CS2110 Spring 2014. Concluding Lecture: History, Correctness Issues, Summary

Final review session: Fri, 9 May. 1:00–3:00. Phillips 101.

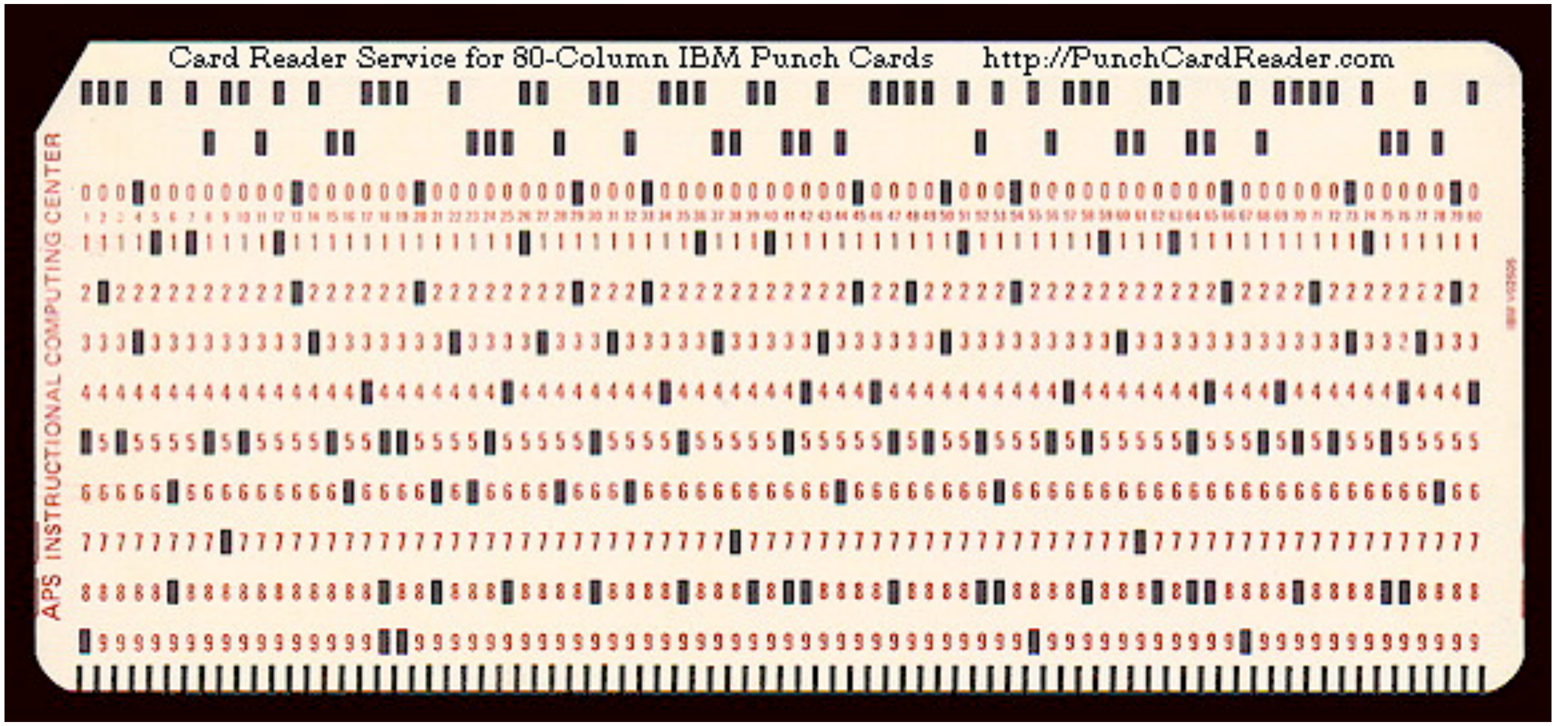
Final: 7:00–9:30PM, Monday, 12 May, Barton Hall

We hope to get you tentative course grades by Wednesday noon, but it may be later. You then visit the CMS and do the assignment to tell us whether you accept the grade or will take the final. There will be a message on the Piazza and the CMS about this when the tentative course grades are available.

