

Charles Babbage (2)

- È il primo vero pioniere del moderno calcolo digitale
- Progettò e cercò di costruire i prototipi di due diverse macchine di calcolo
  - **Difference Engine**
  - **Analytical Engine**
- Non riuscì a portare a compimento i suoi progetti
  - Troppo in anticipo sui tempi, senza una valida tecnologia a disposizione

La Macchina Differenziale

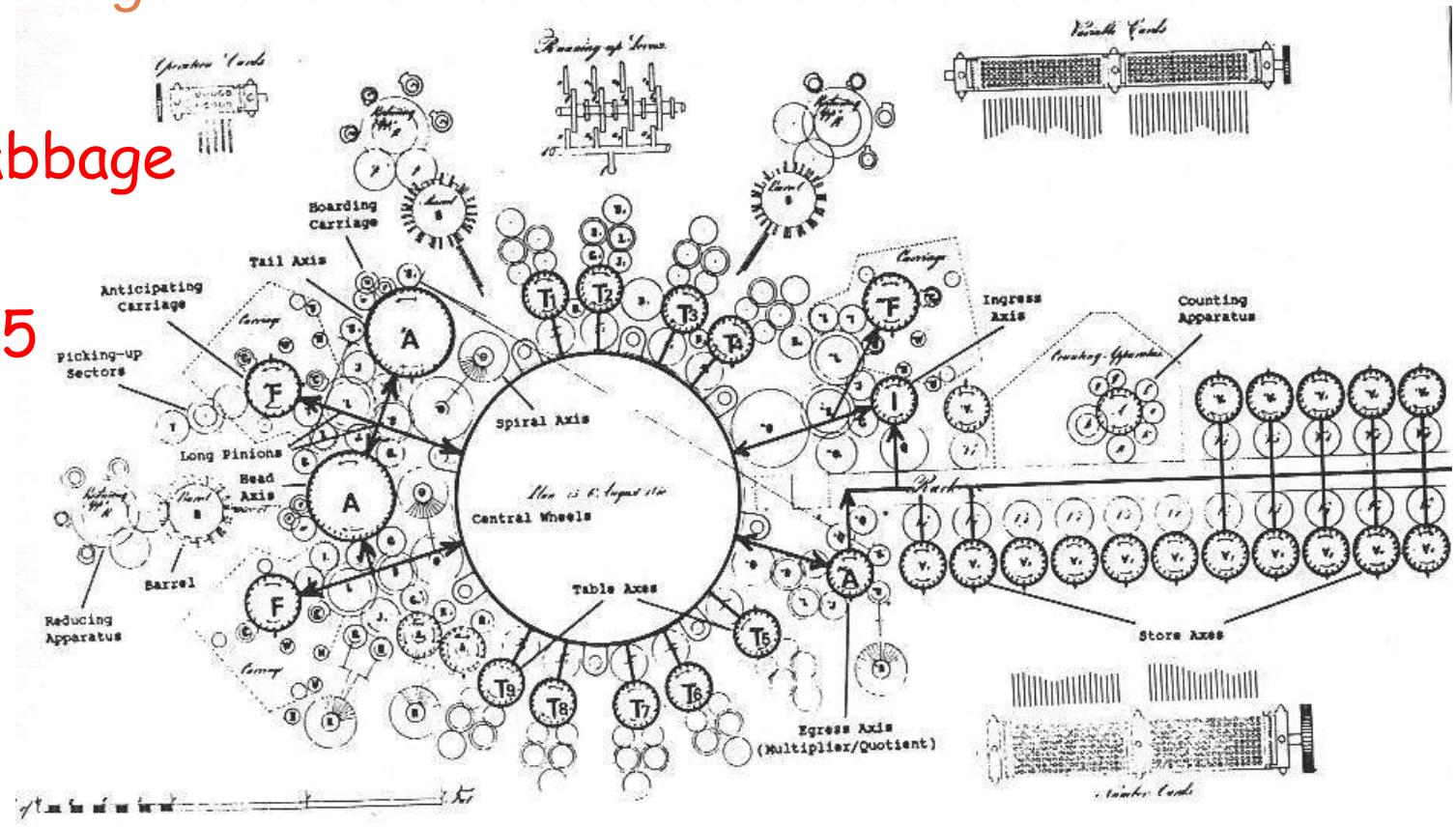
- La macchina differenziale, progettata verso il 1823, aveva la finalità di **calcolare tabelle di numeri utili per la navigazione**
  - Eseguiva il solo metodo delle **differenze finite**, per calcolare e tabellare i valori in più punti di una funzione polinomiale
- Negli anni 1989-91, allo Science Museum di Londra, in occasione del **bicentenario della nascita di Babbage**, è stata costruita una versione completa della Difference Engine, sulla base del progetto originale



La Macchina Analitica (1)

"...for six months been engaged in making the drawings of a new calculating engine of far greater power than the first. I am myself astonished at the power I have been enabled to give to this machine; a year ago I should not have believed this result possible."

Lettera di Babbage
a Quetelet,
27 aprile 1835



La Macchina Analitica (2)

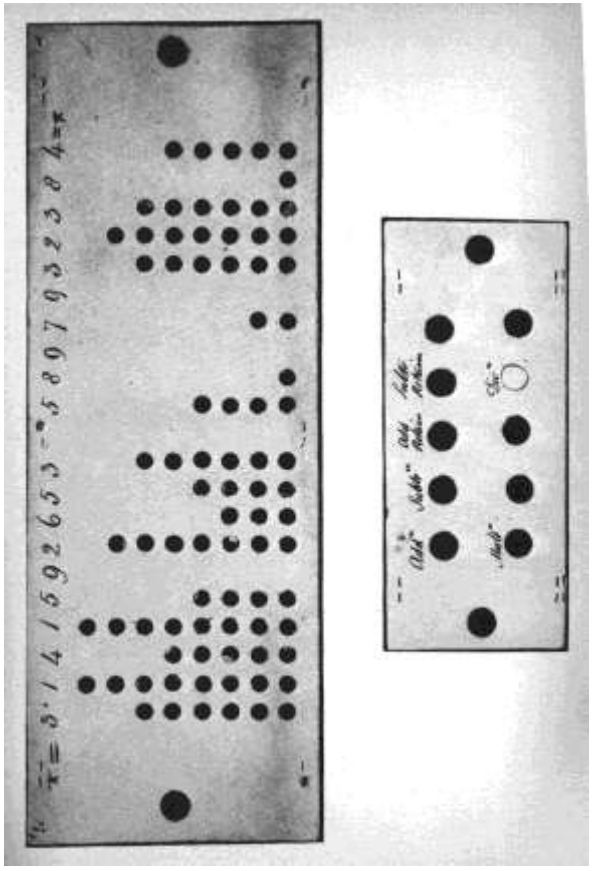
- L'**Analytical Engine** è il **primo vero computer programmabile**
  - È composto di 4 parti:
    - Il **magazzino** (the **store**, la memoria)
      - 1000 parole, ciascuna di 50 cifre decimali
    - Il **mulino** (the **mill**, l'unità di calcolo)
    - Il **dispositivo di input** (lettore di **schede perforate**)
    - I **dispositivi di output** (incisore di **piatti di rame**, perforatore di **schede perforate** e campanello)
- Il metodo di calcolo eseguito è variabile ed è letto da schede di cartone perforate
  - Può essere espresso usando vari tipi di istruzioni:
    - Istruzioni aritmetiche
    - Istruzioni per lo spostamento dati
    - Istruzioni di test e di salto condizionale

La Macchina Analitica (3)

"Forse chi conosce i principi sui quali si fonda il telaio Jacquard e ha una certa familiarità con le formule analitiche, non avrà molta difficoltà a formarsi un'idea generale dei mezzi con i quali la macchina esegue le operazioni"

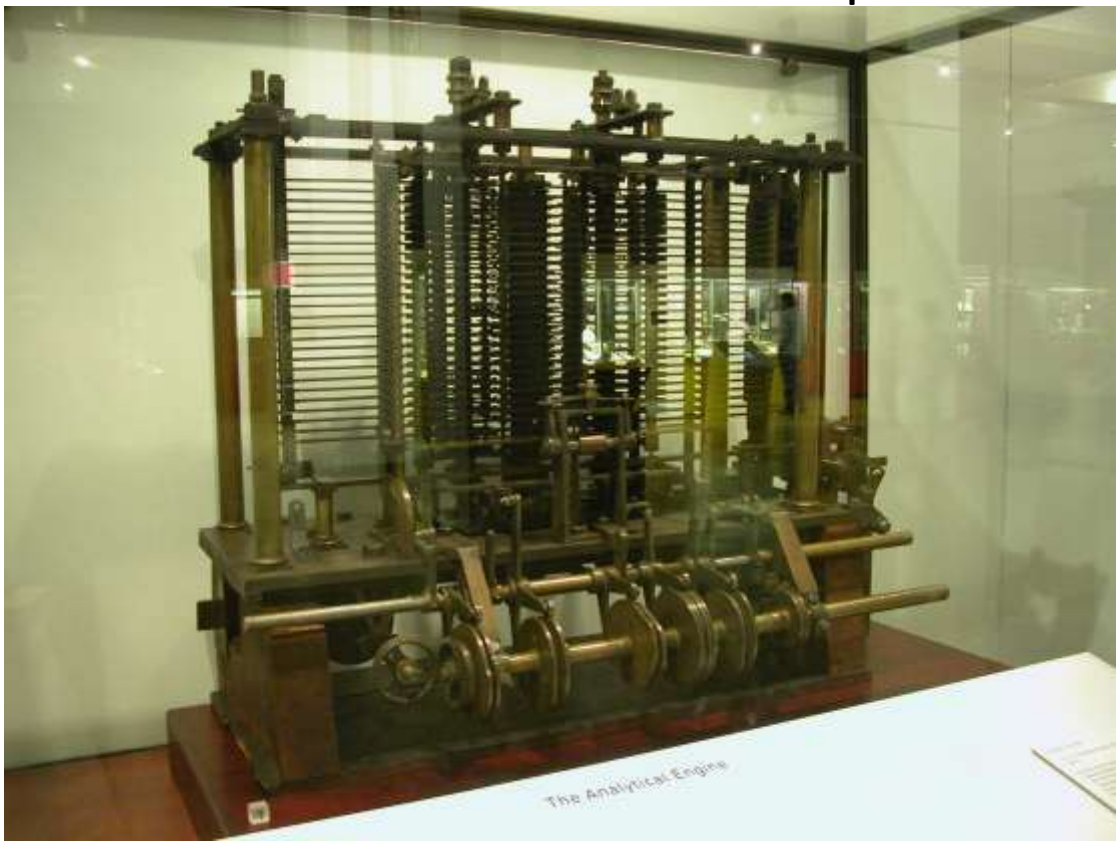
Charles Babbage:

