Strumenti per gli informatici, i linguaggi di programmazione

Storia dell'Informatica
a.a. 2024/25

contenuti

- La questione di fondo
- Dai “primi” linguaggi agli standard longevi
- Formali, logici, a oggetti, dichiarativi
- Dal web ai linguaggi visuali
- L’Open Source e la Babele odierna

I programmi di Ada (1843)

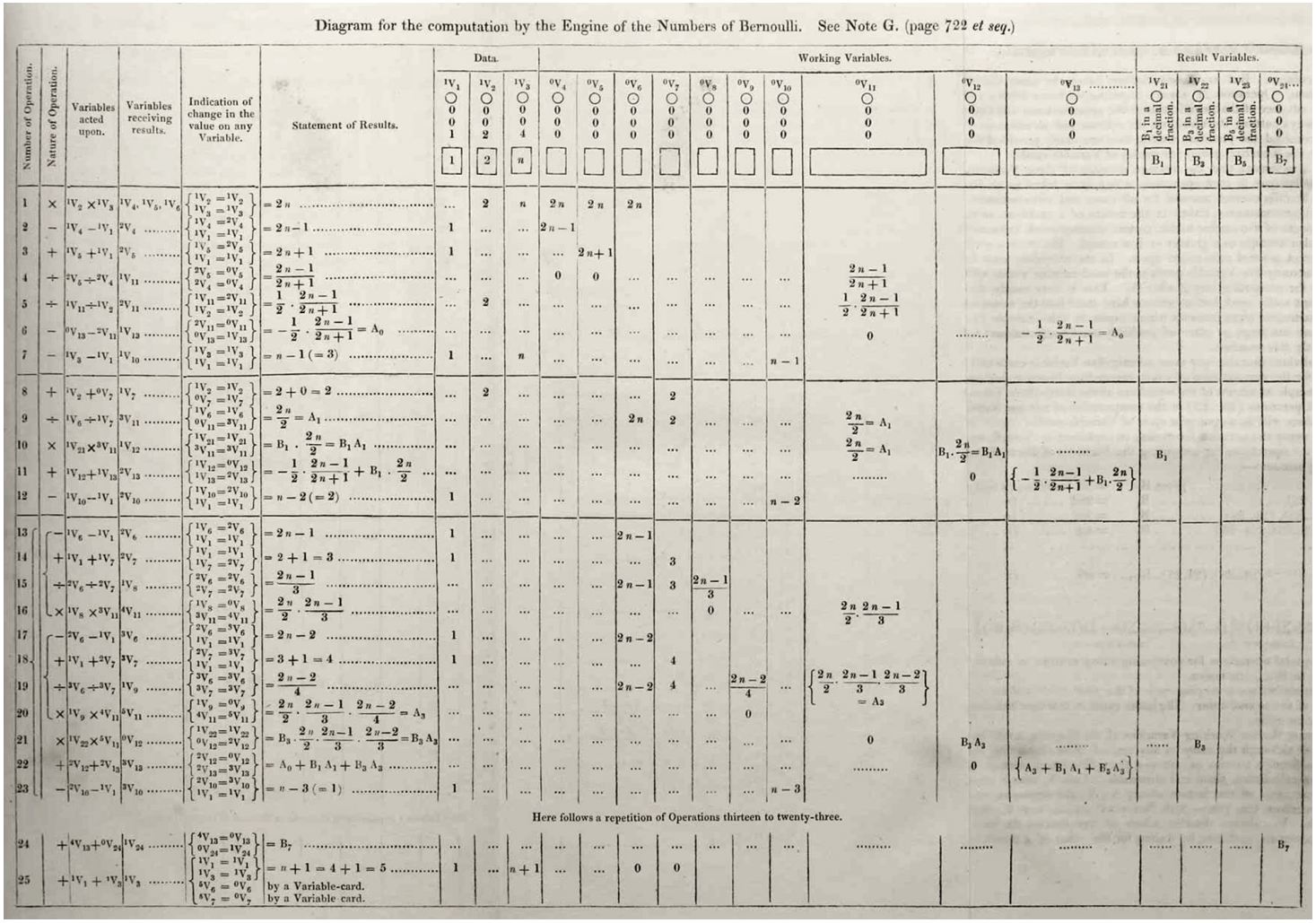


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

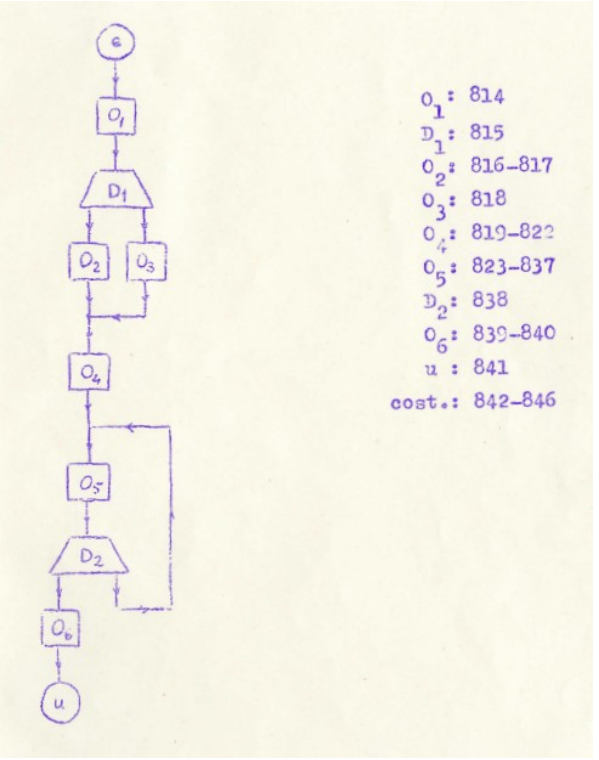
Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. ${}^1V_1$ 1	Data. ${}^1V_2$ 2	Data. ${}^1V_3$ n	Working Variables. ${}^0V_4$	${}^0V_5$	${}^0V_6$	${}^0V_7$	${}^0V_8$	${}^0V_9$	${}^0V_{10}$	${}^0V_{11}$	${}^0V_{12}$	${}^0V_{13}$	Result Variables. ${}^1V_{21}$ B_1	${}^3V_{22}$ B_3	${}^1V_{23}$ B_5	${}^0V_{24}$... B_7
1	×	${}^1V_2 \times {}^1V_3$	${}^1V_4, {}^1V_5, {}^1V_6$	${}^1V_2 = {}^1V_2$; ${}^1V_3 = {}^1V_3$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	${}^1V_4 - {}^1V_1$	${}^2V_4$	${}^1V_4 = {}^2V_4$; ${}^1V_1 = {}^1V_1$	$= 2n-1$	1	...	...	$2n-1$													
3	+	${}^1V_5 + {}^1V_1$	${}^2V_5$	${}^1V_5 = {}^2V_5$; ${}^1V_1 = {}^1V_1$	$= 2n+1$	1	...	...	...	$2n+1$												
4	÷	${}^2V_5 \div {}^2V_4$	${}^1V_{11}$	${}^2V_5 = {}^0V_5$; ${}^2V_4 = {}^0V_4$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$						
5	÷	${}^1V_{11} \div {}^1V_2$	${}^2V_{11}$	${}^1V_{11} = {}^2V_{11}$; ${}^1V_2 = {}^1V_2$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	${}^0V_{13} - {}^2V_{11}$	${}^1V_{13}$	${}^2V_{11} = {}^0V_{11}$; ${}^0V_{13} = {}^1V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	${}^1V_3 - {}^1V_1$	${}^1V_{10}$	${}^1V_3 = {}^1V_3$; ${}^1V_1 = {}^1V_1$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	${}^1V_2 + {}^0V_7$	${}^1V_7$	${}^1V_2 = {}^1V_2$; ${}^0V_7 = {}^1V_7$	$= 2 + 0 = 2$	...	2	...	...	...	...	2										
9	÷	${}^1V_6 \div {}^1V_7$	${}^3V_{11}$	${}^1V_6 = {}^1V_6$; ${}^0V_{11} = {}^3V_{11}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$						
10	×	${}^1V_{21} \times {}^3V_{11}$	${}^1V_{12}$	${}^1V_{21} = {}^1V_{21}$; ${}^3V_{11} = {}^3V_{11}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	.....	B_1			
11	+	${}^1V_{12} + {}^1V_{13}$	${}^2V_{13}$	${}^1V_{12} = {}^0V_{12}$; ${}^1V_{13} = {}^2V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	${}^1V_{10} - {}^1V_1$	${}^2V_{10}$	${}^1V_{10} = {}^2V_{10}$; ${}^1V_1 = {}^1V_1$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$							
