

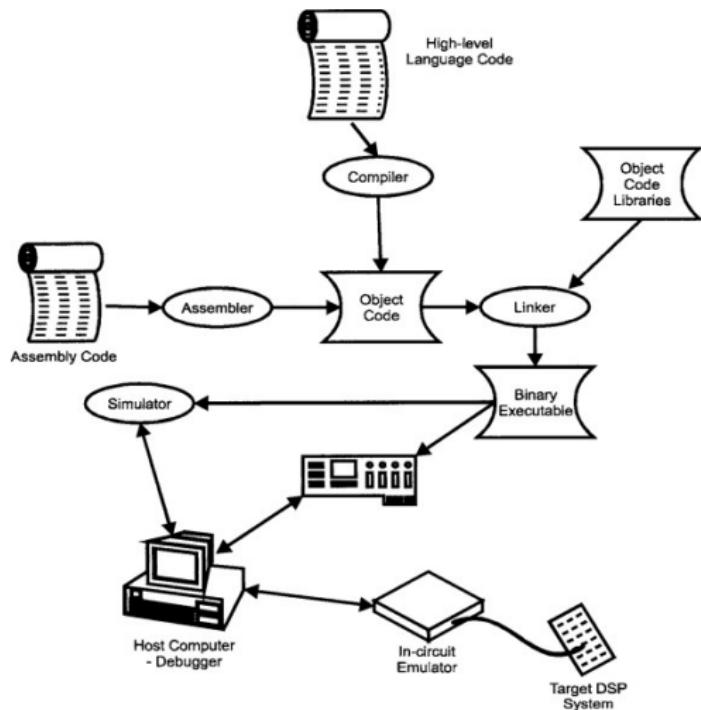
A Brief History of Programming

Cem Bozşahin, Cognitive Science, METU

(starred images and videos courtesy of the web community, ta!)

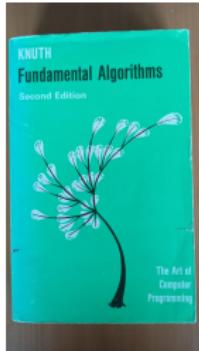
February 29, 2024

- Short: programming is proxying work in worked-out detail.
- Longer: programming is detailing control for something (someone?), to do something on its own.
- 'On its own' means no external interpreter.
- Auto-programming: Can we fully automate programming?
- It requires
 - Programmer: currently, human beings.
 - Programmed: currently, an automaton.
 - What to program: reasons for control.
 - How to program: uses of control.
- Therefore, we need to be specific about
 - control and automaton
 - the act of 'giving'



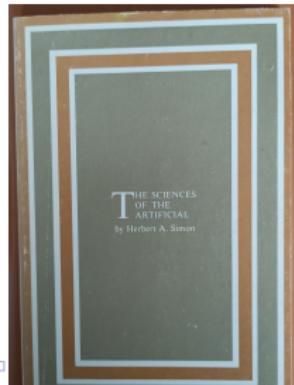
- There was programming before the computer.
- That's why it is called Computer Programming.
- The questions above aren't well understood, but computer programming is better understood.
- Knuth (1974) is a good start for studying the connection.

- Computer programming seems to be a combination of art and science, and combination itself requires careful engineering.
 - The art of writing beautiful computer programs



(we know one when we see one, Gombrich 1950-style)