"Passages from the life of a philosopher"



La Macchina Analitica (4)

- La macchina analitica fu dettagliatamente progettata, ma mai completamente portata a termine
  - Essa avrebbe dovuto essere alimentata da un motore a vapore ed avrebbe dovuto essere lunga 30 metri e larga 10
  - Nel 1847 una parte del mill e del printer sono completati
  - Nel 1858 Babbage torna a lavorare al progetto
  - Nel 1871, anno della sua morte, un modello di prova è pronto
    - Contiene 2 accumulatori di 25 cifre



Babbage e l'Italia

- Nel 1840 Babbage fu invitato da Giovanni Plana a Torino e presentò il suo progetto presso l'Accademia Reale delle Scienze
  - Alla presentazione partecipò anche un giovane ufficiale, **Luigi Federico Menabrea** (1809-1896) che poi sarebbe diventato primo Ministro del Governo italiano nel 1867
  - Menabrea stese una relazione sulla Macchina Analitica, pubblicata in francese a Ginevra
    - Questa versione della relazione è sostanzialmente la più importante fonte di informazioni sulla Macchina Analitica
  - La relazione di Menabrea fu poi tradotta in inglese e arricchita con molte note da **Ada Lovelace** (collaboratrice di Babbage)

Augusta Ada Byron (1)

- **Augusta Ada Byron** (1815-1852), **contessa di Lovelace**, inglese, figlia del poeta **Byron**, studiosa di matematica, fu una preziosa collaboratrice di Babbage
- Le note alla traduzione della relazione di Menabrea migliorano il testo e forniscono esempi di come la macchina analitica si sarebbe potuta usare per risolvere vari problemi

Augusta Ada Byron (2)

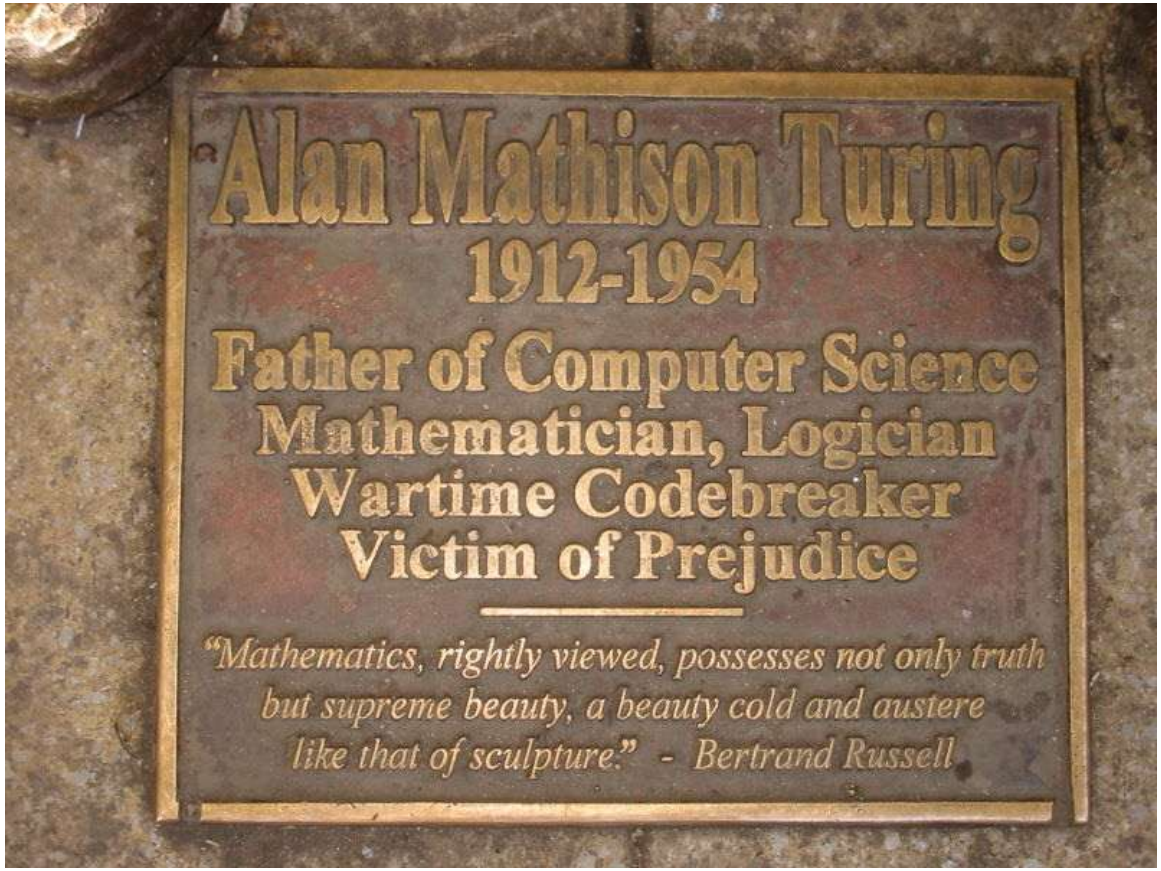
- Ada può essere considerato **il primo programmatore della storia**
  - In particolare, suggerì a Babbage un programma per il calcolo dei numeri di Bernoulli (successione di numeri razionali)
- Nel 1979, il Dipartimento della Difesa degli Stati Uniti ha onorato il ricordo di Ada battezzando **ADA** un nuovo linguaggio di programmazione

La nascita dell'informatica "moderna"

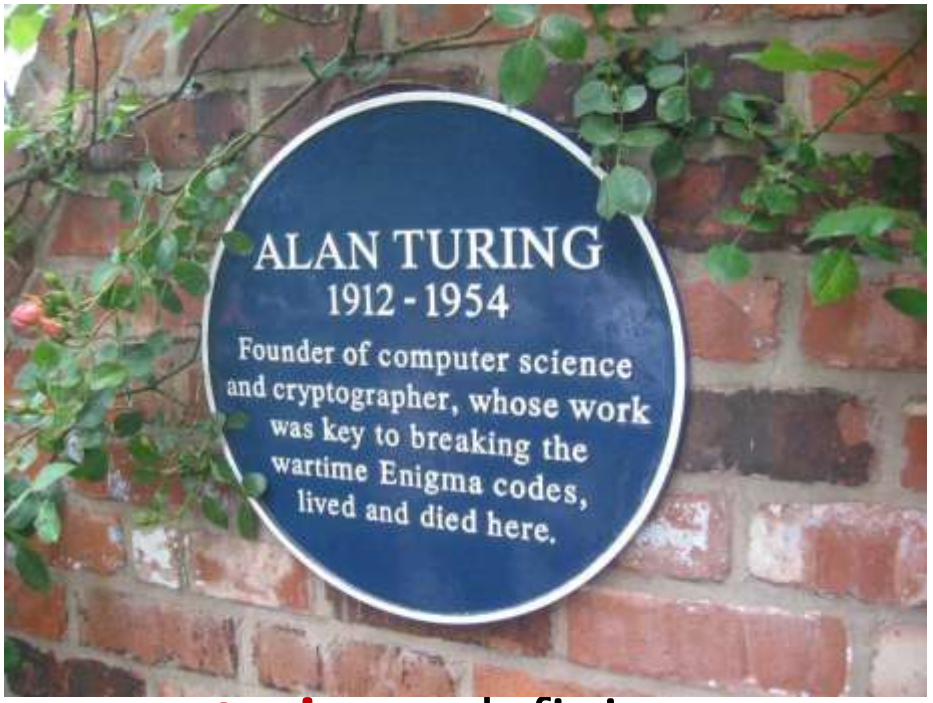
- Può essere fatta risalire a ricercatori quali **Kurt Gödel**, **Alonzo Church** e **Alan Turing** che, nella prima parte '900, studiarono la **teoria degli algoritmi** per capire cosa potesse essere calcolato da una persona capace di eseguire fedelmente una sequenza di istruzioni, con il solo ausilio di carta e penna
  - Prima del 1920 il termine ***computer*** riferiva una persona capace di effettuare calcoli
  - La motivazione per queste ricerche fu, in parte, il desiderio di sviluppare delle ***macchine*** che potessero automatizzare il lavoro di un computer umano
    - Ossia di ideare dei **sistemi di calcolo universali in teoria capaci di eseguire tutte le computazioni possibili**, pervenendo così alla nozione di ***calcolatore universale***

Alan Turing e la sua macchina (1)

- Una figura chiave nell'ideazione del calcolatore universale fu **Alan Mathison Turing** (1912-1954), inglese, della Università di Cambridge