13	−	${}^1V_6 - {}^1V_1$	${}^2V_6$	${}^1V_6 = {}^2V_6$; ${}^1V_1 = {}^1V_1$	$= 2n-1$	1	...	...	...	...	$2n-1$											
14	+	${}^1V_1 + {}^1V_7$	${}^2V_7$	${}^1V_1 = {}^1V_1$; ${}^1V_7 = {}^2V_7$	$= 2 + 1 = 3$	1	...	...	...	...	...	3										
15	÷	${}^2V_6 \div {}^2V_7$	${}^1V_8$	${}^2V_6 = {}^2V_6$; ${}^2V_7 = {}^2V_7$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$									
16	×	${}^1V_8 \times {}^3V_{11}$	${}^4V_{11}$	${}^1V_8 = {}^0V_8$; ${}^3V_{11} = {}^4V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	${}^2V_6 - {}^1V_1$	${}^3V_6$	${}^2V_6 = {}^3V_6$; ${}^1V_1 = {}^1V_1$	$= 2n-2$	1	...	...	...	...	$2n-2$											
18	+	${}^1V_1 + {}^2V_7$	${}^3V_7$	${}^2V_7 = {}^3V_7$; ${}^1V_1 = {}^1V_1$	$= 3 + 1 = 4$	1	...	...	...	...	...	4										
19	÷	${}^3V_6 \div {}^3V_7$	${}^1V_9$	${}^3V_6 = {}^3V_6$; ${}^3V_7 = {}^3V_7$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...	$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = A_3\right\}$						
20	×	${}^1V_9 \times {}^4V_{11}$	${}^5V_{11}$	${}^1V_9 = {}^0V_9$; ${}^4V_{11} = {}^5V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	...	0							
21	×	${}^1V_{22} \times {}^5V_{11}$	${}^0V_{12}$	${}^1V_{22} = {}^1V_{22}$; ${}^0V_{12} = {}^2V_{12}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	.....	.....	B_3		
22	+	${}^2V_{12} + {}^2V_{13}$	${}^3V_{13}$	${}^2V_{12} = {}^0V_{12}$; ${}^2V_{13} = {}^3V_{13}$	$= A_0 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	.....	0	$\{A_3 + B_1 A_1 + B_3 A_3\}$				
23	−	${}^2V_{10} - {}^1V_1$	${}^3V_{10}$	${}^2V_{10} = {}^3V_{10}$; ${}^1V_1 = {}^1V_1$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							

Here follows a repetition of Operations thirteen to twenty-three.

24	+	${}^4V_{13} + {}^0V_{24}$	${}^1V_{24}$	${}^4V_{13} = {}^0V_{13}$; ${}^0V_{24} = {}^1V_{24}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25	+	${}^1V_1 + {}^1V_2$	${}^1V_3$	${}^1V_1 = {}^1V_1$; ${}^1V_3 = {}^1V_3$; ${}^5V_6 = {}^0V_5$ by a Variable-card. ; ${}^5V_7 = {}^0V_7$ by a Variable-card.	$= n+1 = 4+1 = 5$	1	...	$n+1$	...	...	0	0											

diagrammi di flusso

- Il lavoro nella testa del calcolatore (e non solo)
  - Dal procedimento ai singoli passi
  - "Vederlo" prima di codificarlo
- Flow process charts 1921
  - Frank & Lillian Gilbreth
  - Scientific management
- Goldstine, Von Neumann 1947
  - Planning & Coding Problems
- Caracciolo, Fabri 1956
  - Sottoprogrammi I CEP