CS2110 Spring 2014. Concluding Lecture: History, Correctness Issues, Summary

Programming and computers:

Momentous changes since the 1940s –or since even the use of punch cards and attempt at automation …


Card Reader Service for 80-Column IBM Punch Cards http://PunchCardReader.com
APS INSTRUCTIONAL COMPUTING CENTER

Punch cards

Jacquard loom

Loom still used in China

Mechanical loom invented by Joseph Marie Jacquard in 1801.

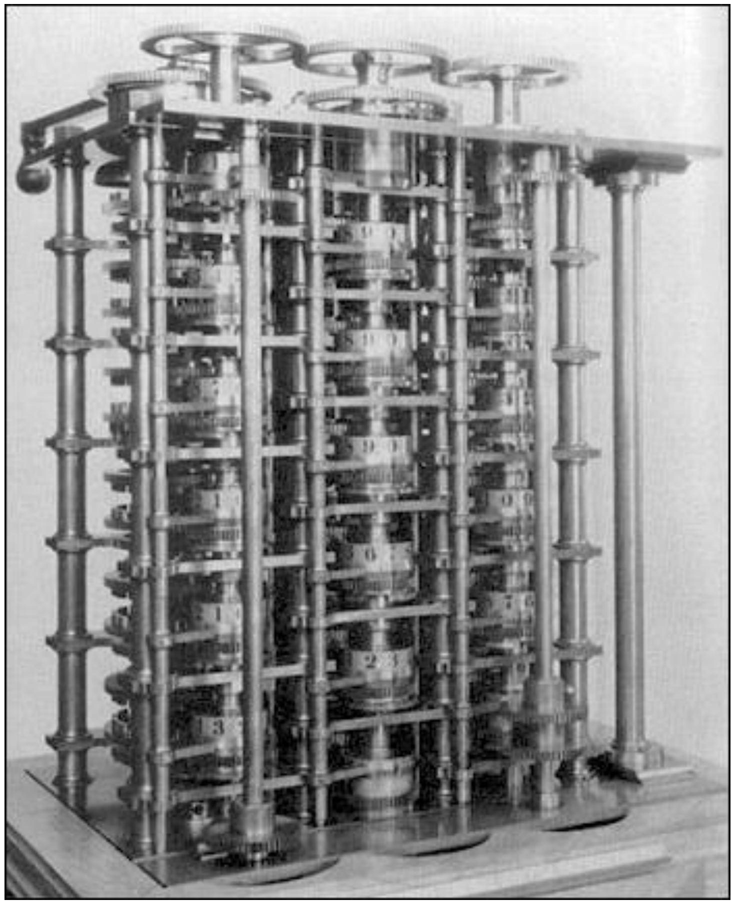
Used the holes punched in pasteboard punch cards to control the weaving of patterns in fabric.

Punch card corresponds to one row of the design.

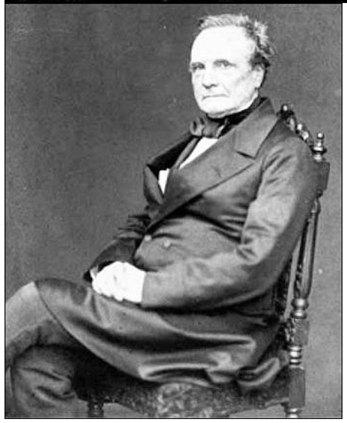
Based on earlier invention by French mechanic Falcon in 1728.

Charles Babbage designed a “difference engine” in 1822

Compute mathematical tables for log, sin, cos, other trigonometric functions.



No electricity



The mathematicians doing the calculations were called computers

Oxford English Dictionary, 1971

Computer: one who computes; a calculator, rekoner. spec. a person employed to make calculations in an observatory, in surveying. etc.

1664: Sir T. Browne. The calendars of these computers.

1704. T. Swift. A very skillful computer.

1744. Walpole. Told by some nice computers of national glory.

1855. Brewster Newton. To pay the expenses of a computer for reducing his observations.

The mathematicians doing the calculations were called computers

Charles Babbage planned to use cards to store programs in his Analytical engine. (First designs of real computers, middle 1800s until his death in 1871.)