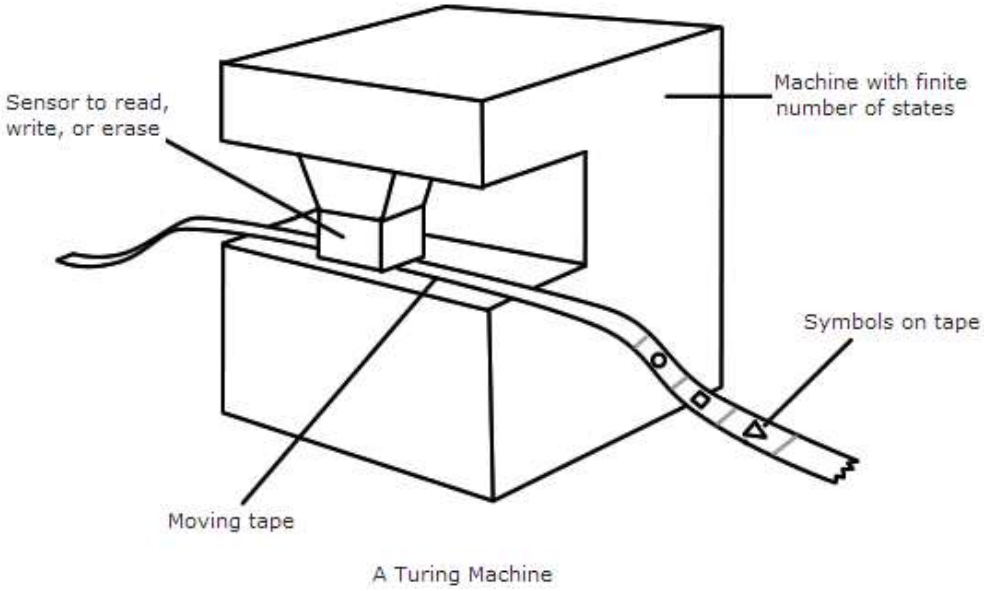
Alan Turing e la sua macchina (1)



- Nel 1937, **Turing** pubblica un importante lavoro sui "computable numbers"
- In esso, per dare una formulazione rigorosa ai concetti intuitivi di **algoritmo** e **computazione**, definisce un **modello computazionale** basato su di una macchina estremamente semplice: la cosiddetta **a-machine** (**automatic**), poi ribattezzata **macchina di Turing**

La macchina di Turing

- Una **macchina di Turing** consta di:
  - Un **alfabeto finito di simboli**
    - Tra cui il simbolo # o □ che denota il *simbolo bianco*

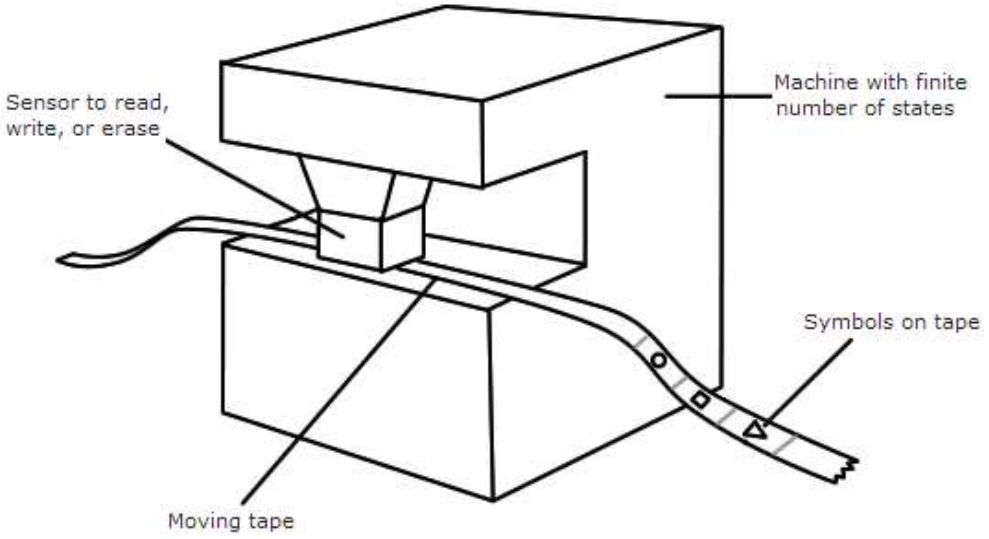


A Turing Machine

  - Un **nastro illimitato** (ossia potenzialmente infinito), suddiviso in quadrati, ciascuno in grado di contenere un simbolo
  - Una **testina di lettura-scrittura** che può muoversi lungo il nastro, un quadrato per volta
  - Un **automa a stati finiti**

La macchina di Turing (2)

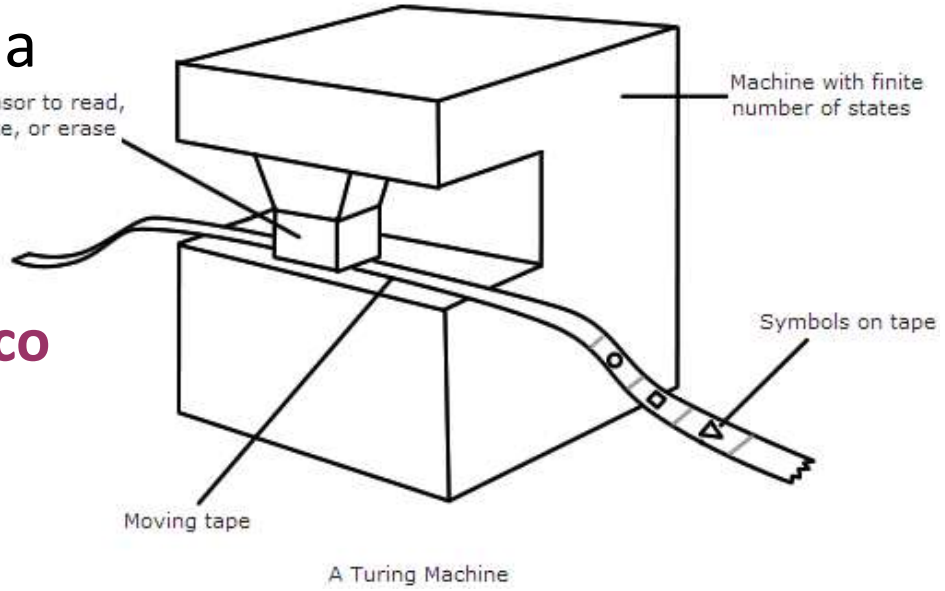
- L' **automa a stati finiti** è formato da
  - Un **insieme finito di stati**
    - Tra cui s'individuano uno stato iniziale ed uno o più stati finali
  - Una **tabella delle transizioni** che porta
    - Sulle righe gli **stati**
    - Sulle colonne i **simboli d'ingresso**
    - In ciascuna casella una terna ***<simbolo, direzione, stato>***, dove ***direzione*** può assumere i valori ***D*** e ***S*** (rispettivamente per destra e sinistra)



A Turing Machine

La macchina di Turing (3)

- La macchina di Turing incomincia la computazione con:
  - L'automa nel suo **stato iniziale**
  - La testina sul **simbolo non bianco più alla sinistra** del nastro



A Turing Machine

- La computazione procede così:
  - La coppia ***<stato corrente, simbolo corrente>*** definisce univocamente una casella della tabella delle transizioni che specifica
    - Il **simbolo da sostituire** nel quadrato corrente del nastro
    - La **direzione dello spostamento** della testina
    - Il **nuovo stato** in cui si porta l'automa

La tesi di Church-Turing (1)

- Il modello computazionale di Turing formalizza la nozione intuitiva di algoritmo
- La semplicità del modello potrebbe però indurre a dubitare della sua utilità e rende quindi legittimo porsi la seguente domanda:

Quali problemi possono essere risolti da una macchina di Turing appropriatamente programmata?

- La risposta a questa domanda, a cui arrivarono indipendentemente Turing e Church negli anni 1936-1937, è sorprendente ed è nota come **tesi di Church-Turing**

Le macchine di Turing sono capaci di risolvere qualsiasi problema algoritmicamente risolubile

La tesi di Church-Turing (2)

- È importante comprendere che la tesi di Church-Turing è in sostanza una **congettura**, non un teorema che possa essere provato in termini matematici
  - La ragione di ciò sta nel fatto che la tesi stabilisce, in sostanza, che la **nozione intuitiva di algoritmo**, ossia di procedura meccanica di risoluzione per passi di un problema, può essere formalizzata in vari modi, ma che non esiste alcun formalismo più potente della macchina di Turing
- Anche se la tesi di Church-Turing non può essere provata in senso matematico, esistono **numerose evidenze** della sua validità

Altri fondamentali risultati di Turing (1)

- **Indecidibilità del problema della terminazione:**

  Non esiste alcun algoritmo halt(A,D) che accettati un qualsiasi algoritmo A e i dati D su cui esso debba operare è in grado di decidere se la computazione A(D) terminerà o meno

- **Esistenza di problemi non computabili o indecidibili:**

  Esistono problemi definiti con precisione per i quali non esiste alcuna macchina di Turing in grado di risolverli

  - **E quindi, per la tesi di Church-Turing, non esiste alcun algoritmo risolutivo**

Altri risultati fondamentali di Turing (2)

- **Esistenza della macchina di Turing universale:**

  Esiste una macchina di Turing UTM(M, D) che accettati la descrizione di una macchina di Turing M e i dati D su cui essa debba operare ne simula il comportamento

  - **Ossia tale che UTM(M, D) = M(D)**

… e le loro implicazioni (1)

- Per quanto complessi e potenti possano diventare i calcolatori e i loro linguaggi, non è possibile, né lo sarà mai, progettare un calcolatore in grado di risolvere un problema che le macchine di Turing non siano capaci di risolvere
- I linguaggi di programmazione, purché sufficientemente generali, sono tutti **Turing-completi**, ossia equivalenti alle macchine di Turing e quindi equivalenti tra di loro
  - **Sufficientemente generali** sono tutti i linguaggi dotati di **forme condizionali e iterative su condizione**

… e le loro implicazioni (2)

- Per le considerazioni sviluppate, se **L** è un linguaggio Turing-completo **è possibile scrivere un programma in L che simula l'esecuzione di un qualsiasi altro programma scritto in L, ossia è possibile scrivere in L l'interprete del linguaggio L**
- Un **interprete** del linguaggio **L** è dunque un programma **IL** che ricevuto un programma **P** scritto in **L** e i suoi dati iniziali **D**, simula la computazione sviluppata da **P** sui dati **D**:

$$IL(P, D) = P(D)$$

- L'interprete di **L** può essere scritto, oltre che in **L** stesso, in un qualsiasi altro linguaggio Turing-completo

La realizzazione del calcolatore universale

- Per la **concreta realizzazione** del calcolatore universale fu, però, necessario attendere gli anni ’40, quando anche grazie alla spinta prodotta dalle esigenze belliche della **seconda guerra mondiale**, furono progettati e realizzati i primi computer
  - Da questo momento in poi, il termine computer ha cominciato a riferire una macchina e non più una persona
- Quando è diventato chiaro che i computer possono essere usati non solo per eseguire calcoli matematici, ma più in generale per ***elaborare informazioni***, ha avuto inizio la ***“rivoluzione informatica”***