POLITECNICO DI MILANO
CENTRO DI CALCOLI NUMERICI

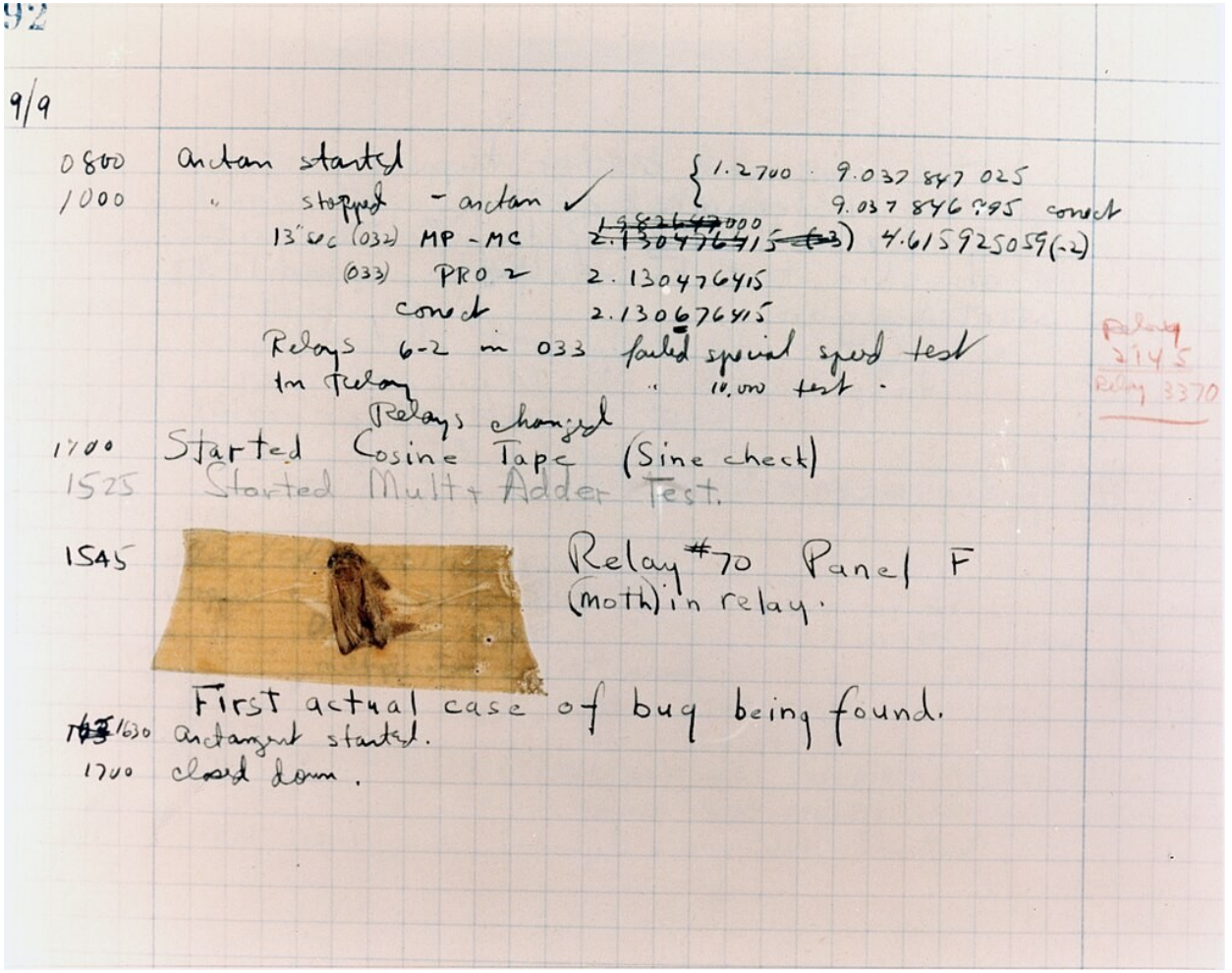
CALCOLATRICE ELETTRONICA CRC102A
Modulo per Programma

N. ..... Pos. .....
Nastro N. ..... Pos. .....
Data: 28.10.57 Pag. 1/11
Titolo: Simulazione della espressione per la somma f.p.
Compilatore: Zacolo

INDIRIZZO	ISTR.	m_1	m_2	m_3	NOTE
10 00	34	3000	2100	1500	
01	11	0162	↓	↓	
02	06	0045	0050	0200	
03	6	0200	1450	0200	$A_5 A_2 O_1 = a_1$
04	6	0046	0050	0201	
05	6	0201	1450	0201	$A_6 A_2 O_1 = a_2$
06	6	0045	0146	0202	
07	6	0145	0046	0203	
10	5	0202	0203	0204	
11	6	0204	1451	0205	$(A_5 A'_5 + A'_6 A_6) O_0 = a_3$
12	6	↓	0101	0206	$E_5 F'_2$
13	6	0206	1444	0207	$E_5 F'_2 G_E$
14	6	0100	0001	0210	$E'_5 F_2$
15	6	0210	1443	0211	$E'_5 F_2 S_E$
16	5	0207	0211	0212	$E_5 F'_2 G_E + E'_5 F_2 S_E = a_4$
17	6	0210	1444	0213	
20	6	0206	1443	0214	
21	5	0213	0214	0215	$E'_5 F_2 G_E + E_5 F'_2 S_E = a_4$
22	5	0206	0210	0216	
23	6	0216	1445	0217	$(E'_5 F_2 + E_5 F'_2) T_E = k$
24	4	0200	2100	0041	→ A_1
25	4	0201	2100	0042	→ A_2