- The science of understanding their limits



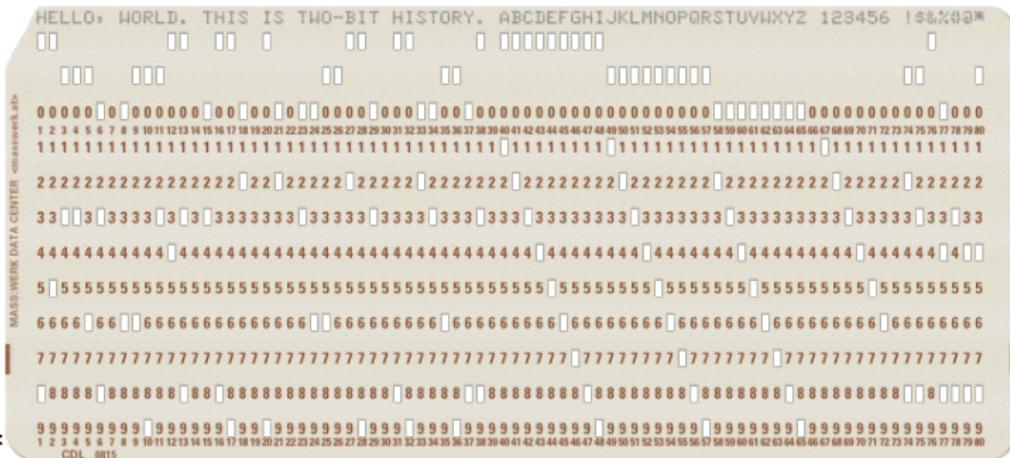
- The word **programming** comes from patterning the weaving machine: The Joseph Marie Jacquard Loom, 1804.



*

* The JMJ Loom in action

- The pattern designer gives the pattern in the form of a program, and the loom can weave it **autonomously** (well, almost).
- The machine could be operated by hand, but mass production required programming.
- And, of course, somebody had to **design** the weaving machine for us to operate it, either by hand or by instruction.
- And, somebody had to design a **language** for patterning, something that can be **physically realized** by an automaton.
- That's the story of the beloved **punched card** of Hollerith.



- Why did the loom operate semi-autonomously?
- Because its mechanical parts worked on kinetic feedback, and got off their rails once in a while,
- and the program could not continue, requiring **skilled** human intervention taking **long** repair times.
- Until Sakichi Toyoda came up with the **automatic power loom**.



Perfect? No. But **one unskilled** worker could fix 20 looms **quickly**.

Make that **zero**, and you've got the **computer**.

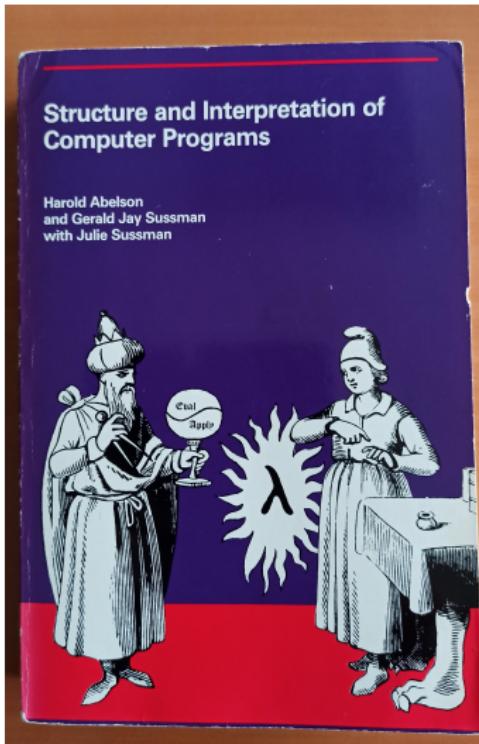
Take **unskilled** to mean **automated**.

What's in a word?

- The word **computer** used to mean ‘human calculator’.
- Sieg (2002) prefers **computor** for such humans, leaving **computer** to automata.
- The Turkish word, **bilgisayar**, literally ‘knowledge counter’, or maybe ‘information counter’, as a late neologism by Aydın Köksal, is specific to the gadget or what it takes.
- My favorite term is the French one, **ordinateur**. No knowledge, no information, no ‘knower’. It will last. NB. **counting** survives.
- Then the **programmer** must start the process of knowledge construction, and explore.
- It started as a trademark of IBM France, then frigidairized by public.
- We will see that it may not be digital chauvinism.