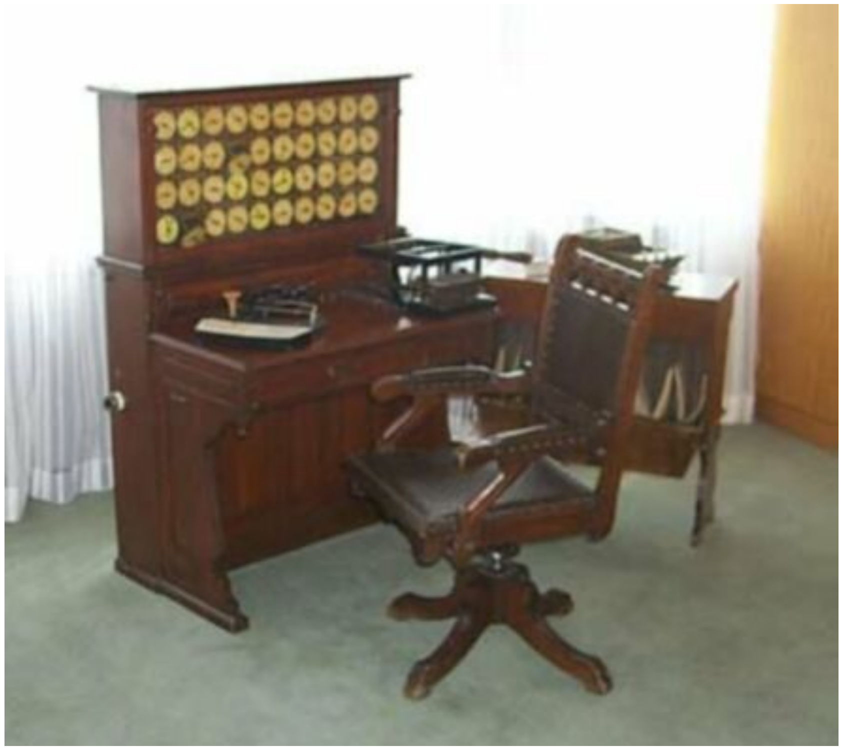
First programmer was Ada Lovelace, daughter of poet Lord Byron.

Privately schooled in math. One tutor was Augustus De Morgan.

The Right Honourable Augusta Ada, Countess of Lovelace.