Cortesia R. Dadda

i problemi, non solo hardware



Zuse e il Plankalkül, 1939-48

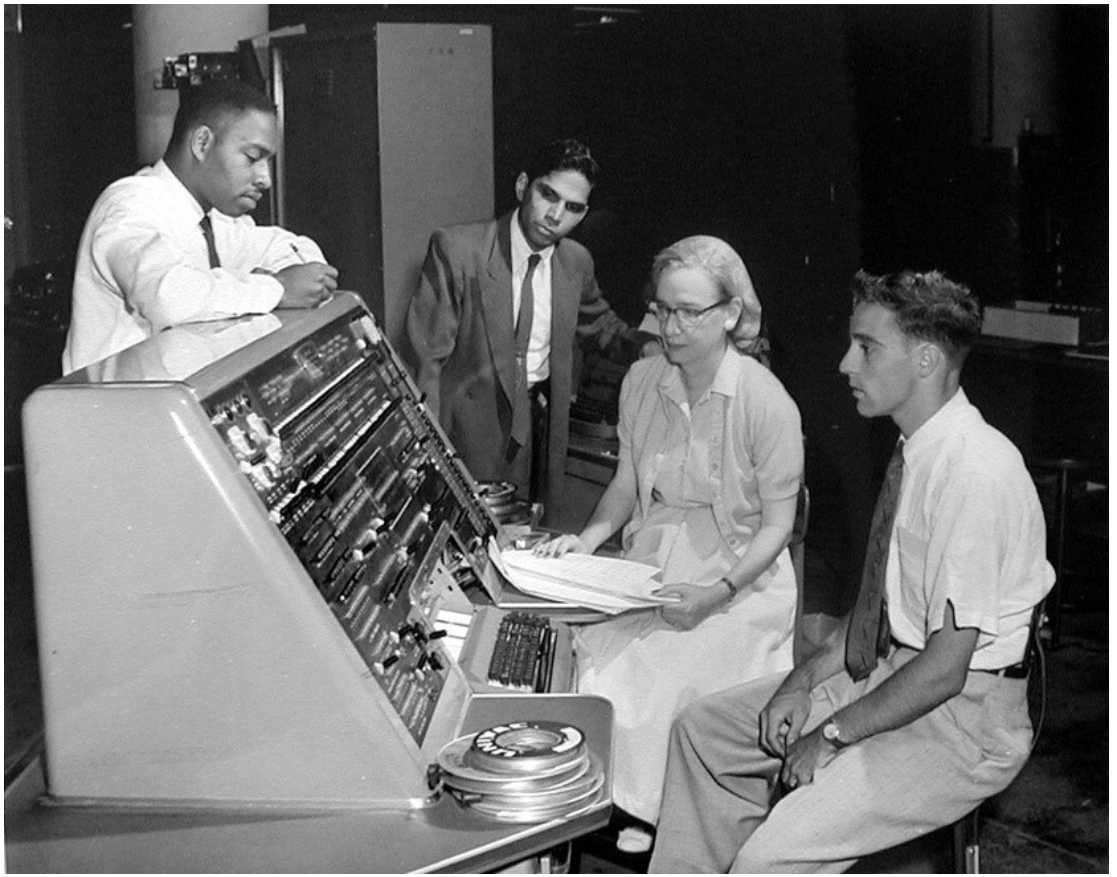
- Un sistema formale per pianificare azioni
  - In tedesco *kalkül* significa *sistema formale*
  - Logica, algebra relazionale, aritmetica binaria
  - Notazione bidimensionale, costrutti come tabelle
- Un lavoro parallelo alle macchine
  - Dalla Z1 del 1934-38, alla Z4 a Zurigo nel 1949
  - 1939 primi appunti, 1942 scacchi, 1945 descrizione
  - 1948 pubblicato su *Archiv der Mathematik*
- Riscoperta successiva
  - Citato dalla scuola svizzera
  - Primo compilatore nel 1975 (notazione lineare)

Autocode, 1951-61

- Una famiglia di *simplified coding systems*
  - Semplificare la programmazione e i programmi
  - Compilati, con inefficienza tollerabile
- Da Manchester...
  - 1951 Alick Glennie, Mark 1
  - 1954 Ralph A. Brooker, Ferranti Mk1, commerciale
- ... a Cambridge e oltre
  - 1961, David Hartley, EDSAC 2
  - Poi CPL sul Titan nel 1964 e poi sull'Atlas
    $\rightarrow$ BCPL (1967) $\rightarrow$ B (1969) $\rightarrow$ C (1972)

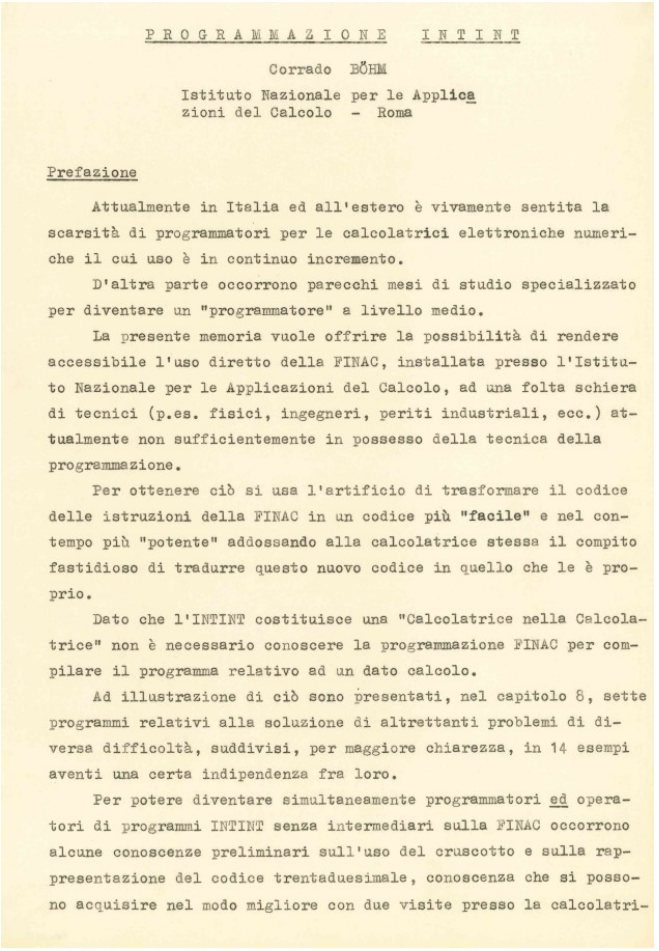
dalle parti di UNIVAC, 1951-55

- A-0
  - Loader & linker
  - Grace Hopper
- Seguiranno
  - A-3 Arith-matic
  - AT-3 Math-matic
  - B-0 Flow-matic
- Proto open source



INT INT, 1958

- INAC
  - Notiziario FINAC n. 2
  - 14 febbraio 1958
- Interprete e integratore
  - C. Böhm & D. Prinz
  - Calcolatrice nella calcolatrice
  - Librerie di sottoprogrammi
- Calcolatrice nella calcolatrice



PROGRAMMAZIONE INTINT

Corrado BÖHM

Istituto Nazionale per le Applica
zioni del Calcolo - Roma

Prefazione

Attualmente in Italia ed all'estero è vivamente sentita la scarsità di programmatori per le calcolatrici elettroniche numeriche il cui uso è in continuo incremento.