Philosophy meets explorers

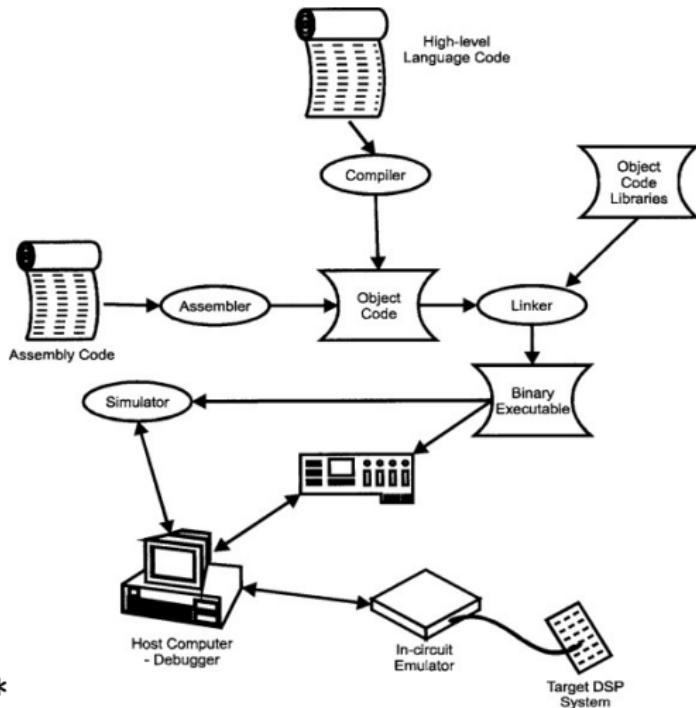
- My take: Program is a more fundamental object than algorithm.
- Algorithm has longer history than program:
Euclides, Archimedes, El Buruni, El Harizmi, Ibn Al-Banna,
Napier
- They created algorithms for other humans to follow.
- A computer program demands that you put yourself in the computer's shoes in thinking about control and behavior.
- That in itself is exploratory thinking.
- The computer is no longer the end of a thinking cycle, it might be the beginning of knowledge construction by exploration.



► * The cover?

"Underlying our approach to this subject is our conviction that 'computer science' is not a science and that its significance has little to do with computers. The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of change is the emergence of what might best be called *procedural epistemology*—the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Mathematics provides a framework for dealing precisely with notions of 'what is'. Computation provides a framework for dealing precisely with notions of 'how to'." Abelson et al. (1985)

Computer programming cycle



- We want to be able to explore complex problems.
- If the mechanism is equally complex, that exploration isn't very promising.
- Can complex behavior arise from a simple mechanism?
- Formulable and solvable problems.

Can we find an item in a collection of records?

- Formulable but unsolvable problems (The Halting Problem)

diagonal(X):

a: if $halt(X,X)$ goto a otherwise halt.

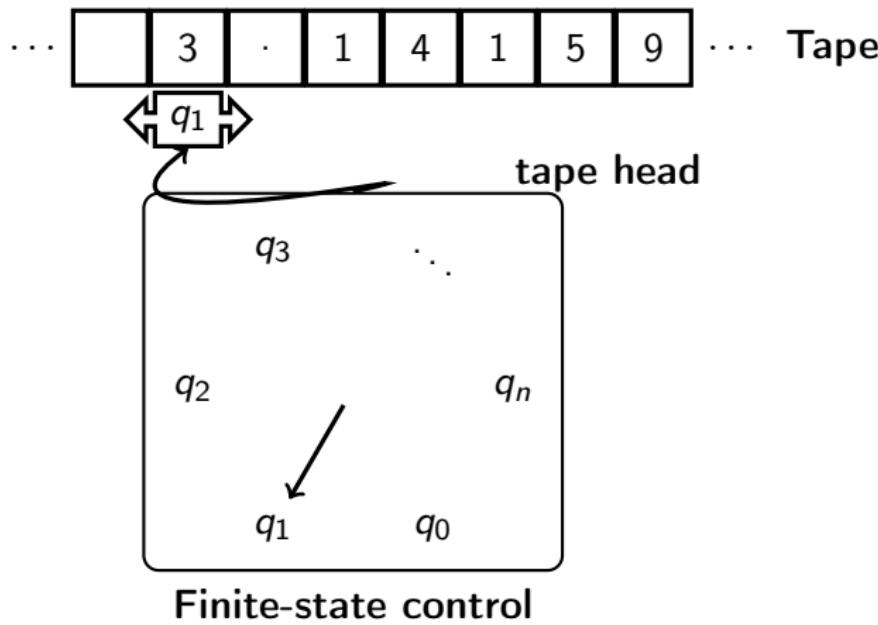
diagonal(diagonal) ?