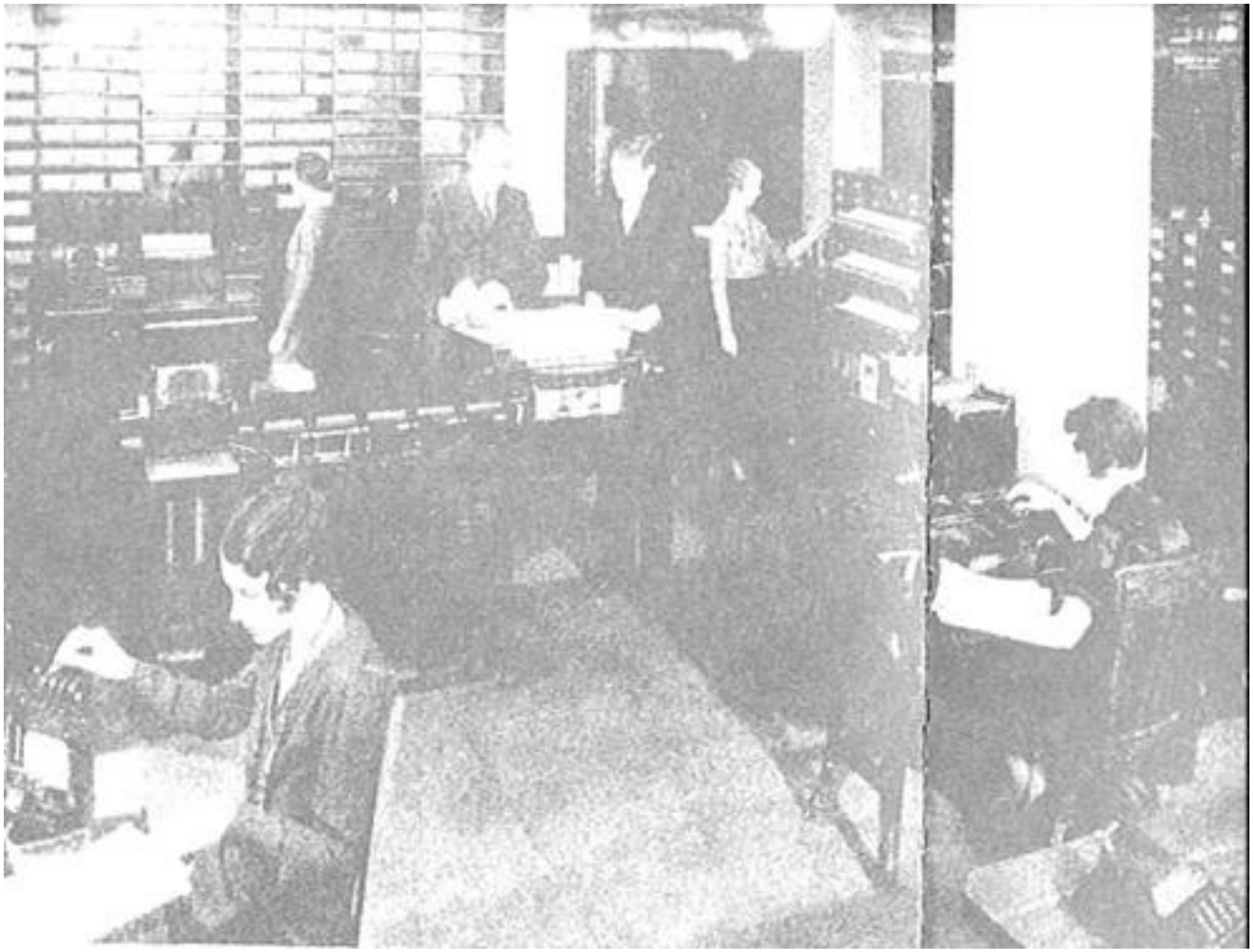
Herman Hollerith.
His tabulating machines used in compiling the 1890 Census.
Hollerith's patents were acquired by the Computing-Tabulating-Recording Co.
Later became **IBM**.