D'altra parte occorrono parecchi mesi di studio specializzato per diventare un "programmatore" a livello medio.

La presente memoria vuole offrire la possibilità di rendere accessibile l'uso diretto della FINAC, installata presso l'Istituto Nazionale per le Applicazioni del Calcolo, ad una folta schiera di tecnici (p.es. fisici, ingegneri, periti industriali, ecc.) attualmente non sufficientemente in possesso della tecnica della programmazione.

Per ottenere ciò si usa l'artificio di trasformare il codice delle istruzioni della FINAC in un codice più "facile" e nel contempo più "potente" addossando alla calcolatrice stessa il compito fastidioso di tradurre questo nuovo codice in quello che le è proprio.

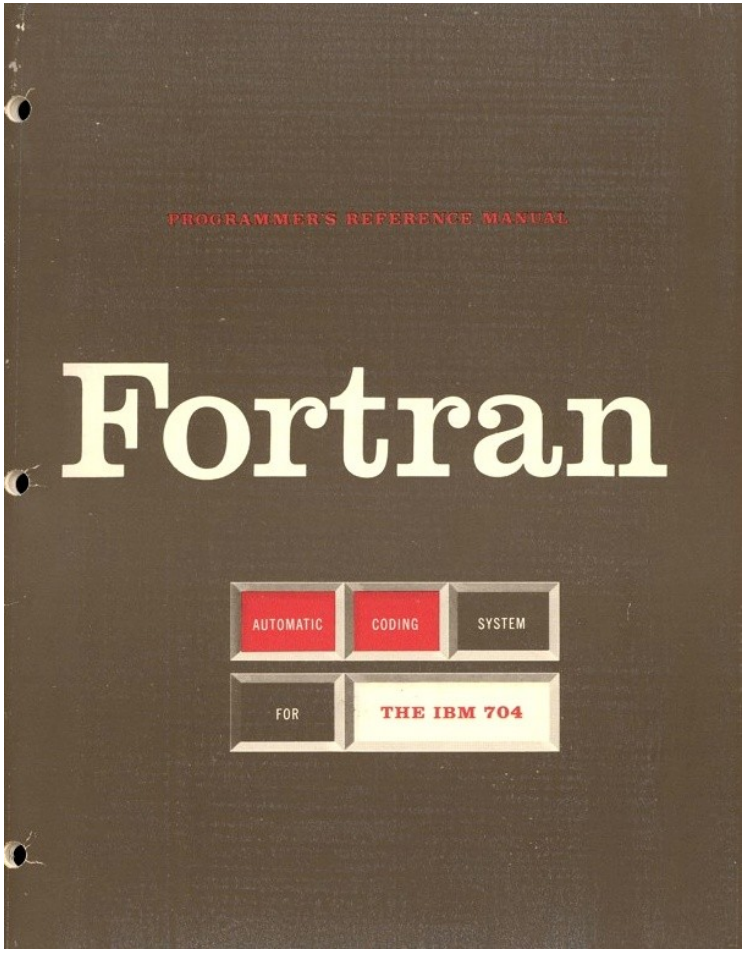
Dato che l'INTINT costituisce una "Calcolatrice nella Calcolatrice" non è necessario conoscere la programmazione FINAC per compilare il programma relativo ad un dato calcolo.

Ad illustrazione di ciò sono presentati, nel capitolo 8, sette programmi relativi alla soluzione di altrettanti problemi di diversa difficoltà, suddivisi, per maggiore chiarezza, in 14 esempi aventi una certa indipendenza fra loro.

Per potere diventare simultaneamente programmatori ed operatori di programmi INTINT senza intermediari sulla FINAC occorrono alcune conoscenze preliminari sull'uso del cruscotto e sulla rappresentazione del codice trentaduesimale, conoscenza che si possono acquisire nel modo migliore con due visite presso la calcolatri-