Una veloce “cavalcata” tra i primi linguaggi di programmazione

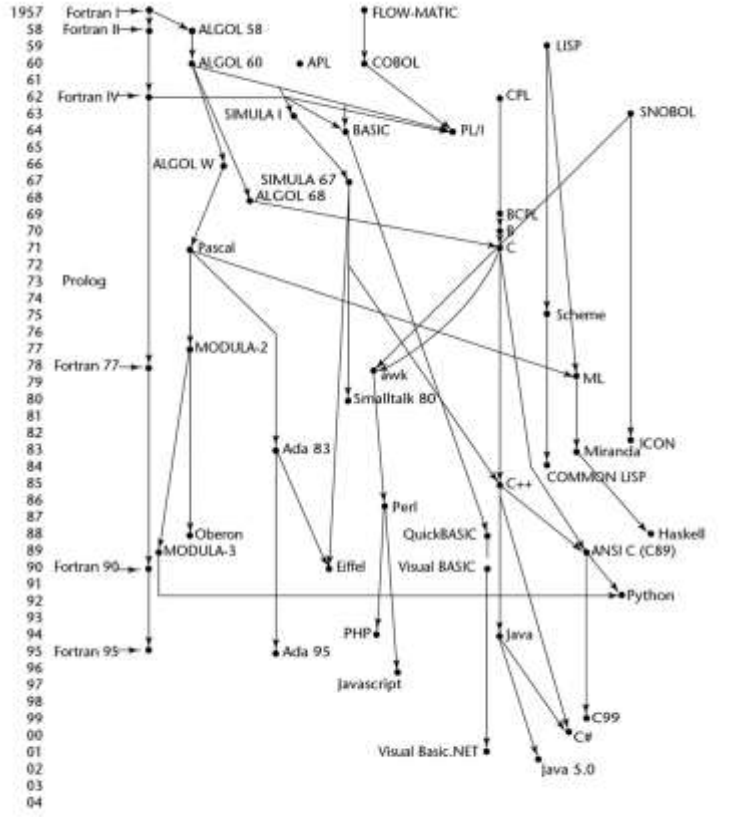


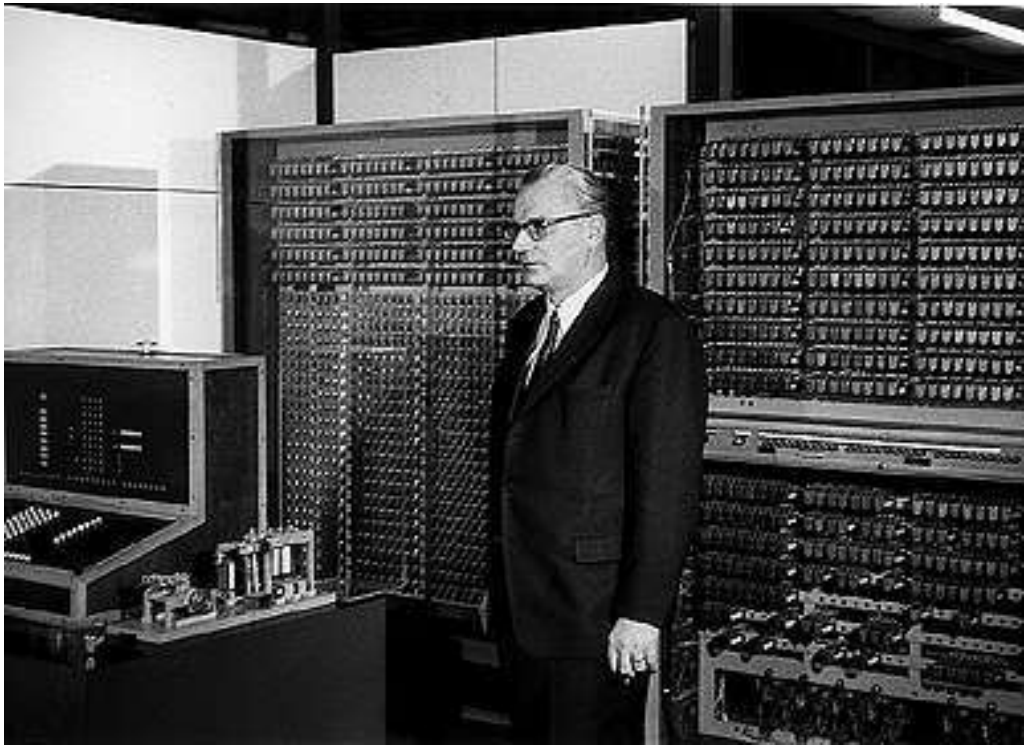
Figure 2.1 Genealogy of common high-level programming languages

- ✓ **Il Plankalcül di Zuse**
- ✓ **Dal linguaggio macchina alle prime notazioni simboliche**
- ✓ **Il FORTRAN**
- ✓ **IL LISP**
- ✓ **L’Algol**
- ✓ **Il COBOL**
- ✓ **…**

Konrad Zuse (1)

- Per comprendere l'importanza di **Konrad Zuse** (1910-1995), tedesco basta riflettere sul fatto che, lavorando da solo, nella Germania nazista in guerra riuscì a realizzare fin dal 1938 dei calcolatori meccanici ed elettromeccanici (Z1, Z2, Z3, Z4)
- In particolare lo Z3, completato nel dicembre 1941 ha le seguenti caratteristiche
  - Il programma è letto da **nastro perforato** che proviene da spezzoni di vecchia pellicola cinematografica perforati a mano
  - Usa **numeri FP** con bit segno, esponente a 7 bit e mantissa a 14-bit, con hidden bit
  - Ha una memoria a **relè** con capacità di 64 parole
  - Non dispone di salti condizionali ma è stato dimostrato nel 1998 che la macchina è Turing completa
    - A prescindere dalla memoria limitata

Konrad Zuse (2)



Lo Z3 originale fu distrutto durante un bombardamento alleato di Berlino

Una copia perfettamente funzionante fu realizzata negli anni ’60 dalla Zuse KG ed è in esposizione permanente al Deutsches Museum di Monaco

Il Plankalcül di Zuse (1)

- Negli anni dal 1943 al 1945, Zuse definì **Plancalcül**, un linguaggio di programmazione ad alto livello dotato di molti meccanismi avanzati quali istruzioni di assegnazione, sottoprogrammi, istruzioni condizionali, iterazioni, aritmetica floating-point, array, record con struttura gerarchica, asserzioni, gestione delle eccezioni ed altre caratteristiche avanzate

Il Plankalcül di Zuse (2)

- Zuse usò il linguaggio per scrivere numerosi algoritmi e un programma per il gioco degli scacchi
- A causa dell'isolamento provocato dalla guerra, il suo lavoro rimase largamente ignorato e fu pubblicato solo nel 1972
- Un'istruzione di assegnazione **`A[5] = A[4]+1`**

```
    |    A + 1 => A
V   |    4          5       (indici)
S   |    1.n        1.n     (tipi di dato)
```

Dal linguaggio macchina alle prime notazioni simboliche

THE EARLY DEVELOPMENT OF PROGRAMMING LANGUAGES

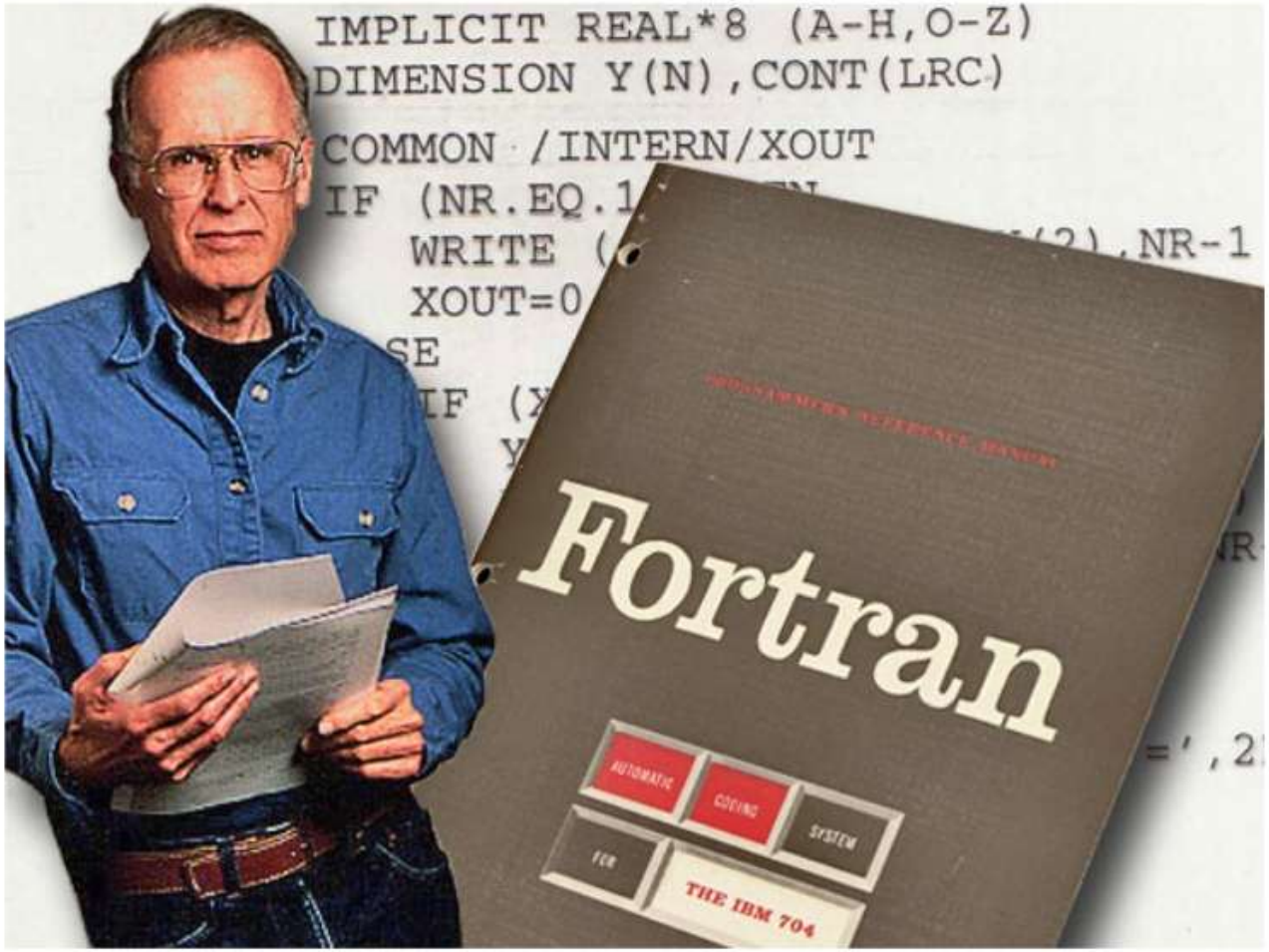
by

Donald E. Knuth
Luis Trabb Pardo

STAN-CS-76-562
AUGUST 1976

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Il FORTRAN (1)