The operator places each card in the reader, pulls down a lever, and removes the card after each punched hole is counted.



Hollerith 1890 Census Tabulator

Computers, calculating the US census



History of computers

1935-38. Konrad Zuse - **Z1 Computer**

1935-39. John Atanasoff and Berry (grad student). Iowa State

1944. Howard Aiken & Grace Hopper **Harvard Mark I Computer**

1946. John Presper Eckert & John W. Mauchly **ENIAC 1 Computer** 20,000 vacuum tubes later ...

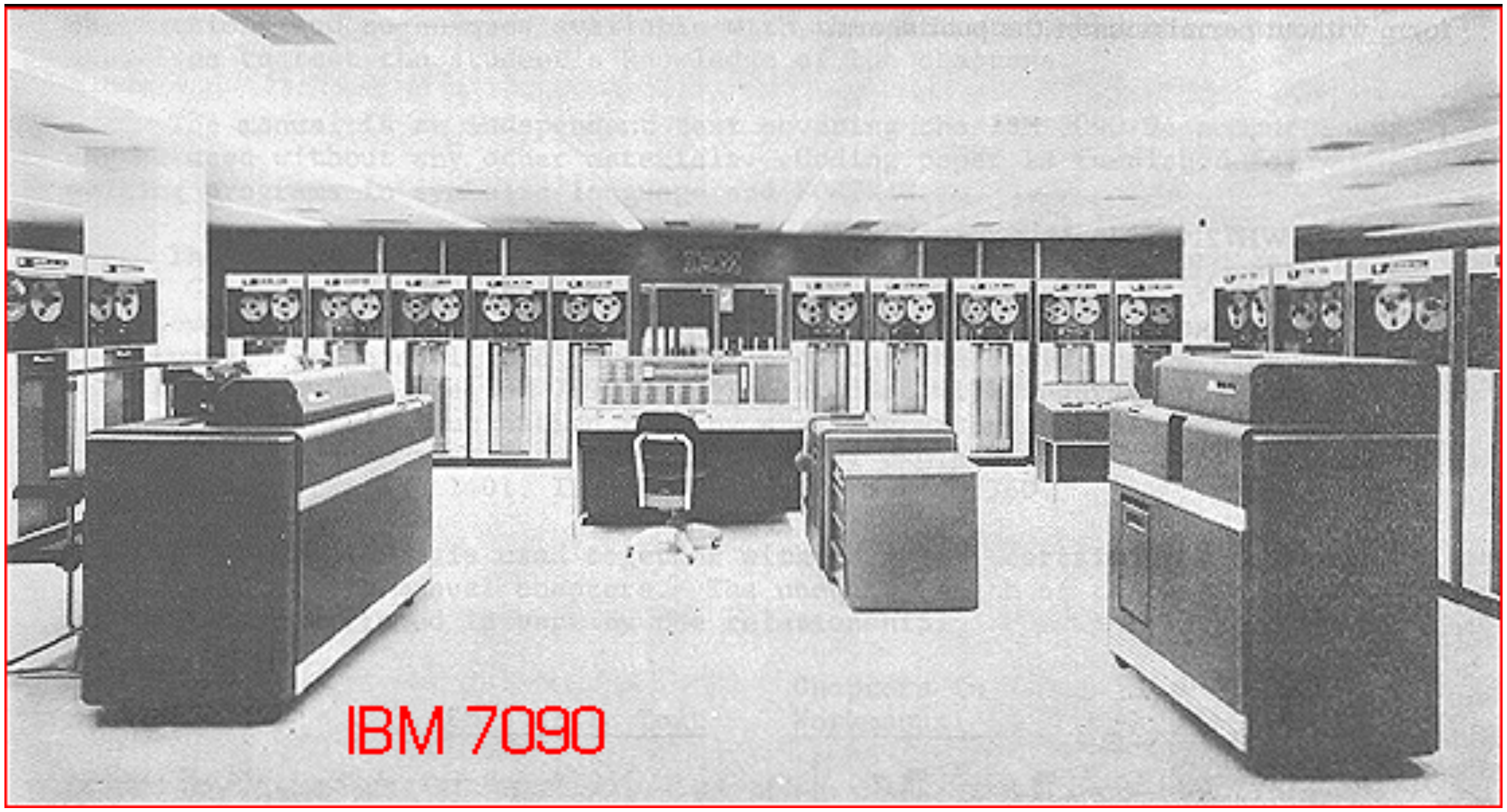
1947-48 The Transistor, at Bell-labs.

1953. IBM. the IBM 701.

How did Gries get into Computer Science?

1959. Took his only computer course. Senior, Queens College.

1960. Mathematician-programmer at the US Naval Weapons Lab in Dahlgren, Virginia.



1960. Mathematician-programmer at the US Naval Weapons Lab in Dahlgren, Virginia.

Programmed in Fortran and IBM 7090 assembly language

```
if (SEX ==  ‘M’ ) MALES= MALES + 1;
else FEMALES= FEMALES + 1;
```

```
         CLI   SEX,'M'      Male?
         BNO   IS_FEM       If not, branch around
         L     7,MALES      Load MALES into register 7;
         LA    7,1(,7)      add 1;
         ST    7,MALES      and store the result
         B     GO_ON        Finished with this portion
IS_FEM   L     7,FEMALES    If not male, load FEMALES into register 7;
         LA    7,1(,7)      add 1;
         ST    7,FEMALES    and store
GO_ON    EQU   *
```

1960: Big Year for Programming Languages

LISP (List Processor): McCarthy, MIT (moved to Stanford). First functional programming language. No assignment statement. Write everything as recursive functions.

COBOL (Common Business-Oriented Language). Became most widely used language for business, data processing.

ALGOL (Algorithmic Language). Developed by an international team over a 3-year period. McCarthy was on it, John Backus was on it (developed Fortran in mid 1950's). Gries's soon-to-be PhD supervisor, Fritz Bauer of Munich, led the team.

1959. Took his only computer course. Senior, Queens College.

1960. Mathematician-programmer at the US Naval Weapons Lab in Dahlgren, Virginia.

1962. Back to grad school, in Math, at University of Illinois

Graduate Assistantship: Help two Germans write the ALCOR-Illinois 7090 Compiler.