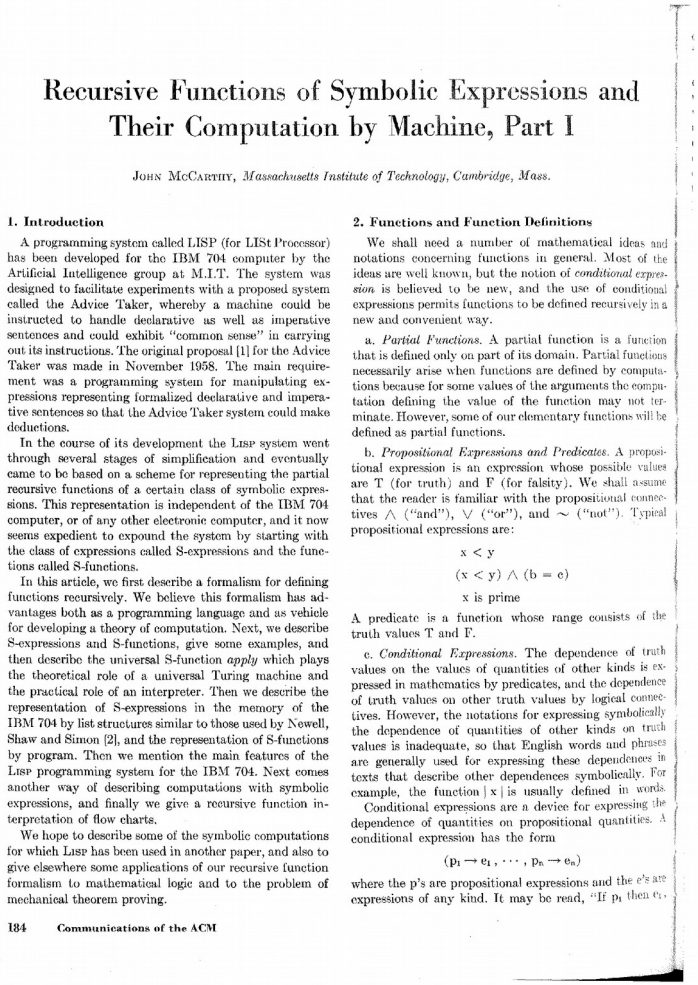
Fortran, 1956/58

- Sull'IBM 704
  - John W. Backus
  - Formula Translating System
  - Le minuscole non c'erano
- Possibili influenze
  - Laning & Zierler System 1952, Whirlwind
  - General Order Generator 1957, Eng. El. DEUCE
- Ancora nel TIOBE Idx



LISP, 1958/60

- AI MIT
  - Legato al λ calcolo di Church
  - John McCarthy e Steve Russell
- List Processing
  - Strutture dati sofisticate
  - Compilatore con l'interprete
- Intelligenza artificiale
  - Information Processing Lang. RAND (1955-57)
  - Logic Theorist



Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

JOHN MCCARTHY, *Massachusetts Institute of Technology, Cambridge, Mass.*

1. Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions. The original proposal [1] for the Advice Taker was made in November 1958. The main requirement was a programming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions.

In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a certain class of symbolic expressions. This representation is independent of the IBM 704 computer, or of any other electronic computer, and it now seems expedient to expound the system by starting with the class of expressions called S-expressions and the functions called S-functions.

In this article, we first describe a formalism for defining functions recursively. We believe this formalism has advantages both as a programming language and as vehicle for developing a theory of computation. Next, we describe S-expressions and S-functions, give some examples, and then describe the universal S-function *apply* which plays the theoretical role of a universal Turing machine and the practical role of an interpreter. Then we describe the representation of S-expressions in the memory of the IBM 704 by list structures similar to those used by Newell, Shaw and Simon [2], and the representation of S-functions by program. Then we mention the main features of the LISP programming system for the IBM 704. Next comes another way of describing computations with symbolic expressions, and finally we give a recursive function interpretation of flow charts.

We hope to describe some of the symbolic computations for which LISP has been used in another paper, and also to give elsewhere some applications of our recursive function formalism to mathematical logic and to the problem of mechanical theorem proving.

2. Functions and Function Definitions

We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional expressions permits functions to be defined recursively in a new and convenient way.

a. *Partial Functions.* A partial function is a function that is defined only on part of its domain. Partial functions necessarily arise when functions are defined by computations because for some values of the arguments the computation defining the value of the function may not terminate. However, some of our elementary functions will be defined as partial functions.

b. *Propositional Expressions and Predicates.* A propositional expression is an expression whose possible values are T (for truth) and F (for falsity). We shall assume that the reader is familiar with the propositional connectives $\wedge$ ("and"), $\vee$ ("or"), and $\sim$ ("not"). Typical propositional expressions are:

$$x < y$$

$$(x < y) \wedge (b = c)$$

$$x \text{ is prime}$$

A predicate is a function whose range consists of the truth values T and F.

c. *Conditional Expressions.* The dependence of truth values on the values of quantities of other kinds is expressed in mathematics by predicates, and the dependence of truth values on other truth values by logical connectives. However, the notations for expressing symbolically the dependence of quantities of other kinds on truth values is inadequate, so that English words and phrases are generally used for expressing these dependences in texts that describe other dependences symbolically. For example, the function $|x|$ is usually defined in words.

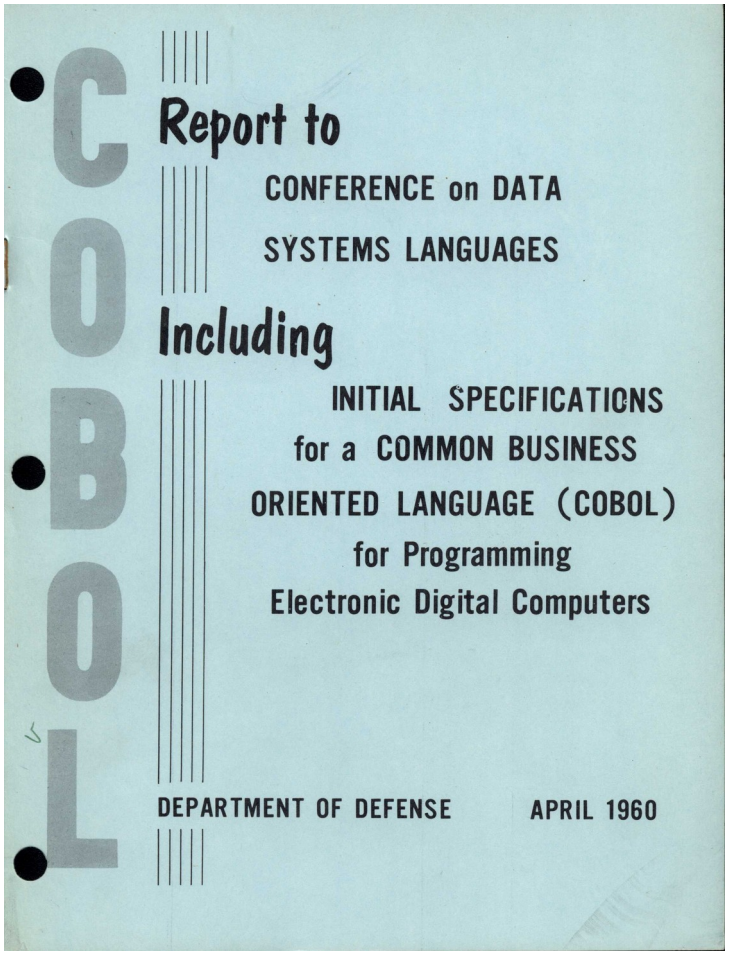
Conditional expressions are a device for expressing the dependence of quantities on propositional quantities. A conditional expression has the form

$$(p_1 \rightarrow e_1, \cdots, p_n \rightarrow e_n)$$

where the p's are propositional expressions and the e's are expressions of any kind. It may be read, "If p_1 then e_1,