- Expressible but unformulable problems

What is the next number after π ?

- The difference is TM representability.

π is transfinite!



- Turing's universal combinator from 1937
- $\mathbf{U}fg = g(\mathbf{ffg})$

$$\mathbf{U} = (\lambda x \lambda y. y(xxy))(\lambda x \lambda y. y(xxy))$$

- $\mathbf{U}f = [(\lambda x \lambda y. y(xxy))(\lambda x \lambda y. y(xxy))] f$
- $[\lambda y. y([\lambda x_1 \lambda y_1. y_1(x_1 x_1 y_1)] [\lambda x_2 \lambda y_2. y_2(x_2 x_2 y_2)] y)] f$
- $f([\lambda x_1 \lambda y_1. y_1(x_1 x_1 y_1)] [\lambda x_2 \lambda y_2. y_2(x_2 x_2 y_2)] f)$
- $f(\mathbf{U}f)$

$$\mathbf{U}f = f(\mathbf{U}f)$$

- Recursion without names, programs without variables!

- Recursion by value is common in the natural world (language, planning, social organization, molecular structure, membranes, etc.; see e.g. Păun 2000, Searle 2005)
- Recursion by name is unheard of, except in a computer.
- If all kinds of recursion can be eliminated from a program, what's the point? (see e.g. Bozşahin 2016)
- It allows us to discuss (not just chat about) Cartesian Theater by making explicit the pitfalls of infinite regress. Dennett 1991
- How can a computer address change if all variables can be eliminated from a program before it runs?
(free variables, side effects—recall *procedural epistemology*, state change, configuration)

Two facets of computer science, and the loop

- 1 The inward concern: what are the limits of computation, its terms, realizations, data and control structures?
 - 2 The outward concern: given 1, what kind of empirical questions can be explored?
- 2-1** What can findings in 2 say about 1 in return?
- Having to program for 2, rather than staying at the algorithmic level, forces us to think like builders.
 - It might just start with having fun in the exploration.
Try 'Are Toy Problems Useful?' in Knuth (1996).

Two fundamental questions from computer science

For 1 Knuth 1996:3 question: What can be automated?

read: what kinds of data/control do the programmer/computer scientist need for the computer to make it work on its own?

For 2 Marr 1977 question: Can we say something about the nature of a problem by computationalizing it, not just to enumerate its solutions?

The world does not seem to consist solely of problems whose solutions can be found just as easily as a given solution can be checked.

- Abelson, H., G. J. Sussman, and J. Sussman (1985). *Structure and Interpretation of Computer Programs*. MIT Press.
- Bozşahin, C. (2016). Natural recursion doesn't work that way: Automata in planning and syntax. In V. Müller (Ed.), *Fundamental issues of AI*. Springer.
- Dennett, D. C. (1991). *Consciousness explained*. New York: Little Brown & Co.
- Gombrich, E. (1950). *The story of art*. Phaidon London.
- Knuth, D. E. (1974). Computer programming as an art. *Communications of the ACM* 17(12), 667–673.
- Knuth, D. E. (1996). *Selected papers on computer science*. Cambridge University Press.
- Marr, D. (1977). Artificial intelligence: A personal view. *Artificial Intelligence* 9, 37–48.
- Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences* 61(1), 108–143.
- Searle, J. R. (2005). What is an institution. *Journal of institutional economics* 1(1), 1–22.
- Sieg, W. (2002). Calculations by man and machine: conceptual analysis. In W. Sieg, R. Sommer, and C. Talcott (Eds.), *Reflections on the foundations of mathematics: Essays in honor of Solomon Feferman*.
- Turing, A. M. (1937). Computability and λ -definability. *J. of Symbolic Logic* 2(4), 153–163.