> John Backus, FORTRAN, mid 1950’s: 30 people years
>
> This compiler: 6 ~people-years
>
> Today, CS compiler writing course: 2 students, one semester

1963-66 Dr. rer. nat. in Math in Munich Institute of Technology

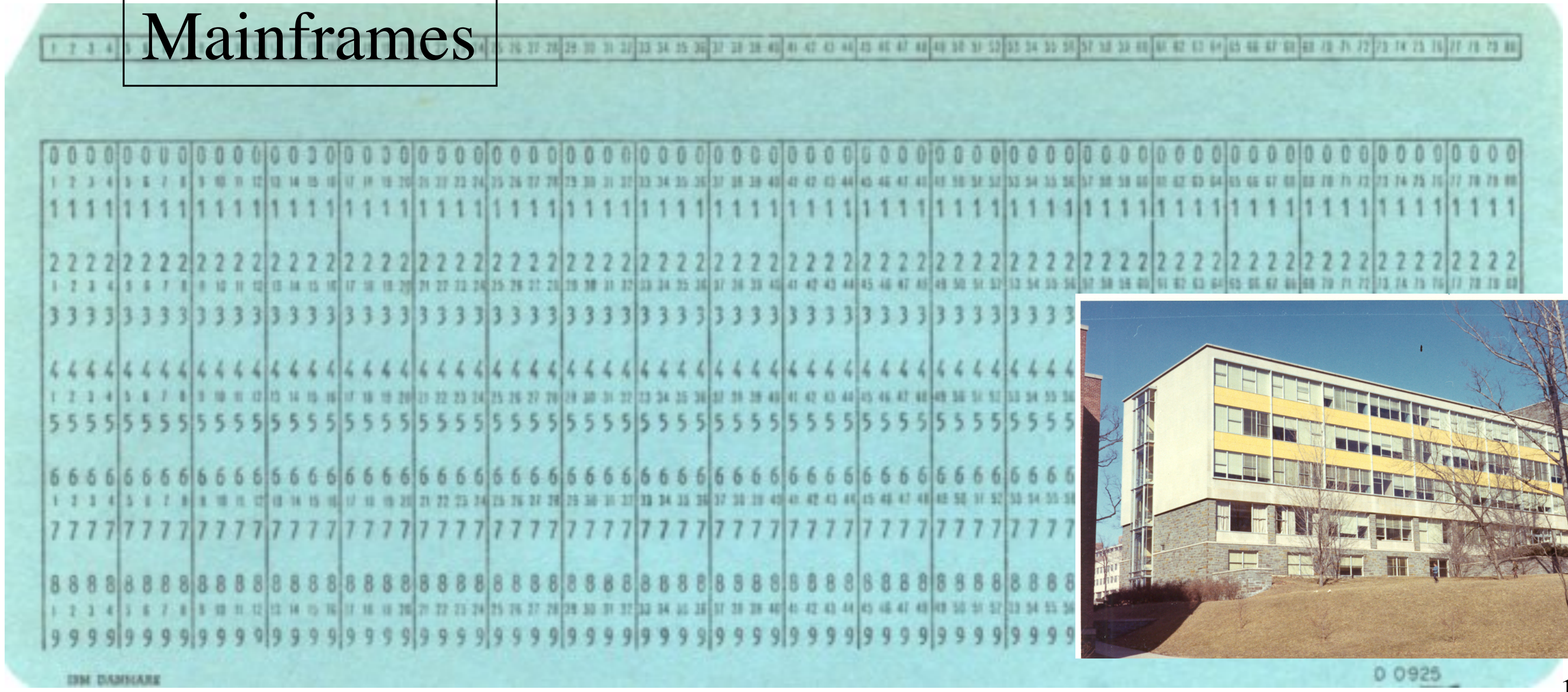
1966-69 Asst. Professor, Stanford CS

1969- Cornell!