COBOL, 1960

- CoDaSyL & COBOL
  - Committee on Data System Languages
  - Common Business Oriented Language
- Avvio maggio 1959
  - Pentagono, logistica, personale, commesse
  - Aziende: Burroughs, GE, Honeywell, IBM, Philco, Univac, RCA, Sylvania



Böhm, dalla tesi al teorema

- Corrado Böhm
  - Milano, Losanna, Zurigo, Roma, anche Pisa
  - Dottorato a Zurigo, contatti con la Z4 (poi noleggiata)
- Tesi di dottorato
  - 1951, una macchina e un linguaggio
  - 1952, un brevetto, senza seguito, ma citato
- Il teorema con Giuseppe Jacopini
  - Sequenza, condizione, iterazione
- Ci ricorda che macchine e linguaggi pari sono

per tutti – BASIC

- Beginners' All-purpose Symbolic Instruction Code
  - John J. Kemeny & Thomas E. Kurtz
  - Dartmouth Math Department
  - Serie di esperimenti di *programming literacy*
- Sul Dartmouth Time Sharing System (1964)
  - GE-225 e Teletype Model 33
  - Interprete interattivo
    e l'illusione di avere un "proprio" calcolatore
- Poi su PDP-n, Data General Nova, infine sui PC

formali – Algol, Pascal

- Backus-Naur form
  - John W. Backus (USA), Peter Naur (DK)
  - Conf. di Zurigo: International Algebraic Language
  - IAL $\rightarrow$ AlgoL 58, procedure, blocchi (begin/end)
  - Algol 60, blocchi con scope, ricorsione
  - Compilatore su Z22, successo accademico
- Dall'Algol W (1966) al Pascal (1970)
  - Niklaus E. Wirth (CH) e Tony Hoare (UK)
  - Strutture dati, tipi, annidamento delle definizioni
  - Non solo accademia:
    soliti PDP-11, ma poi Apple, Borland, Delphi...

ancora più formali – Prolog

- Programmation en Logique
  - *Horn clauses* (1951, Alfred Horn), un formalismo TE
  - Robert A. Kowalski, *SLD resolution* (1970)
  - Alain Colmerauer & Philippe Roussel (1972)
- La scuola di Edimburgo (Kowalski)
  - David H.D Warren, Abstract Machine e compilatore
  - Sintassi e semantica standard
  - Europa vs USA: Prolog vs Lisp
- Non diffusissimo, ma ci sono usi notevoli
  - Sistemi esperti
  - IBM Watson

Oggetti, da C a C++

- Dal MIT a Oslo allo Xerox PARC
  - LISP (1960), Sketchpad (1961), MIT Algol (1968) oggetti, istanze, metodi
  - Simula (1961-67), ereditarietà, procedure virtuali
  - Smalltalk (1970), object-oriented programming
- Ai Bell Labs: BCPL, B, C (1967-69-70)
  - Dennis Ritchie, Ken Thompson, Unix & strumenti
  - Compatto (librerie), efficiente, compilazione inc.
- Dalla Danimarca ai Bell Labs
  - Bjarne Stroustrup, C with Classes (1979) C++ (1982)

SQL, dichiarativo, ma...

- Modello relazionale, Ted Codd (IBM), 1970
  - Una soluzione algebrica, un livello applicativo
  - IBM System R ('74), Berkeley InGReS ('75), Honeywell Multics ('76), Relational Sw. Oracle ('79)
  - Nessun RDBMS propriamente fedele al modello
- SQUARE, QueL, SeQUEL, SQL (1979)
  - Specifying QUeries in A Rel. Env. (System R)
  - Query Lanquage (Ingres)
  - Cosa, non come
- Formalmente, un linguaggio solo dal 2003
  - Windowing & transitive closure

WORE e Java

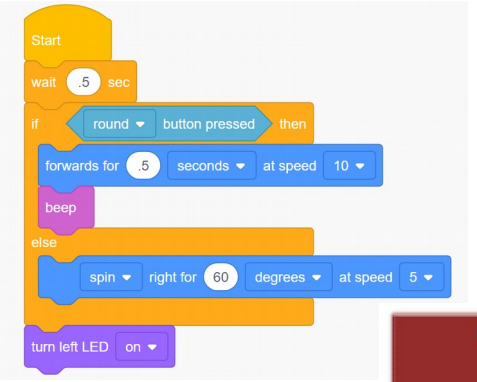
- Write Once and Run Everywhere
  - Precedenti: dal P-Code alla compilazione in C
  - Uno slogan, un buon sorgente compila ovunque
  - Non distribuire sorgenti, ma Bytecode non leggibile
- Java, dalla Sun a Oracle
  - James Gosling, Mike Sheridan, Patrick Naughton
  - Dal 1991, Oak, Green, Java (dal caffè), 1.0 1996
  - Fortuna con le applet web (eliminate 2015-17)
  - Gran promozione dopo l'acquisizione (2009/10)
- Sintassi e semantica stile C/C++

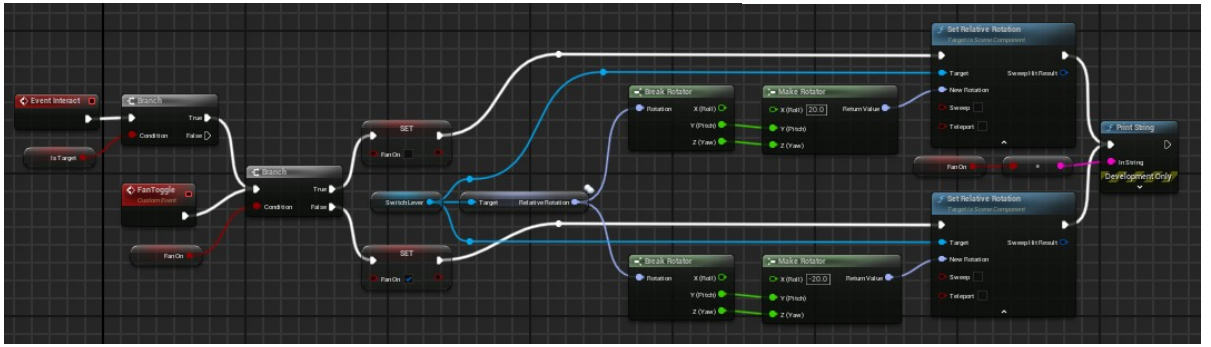
Il mondo del web (ex WWW)