John Backus (1924 – 2007)

Il FORTRAN (2)

- **Fortran 0**: 1954 – non implementato
- **Fortran I**: 1957
  - Progettato per il nuovo computer **IBM 704** dotato di registri indice e hardware floating-point
  - Ambiente di sviluppo
    - Calcolatori “piccoli” e inaffidabili
    - Applicazioni di tipo scientifico
    - Nessuna metodologia o strumento di programmazione
    - Efficienza della macchina assolutamente prioritaria

Il FORTRAN (3)

- Nessuna necessità di memoria dinamica
- Necessità di buona gestione degli array e di cicli a conteggio
- Nessuna necessità di gestione stringhe, aritmetica decimale e input/output sofisticato (utili in un contesto di applicazioni commerciali)
- Singola struttura dati: array
- Commenti
- Espressioni aritmetiche
- Cicli DO
- Sottoprogrammi e funzioni
- I/O semplificato
- Indipendenza dalla macchina

L'IBM 704

Il FORTRAN (4)

- Nella prima versione implementata del Fortran
  - Nomi lunghi al più 6 caratteri
  - Ciclo a conteggio (DO)
  - I/O formattato
  - Sottoprogrammi definiti dal programmatore
  - Istruzione di scelta a tre vie (IF aritmetico)
  - Senza dichiarazioni di tipo
  - Senza compilazione separata
- Il compilatore fu rilasciato nell'aprile 1957, dopo 18 anni-uomo di lavoro
  - I programmi con più di 400 linee difficilmente compilavano correttamente a causa della scarsa affidabilità del 704
  - Il codice era molto veloce e il linguaggio divenne largamente

Il FORTRAN (5)

- **Fortran II**
  - Distribuito nel 1958
  - Dotato di compilazione indipendente
- **Fortran IV**
  - Sviluppato negli anni 1960-62
  - Dotato di dichiarazioni esplicite di tipo
  - Istruzione di scelta logica
  - Nomi di sottoprogrammi utilizzabili come parametri
  - Standard ANSI nel 1966

Un esempio di programma FORTRAN


```
C       FIND THE MEAN OF N NUMBERS AND THE NUMBER OF
C       VALUES GREATER THAN IT
        DIMENSION A(99)
        REAL MEAN
        READ(1,5)N
 5      FORMAT(I2)
        READ(1,10)(A(I),I=1,N)
 10     FORMAT(6F10.5)
        SUM=0.0
        DO 15 I=1,N
 15        SUM=SUM+A(I)
        MEAN=SUM/FLOAT(N)
        NUMBER=0
        DO 20 I=1,N
           IF (A(I) .LE. MEAN) GOTO 20
           NUMBER=NUMBER+1
 20     CONTINUE
        WRITE (2,25) MEAN,NUMBER
 25     FORMAT(11H MEAN = ,F10.5,5X,21H NUMBER SUP = ,I5)
        STOP
        END
```

Il foglio di codifica FORTRAN

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)	IDENTIFICATION (73–80)
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C	X	ATTAINED BY A SET OF NUMBERS	
		DIMENSION A(999)	
		FREQUENCY 30(2,1,10), 5(100)	
		READ 1, N, (A(I), I = 1,N)	
1		FORMAT (I3/(12F6.2))	
		BIGA = A(1)	
5		DO 20 I = 2,N	
30		IF (BIGA-A(I)) 10,20,20	
10		BIGA = A(I)	
20		CONTINUE	
		PRINT 2, N, BIGA	
2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
		STOP 77777	

La scheda perforata

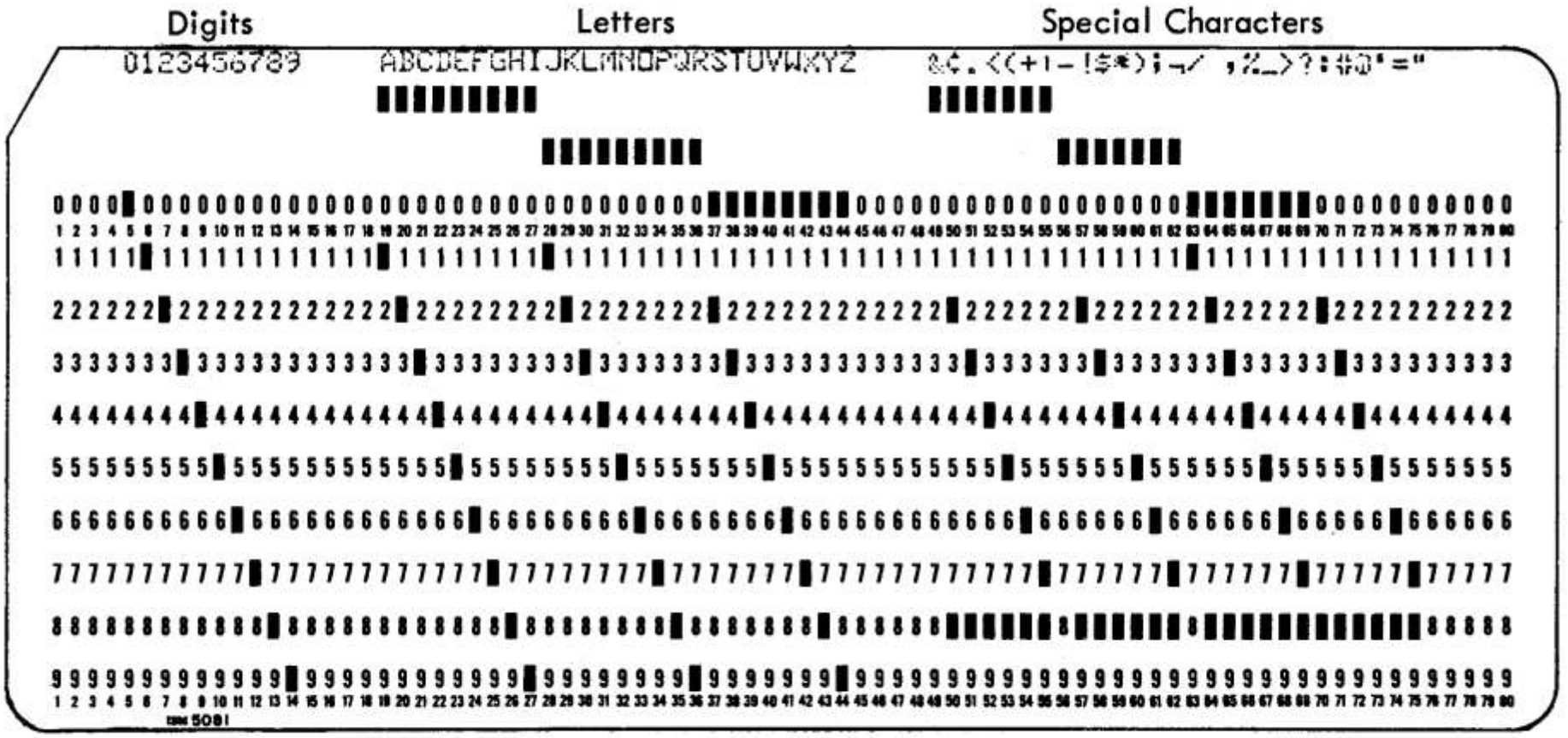
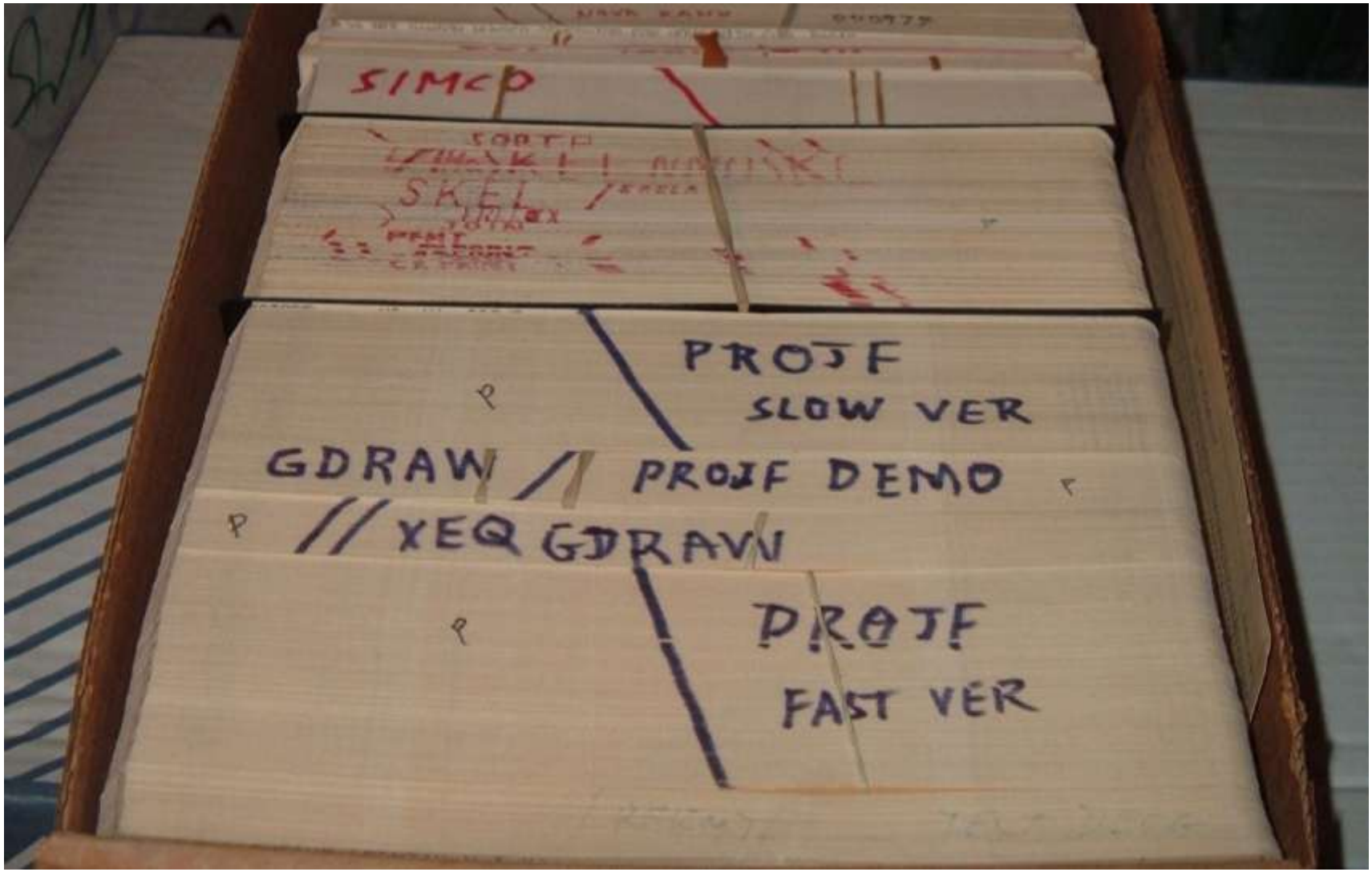