- Rivisitazione dell'architettura client-server
  - Dai terminali dei mainframe, a webserver & browser
  - Due macchine universali su cui distribuire il calcolo
- Ipertesti in rete: HTTP, URL e HTML
  - Protocollo, identificazione delle risorse, linguaggio
  - Tim Berners-Lee & Robert Calliau, CERN (1989/94)
- Linguaggi, un esempio di stack
  - Server: PHP/SQL; client: HTML/CSS/Javascript
- Java is to Javascript as car is to carpet
  - Netscape 1995, inizialmente Livescript

Programmazione visuale

- Educativi
  - Scratch, 2003 MIT
  - StarLogo TNG, 2008 MIT
- RAD
  - Se ne parla dagli anni ’70…
  - UE Blueprints dal 4 (2014)
  - Ex scripting, e.g. Blender

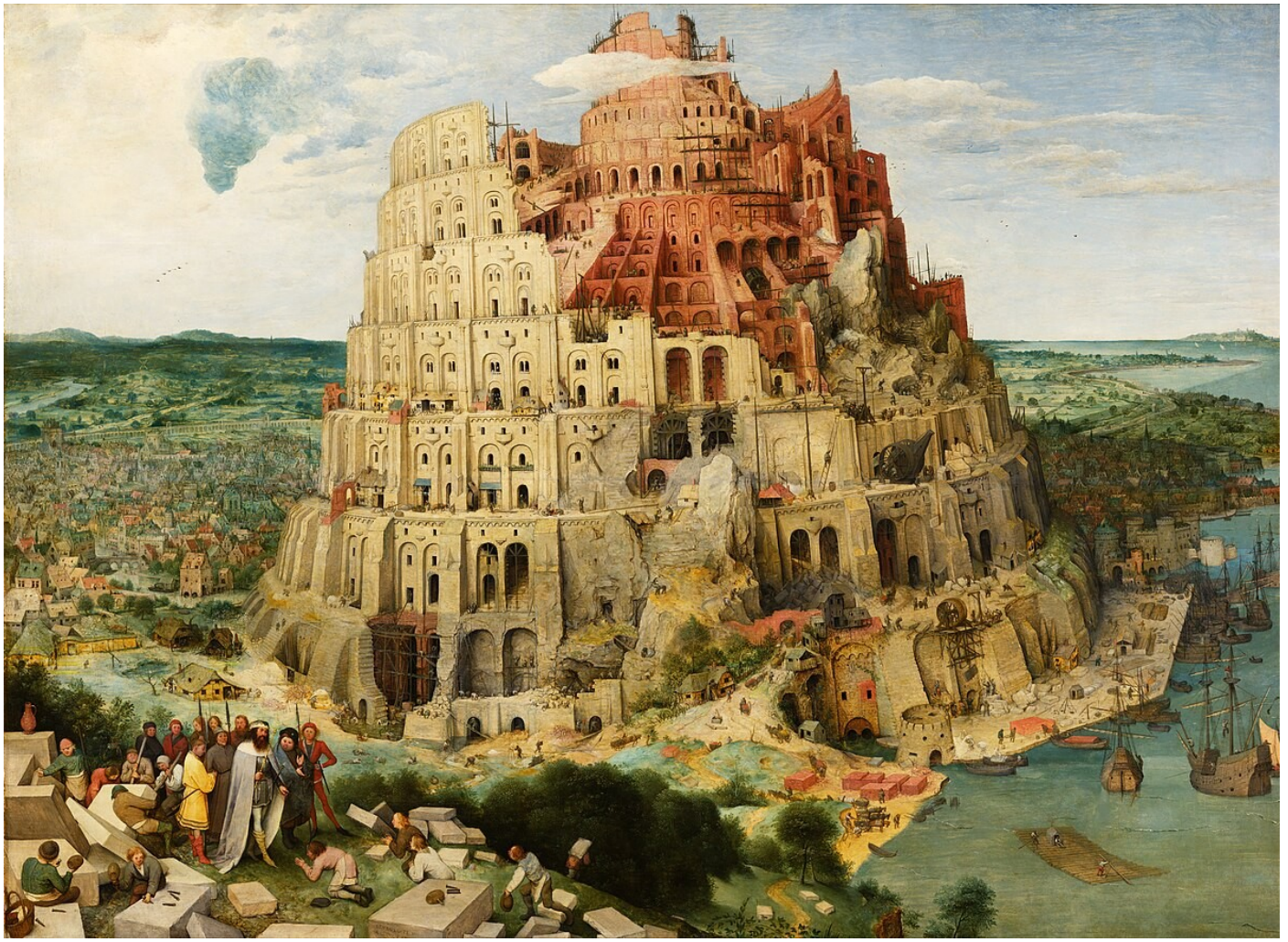




Free Software e Open Source

- Linguaggi per scrivere/capire meglio
  - Obiettivo negato dall'indisponibilità dei sorgenti
  - Negata anche la modifica, per adattare e migliorare
- Free Software Movement, 1983, MIT
  - Richard Stallmann, e Minsky, Sussman, Winston...
  - GNU is Not Unix, GPL, GLPL
  - Essere "open source" è un requisito
  - Free as in *free speech*, not *free beer*
- Evoluzione del copyright e delle relative economie

una Babele?



riferimenti

- D. Knuth, L. Trabb Pardo “The Early Development of Programming Languages”, rapp. tec. STAN-CS-76-562, Stanford University, 1976
- “About the FSF”, Free Software Foundation