Figure 4. Card Codes and Graphics for 64-Character Set

Un contenitore di schede perforate



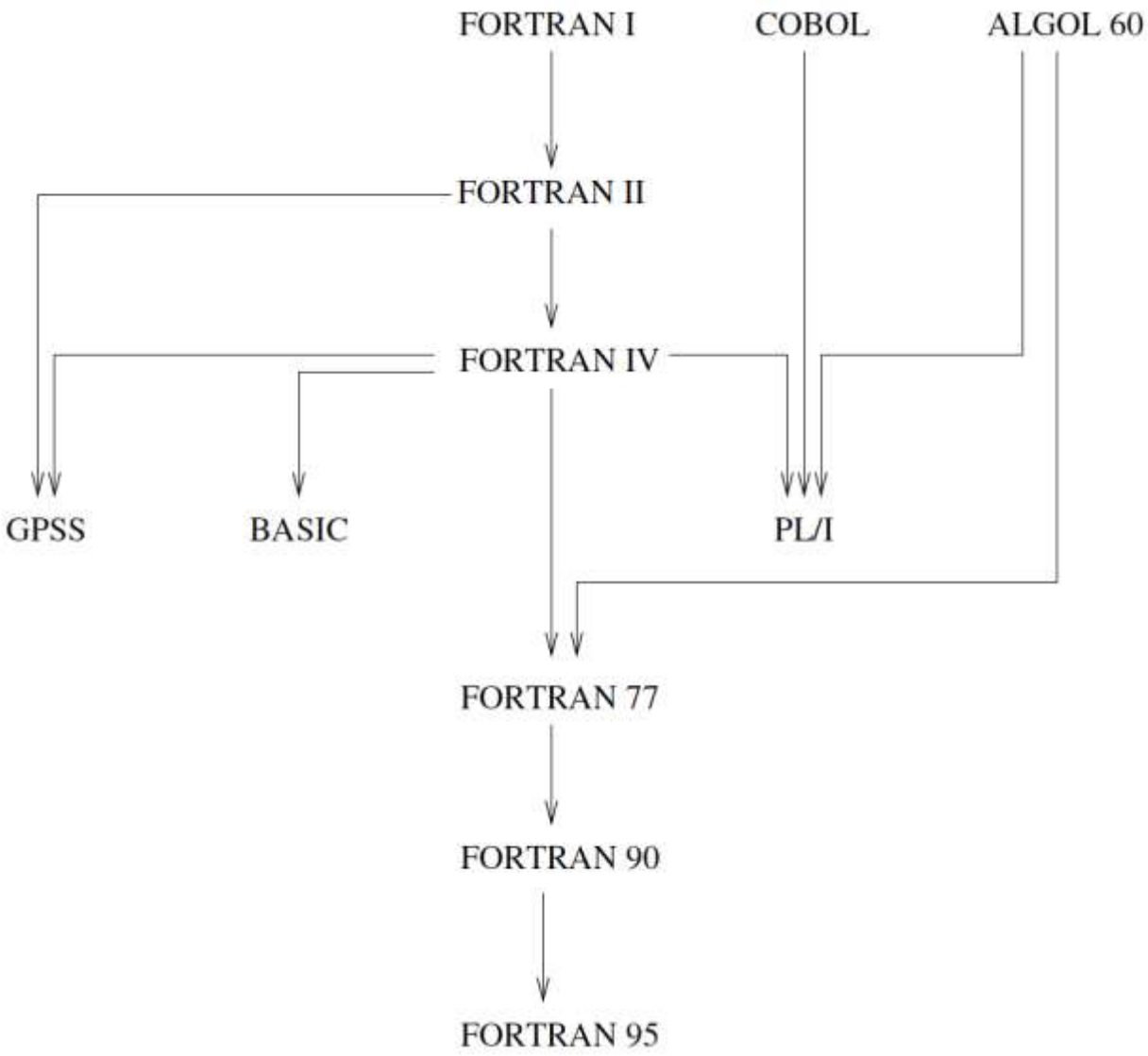
I FORTRAN 77 e 90

- **Fortan 77**
  - Nuovo standard nel 1978
  - Gestione delle stringhe di caratteri
  - Cicli logici
  - Istruzione IF-THEN-ELSE
- **Fortran 90**
  - Modifiche significative rispetto al Fortran 77
    - Moduli
    - Array dinamici
    - Puntatori
    - Ricorsività
    - Istruzione CASE
    - Type checking dei parametri

Una valutazione del FORTRAN

- Compilatori altamente ottimizzanti
  - Tipi e memoria di tutte le variabili sono fissati staticamente
- Ha radicalmente modificato per sempre il modo con cui I computer erano utilizzati
- Si caratterizzò come la “lingua franca” del mondo informatico

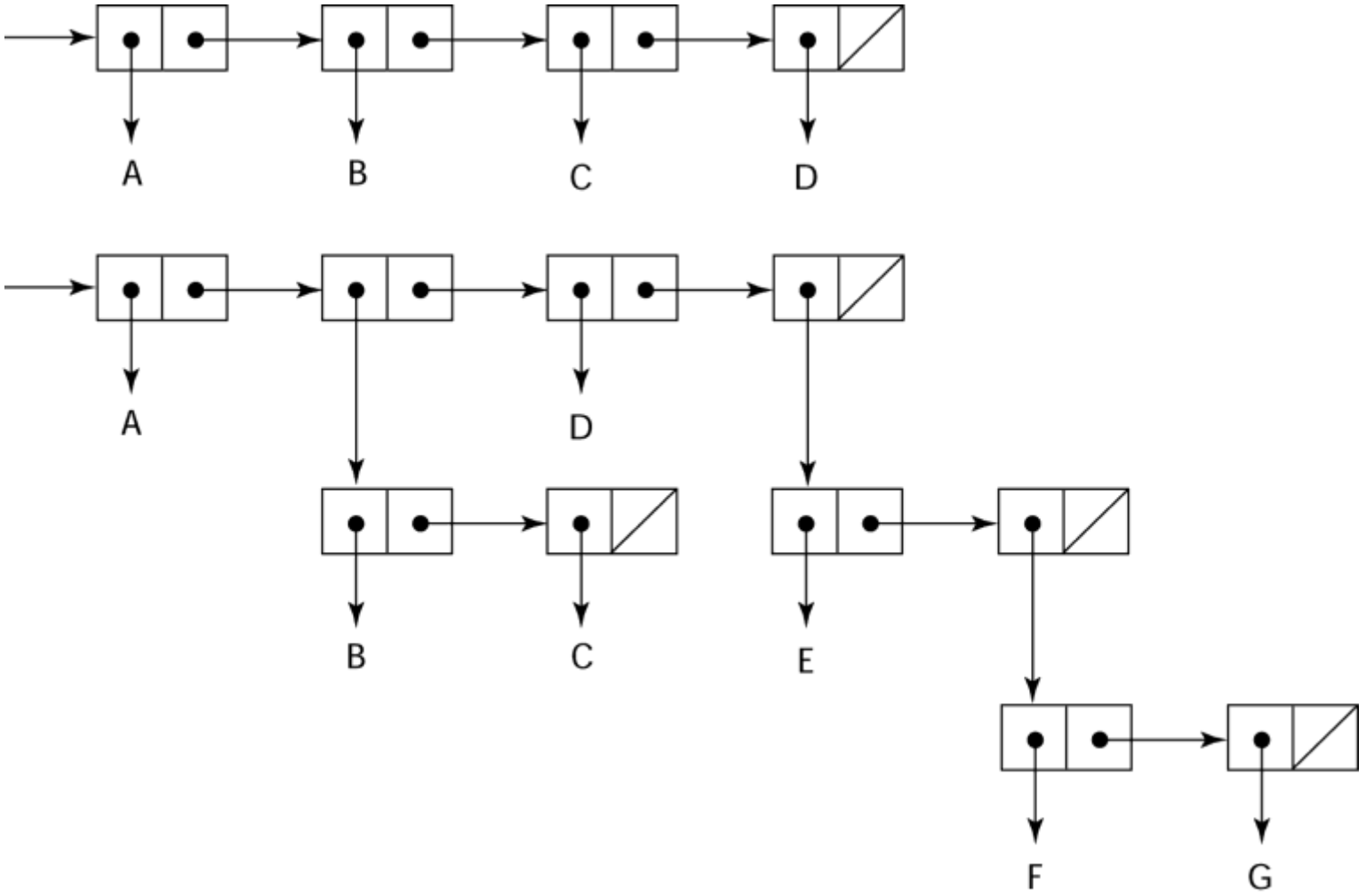
I FORTRAN e …


FORTRAN I
COBOL
ALGOL 60
FORTRAN II
FORTRAN IV
GPSS
BASIC
PL/I
FORTRAN 77
FORTRAN 90
FORTRAN 95

Il LISP e la programmazione funzionale (1)

- **LIS**t **P**rocessing
  - Progettato al MIT da **John Mc Carthy** nel 1958
- La ricerca in AI aveva bisogno di un linguaggio
  - Per processare dati organizzati in liste, piuttosto che array
  - Per sviluppare computazioni simboliche piuttosto che numeriche
- Usa solo due tipi di dati: atomi e liste
- Ha sintassi basata sul lambda calcolo di Alonzo Church

Il LISP e la programmazione funzionale (2)



Una valutazione del LISP

- È il primo linguaggio che ha introdotto la programmazione funzionale
  - Nessuna necessità di variabili e istruzioni di assegnazione
  - Controllo realizzato con la ricorsività e le espressioni condizionali
- È ancora oggi il linguaggio dominante per le applicazioni di AI
  - COMMON LISP e Scheme sono dialetti contemporanei del LISP
  - ML, Miranda, and Haskell are linguaggi funzionali più moderni

Scheme e COMMON LISP

- **Scheme** fu sviluppato al MIT a metà degli anni ’70
  - Fa uso esclusivo dello scoping statico
  - Le funzioni sono entità di prima classe
  - Ha sintassi semplice ed è complessivamente “di piccole dimensioni” ed è largamente impiegato nella didattica
- **COMMON LISP** fu creato nel tentativo di combinare le caratteristiche di diversi dialetti del LISP
  - Usa sia lo scope statico che quello dinamico

L'Algol (1)

- L'esigenza di avere un linguaggio portabile e universale per comunicare algoritmi portò alla progettazione dei linguaggi della famiglia Algol
- La prima fase del processo di progetto fu realizzata da ACM (America) e GAMM (Germania) che si incontrarono a Zurigo per 4 giorni dal 27 maggio al 1 giugno 1958 e stilarono i seguenti obiettivi fondamentali
  - Sintassi il più possibile vicina alla notazione matematica
  - Adatto per la descrizione di algoritmi
  - Traducibile in linguaggio macchina

L'Algol (2)

- Nacque così l'**Algol 58**
  - Nomi di lunghezza qualsiasi
  - Concetto di tipo formalizzato
  - Array con qualsiasi numero di indici, posti tra parentesi
  - Parametri separati per modo (in & out)
  - Istruzioni composte (begin … end)
  - Punto e virgola usato come separatore
  - Assegnazione denotata con :=
  - Istruzione if con clausola else-if
  - Nessuna istruzione di I/O

L'Algol (3)

- L'Algol 58 non fu implementato, anche se lo furono alcune sue variazioni (MAD, JOVIAL)
- Durante il 1959 il linguaggio fu furiosamente dibattuto sia in Europa che in America e numerose furono le proposte di modifica
- Nel gennaio 1960 si tenne un secondo meeting a Parigi, dove furono dibattuti gli 80 suggerimenti che erano stati formalmente sottomessi al comitato
- In questa fase assunse grande rilievo il contributo di **Peter Naur** (1928) che perfezionò il formalismo **BNF** di Backus per esprimere la sintassi del linguaggio e la utilizzò per formulare la sintassi del nuovo linguaggio

L'Algol (4)

for loop syntax

```
<for statement>
            ::= <for clause> <statement>
              | <label>: <for statement>

<for clause> ::= for <variable> := <for list> do

<for list> ::= <for list element>
             | <for list> , <for list element>

<for list element>
 ::= <arithmetic expression>
   | <arithmetic expression> step <arithmetic expression>
                            until <arithmetic expression>
   | <arithmetic expression> while <Boolean expression>
```

L'Algol (5)

Figure: John Mac Carthy, Fritz Bauer, Joe Wegstein. Bottom row: John Backus, Peter Naur, Alan Perlis [1]

L'Algol (6)

- Le principali caratteristiche dell'**Algol 60**
  - Sintassi descritta in BNF
  - Struttura a blocchi
  - Metodi di passaggio dei parametri per nome e per valore
  - Sottoprogrammi ricorsivi
  - Array dinamici sullo stack
  - Variabili own

ALGOL was a great improvement on its successors

— C.A.R. Hoare

L'Algol (7)

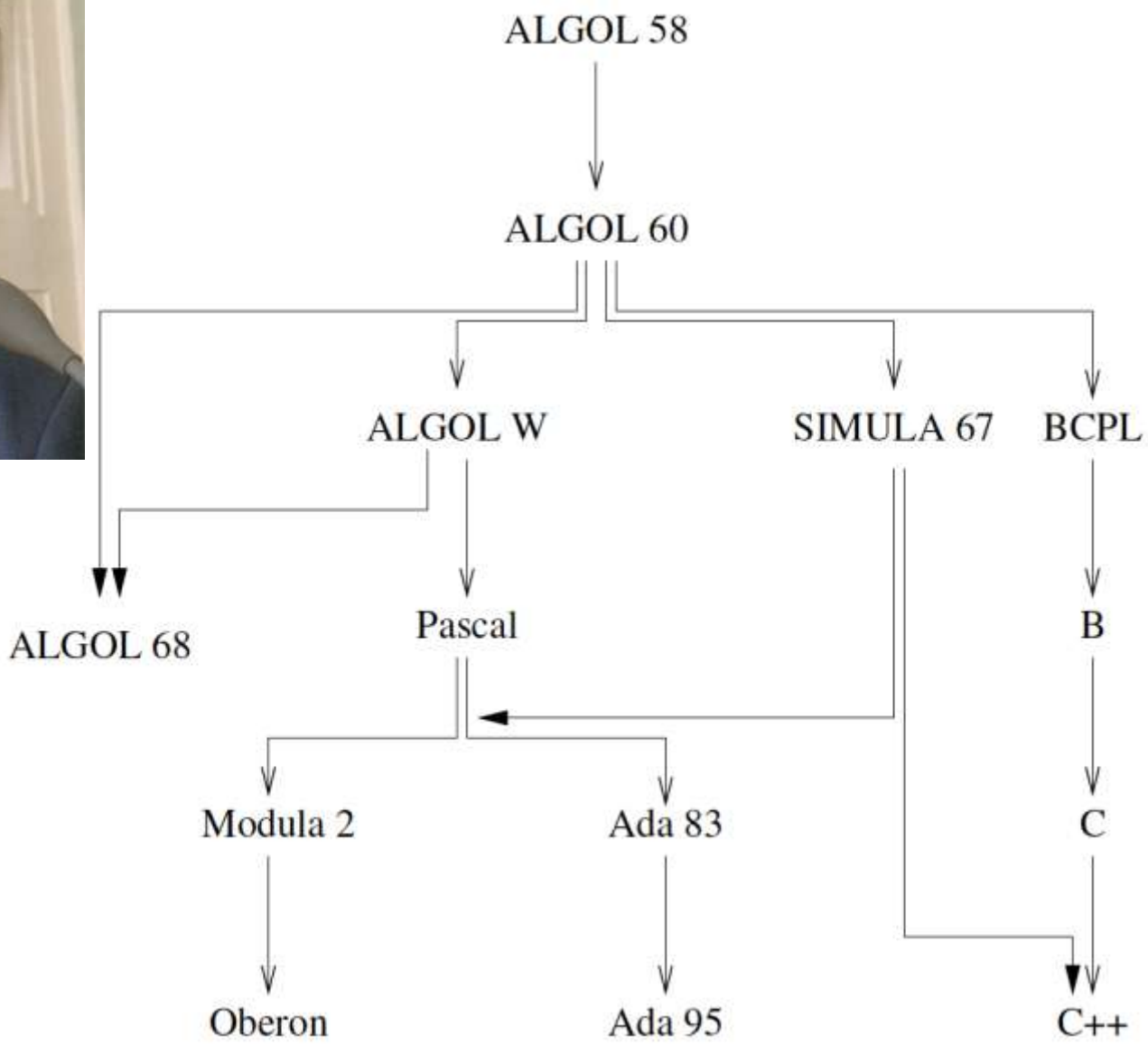

```
begin
  comment The mean of numbers and the number of greater values;
  integer n;
  read(n);
  begin
    real array a[1:n];
    integer i, number;
    real sum, mean;
    for i := 1 step 1 until n do
      read (a[i]);
    sum := 0;
    for i := 1 step 1 until n do
      sum := sum + a[i];
    mean := sum / n;
    number := 0;
    for i := 1 step 1 until n do
      if a[i] > mean then
        number := number + 1;
    write ("Mean = ", mean, "Number sups = ", number);
  end
end
```

L'Algol (8)

- **Successi**
  - È stato per oltre 20 anni il linguaggio per la pubblicazione di algoritmi
  - Tutti i successivi linguaggi imperativi sono dei suoi derivati
  - È stato il primo linguaggio la cui sintassi è stata formalmente definita
  - È stato il primo linguaggio indipendente dalla macchina
- **Fallimenti**
  - Mai largamente utilizzato, soprattutto negli Stati Uniti
    - Mancanza di supporto da parte dell'IBM
    - Più complicato e meno efficiente del FORTRAN
    - Mancanza d'istruzioni di I/O

L'Algol e …

Niklaus Wirth (1934)



Il COBOL (1)

- Al contrario dell'Algol, il **COBOL** (Common Business Oriented Language) è stato ed è ancora un linguaggio molto utilizzato, ma non ha avuto quasi alcuna influenza nello sviluppo dei linguaggi successivi
- Anche il COBOL è stato progettato da un comitato che si è riunito la prima volta con l'obiettivo di realizzare un linguaggio comune per applicazioni commerciali al Pentagono nei giorni 28 e 29 maggio 1959
  - Al momento la situazione era alquanto disgregata
    - UNIVAC aveva cominciato a usare FLOW-MATIC
    - U.S. Air Force aveva cominciato a usare AIMACO
    - IBM stava sviluppando COMTRAN

Il COBOL (2)

- I membri del comitato di progetto provenivano tutti dai costruttori di computer e dal DoD (Department of Defence)
- Gli obiettivi di progetto furono quelli di ottenere un linguaggio
  - Somigliante a un semplice linguaggio naturale (Inglese)
  - Facile da usare, anche rinunciando a qualche caratteristica utile
  - In grado di ampliare la base degli utenti
  - Non polarizzato dai problemi correnti di sviluppo dei compilatori
- Problemi di progetto: Espressioni aritmetiche? Indici?

Il COBOL (3)

- **Contributi**
  - Primo linguaggio ad alto livello a rendere disponibile una macro facility
  - Strutture di dati gerarchiche (record)
  - Istruzioni di scelta innestate
  - Nomi lunghi (fino a 30 caratteri), con hyphen (_)
  - Data division separata
- Il supporto del DoD fu probabilmente decisivo per il successo del linguaggio

Il COBOL (4)


```
IDENTIFICATION DIVISION.
PROGRAM-ID. INOUT.
* Read a file, add information to records, and save
* as another file.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

       SELECT INP-FIL ASSIGN TO INFILE.
       SELECT OUT-FIL ASSIGN TO OUTFILE.
```

Il COBOL (5)


```
DATA DIVISION.
FILE SECTION.

FD        INP-FIL
          LABEL RECORDS STANDARD
          DATA RECORD IS REC-IN.
01        REC-IN.
           05 ALPHA-IN PIC A(4).
           05 SP-CH-IN PIC X(4).
           05 NUM-IN   PIC 9(4).
FD        OUT-FIL
          LABEL RECORDS STANDARD
          DATA RECORD IS REC-OUT.
01        REC-OUT.
           05 ALPHA-OUT PIC A(4).
           05 SP-CH-OUT PIC X(4).
           05 NUM-OUT   PIC 9(4).
           05 EXTRAS    PIC X(16).
```

Il COBOL (6)


```
WORKING-STORAGE SECTION.
01        EOF PIC X VALUE IS 'N'.

PROCEDURE DIVISION.
AA.
          OPEN INPUT INP-FIL
          OPEN OUTPUT OUT-FIL

          PERFORM CC
          PERFORM BB THRU CC UNTIL EOF = 'Y'

          CLOSE INP-FIL, OUT-FIL
          DISPLAY "End of Run"

          STOP RUN
BB.
          MOVE REC-IN TO REC-OUT
          MOVE 'EXTRA CHARACTERS' TO EXTRAS
          WRITE REC-OUT.

CC.
          READ INP-FIL
                    AT END MOVE 'Y' TO EOF.
```

Un personaggio chiave: Grace Murray-Hopper (1)

Un personaggio chiave: Grace Murray-Hopper (2)

Un personaggio chiave: Grace Murray-Hopper (3)

Un personaggio chiave: Grace Murray-Hopper (4)

Il primo bug

9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 ?95 correct
13"UC (032) MP - MC ~~2.130476915 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
in relay . 11.00 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.

1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Mark II Computer at Harvard University in 1947

II BASIC (1)

- Designed by Kemeny & Kurtz at Dartmouth
- Design Goals:
  - Easy to learn and use for non-science students
  - Must be “pleasant and friendly”
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- Current popular dialect: Visual BASIC
- First widely used language with time sharing

Il BASIC (2)


```
 10 REM FIND THE MEAN OF N
 12 REM NUMBERS AND THE
 14 REM NUMBER OF VALUES
 16 REM GREATER THAN IT
 20 DIM A(99)
 30 INPUT N
 40 FOR I = 1 TO N
 50 INPUT A(I)
 60 LET S = S + A(I)
 70 NEXT I
 80 LET M = S / N
 90 LET K = 0
100 FOR I = 1 TO N
110 IF A(I) < M THEN 130
120 LET K = K + 1
130 NEXT I
140 PRINT "MEAN = ", M
150 PRINT "NUMBER SUP = ", K
160 STOP
170 END
```

II PL/I (1)

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)
  - Scientific computing
    - IBM 1620 and 7090 computers
    - FORTRAN
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group

II PL/I (2)

- By 1963
  - Scientific users began to need more elaborate I/O, like COBOL had; business users began to need floating point and arrays
  - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly
- The obvious solution
  - Build a new computer to do both kinds of applications
  - Design a new language to do both kinds of applications

II PL/I (3)

- Designed in five months by the 3 X 3 Committee
  - Three members from IBM, three members from SHARE
- Initial concept
  - An extension of Fortran IV
- Initially called NPL (New Programming Language)
- Name changed to PL/I in 1965

II PL/I (4)

- PL/I contributions
  - First unit-level concurrency
  - First exception handling
  - Switch-selectable recursion
  - First pointer data type
  - First array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex

> If Fortran has been called an infantile disorder, PL/I must be classified as a fatal disease.
>
> — Edsger Dijkstra

L'APL (1)

- Designed as a hardware description langu

  Ken Iverson around 1960

  - Highly expressive (many operators, for bot

    scalars and arrays of various dimensions)

  - Programs are very difficult to read

- APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.

  — Edsger Dijkstra, 1968

L'APL (2)

Prime Numbers up to R

$$(\sim R \in R \circ . \times R)/R \leftarrow 1 \downarrow \iota R$$

II SIMULA 67

- Designed primarily for system simulation in Norway by Nygaard and Dahl
- Based on ALGOL 60 and SIMULA I
- Primary Contributions
  - Co-routines - a kind of subprogram
  - Implemented in a structure called a class
  - Classes are the basis for data abstraction

L'Algol 68 (1)

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
- Design is based on the concept of orthogonality
  - A few principle concepts, few combining mechanisms

L'Algol 68 (2)

- Contributions
  - User-defined data structures
  - Reference types
  - Dynamic arrays (called flex arrays)
- Comments
  - Less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

I primi discendenti dei linguaggi Algol

- ALGOL languages impacted all imperative languages
  - Pascal
  - C
  - Modula/Modula 2
  - Ada
  - Oberon
  - C++/Java
  - Perl (to some extent)

Il Pascal

- Developed by Wirth (a member of the ALGOL 68 committee)

  1971

- Designed for teaching structured programming

- Small, simple, nothing really new

- Largest impact on teaching programming

  - From mid-1970s until the late 1990s, it was the most widely used

    language for teaching programming

Il C

- Designed for systems programming (at Bell Labs by Dennis Richie) 1972
- Evolved primarily from BCLP, B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX
- Many areas of application

ll PERL

- Related to ALGOL only through C
- A scripting language
  - A script (file) contains instructions to be executed
  - Other examples: sh, awk, tcl/tk
- Developed by Larry Wall
- Perl variables are statically typed and implicitly declared
  - Three distinctive namespaces, denoted by the first character of a variable's name
- Powerful but somewhat dangerous
- Widely used as a general purpose language

Il Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
- Based on formal logic
- Non-procedural
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries
- Highly inefficient, small application areas

Il linguaggio ADA (1)

- Huge design effort, involving hundreds of people, much money, and about eight years
  - Strawman requirements (April 1975)
  - Woodman requirements (August 1975)
  - Tinman requirements (1976)
  - Ironman equipments (1977)
  - Steelman requirements (1978)
- Named Ada after Augusta Ada Byron, known as being the first

Il linguaggio ADA (2)

- Contributions
  - Packages - support for data abstraction
  - Exception handling - elaborate
  - Generic program units
  - Concurrency - through the tasking model
- Comments
  - Competitive design
  - Included all that was then known about software engineering and language design
  - First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

ADA 95

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data
  - New concurrency features
  - More flexible libraries
- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic type binding)
- Pioneered the graphical user interface design
- Promoted OOP

C++

- Developed at Bell Labs by Stroustrup in 1980
  - Evolved from C and SIMULA 67
- Facilities for OO programming, taken partially from SIMULA 67
- Provides exception handling
- A large and complex language, in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November 1997
- Microsoft's version (released with .NET in 2002): Managed C++
  - Delegates, interfaces, no multiple inheritance

Altri linguaggi ad oggetti

- Eiffel (designed by Bertrand Meyer - 1992)
  - Not directly derived from any other language
  - Smaller and simpler than C++, but still has most of the power
  - Lacked popularity of C++ because many C++ enthusiasts were already C programmers
- Delphi (Borland)
  - Pascal plus features to support OOP
  - More elegant and safer than C++

Java (1)

- Developed at Sun in the early 1990s
  - C and C++ were not satisfactory for embedded electronic devices
- Based on C++
  - Significantly simplified (does not include struct, union, enum, pointer arithmetic, and half of the assignment coercions of C++)
  - Supports only OOP
  - Has references, but not pointers

Java (2)

- Eliminated unsafe features of C++
- Concurrency features
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for WWW pages
- Use for other areas increased faster than any other language
- Most recent version, 5.0, released in 2004

Linguaggi di scripting per il web

- JavaScript
  - A joint venture of Netscape and Sun Microsystems
  - Used in Web programming (client side) to create dynamic HTML documents
  - Related to Java only through similar syntax
- PHP
  - PHP: Hypertext Preprocessor
  - Used for Web applications (server side); produces HTML code as output
- Python
  - An OO interpreted scripting language
  - Type checked but dynamically typed

C#

- Part of the .NET development platform
- Based on C++ , Java, and Delphi
- Provides a language for component-based software development
- All .NET languages (C#, Visual BASIC.NET, Managed C++, J#.NET, and Jscript.NET) use Common Type System (CTS), which provides a common class library
- Likely to become widely used

Linguaggi di markup/programmazione ibridi

- XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display
  - Programming constructs (e.g., looping)
- JSP
  - Java Server Pages: a collection of technologies to support dynamic Web documents
  - servlet: a Java program that resides on a Web server; servlet's output is displayed by the browser