Late 1960s

IBM 360 Mainframes

Write programs on IBM “punch cards. Deck of cards making up a program trucked to Langmuir labs by the airport 2-3 times a day; get them back, with output, 3-4 hours later

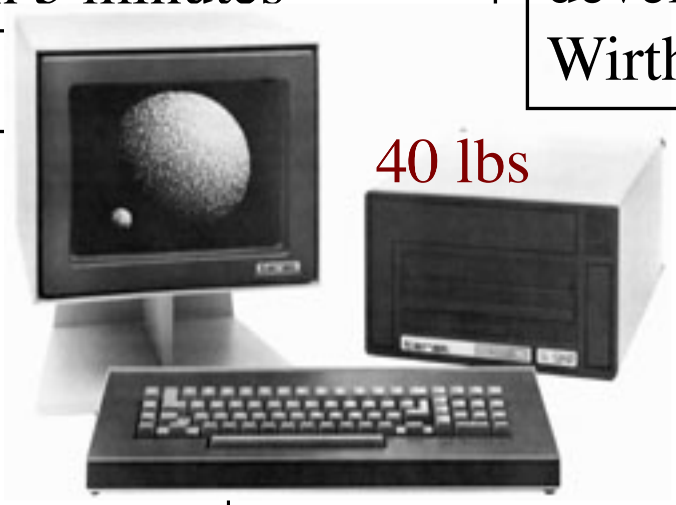


About 1973. BIG STEP FORWARD

1. Write program on punch cards.
2. Wait in line (20 min) to put cards in card reader in Upson basement
3. Output comes back in 5 minutes

About 1973. BIG STEP FORWARD

Switched to using the programming language Pascal, developed by Niklaus Wirth at Stanford.



About 1979. Teraks

Prof. Tim Teitelbaum sees opportunity. He and grad student Tom Reps develop “Cornell Program Synthesizer”. Year later, Cornell uses Teraks in its prog course.

November 1981, Terak with 56K RAM, one floppy drive: $8,935.

Want 10MB hard drive? $8,000 more

1983-84

Switched to Macintosh in labs

Late 1980s

Put fifth floor addition on Upson. We made the case that our labs were in our office and therefore we need bigger offices.

1980s

CS began getting computers on their desks.

Nowadays

Everybody has a computer in their office.

2014

Moved into Gates Hall!

Programming languages. Dates approximate

Year	Major languages	Teach at Cornell
1956's	Fortran	
1960	Algol, LISP, COBOL	
1965	PL/I	PL/C (1969)
1970	C	
1972	Pascal	
1980's	Smalltalk (object-oriented)	Pascal (1980's)
1980's	(late) C++	
1996	Java	C and C++
1998		Java / Matlab

The NATO Software Engineering Conferences

homepages.cs.ncl.ac.uk/brian.randell/NATO/

7-11 Oct 1968, Garmisch, Germany
27-31 Oct 1969, Rome, Italy

Download Proceedings, which
have transcripts of discussions.
See photographs.

Software crisis:
Academic and industrial people.
Admitted for first time that they did
not know how to develop software
efficiently and effectively.

SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

January 1969

Software Engineering, 1968

Next 10-15 years: intense period of research on software engineering, language design, proving programs correct, etc.

Software Engineering, 1968

During 1970s, 1980s, intense research on
How to prove programs correct,
How to make it practical,
Methodology for developing algorithms

The way we understand recursive methods is based on that methodology.
Our understanding of and development of loops is based on that methodology.

Throughout, we try to give you thought habits and strategies to help you solve programming problems for effectively, e.g.
Write good method specs.
Keep methods short.
Use method calls to eliminate nested loops.
Put local variable declarations near first use.

Mark Twain: Nothing needs changing so much as the habits of **others**.

The way we understand recursive methods is based on that methodology.
Our understanding of and development of loops is based on that methodology.

Simplicity is key:
Learn not only to simplify, learn not to complify.

Separate concerns, and focus on one at a time.

Develop and test incrementally.

Throughout, we try to give you thought habits to help you solve programming problems for effectively

Don' t solve a problem until you know what the problem is (give precise and thorough specs).

Learn to read a program at different levels of abstraction.

Use methods and method calls so that you don't have nested loops

Simplicity and beauty: keys to success

CS has its field of
computational complexity.
Mine is computational
simplicity,

David Gries

Inside every large program is a
little program just trying to
come out. **Tony Hoare**

Beauty is our Business.
Edsger Dijkstra

CS professor's non-dilemma

I do so want students to see
beauty and simplicity.
A language used just has to be
one only with that property.
Therefore, and most reasonably,
I will not and do not teach C.

David Gries

Admonition
a little Grook

In correctness concerns
one must be immersed.
To use only testing
is simply accursed.

Correctness of programs, the teaching of programming

simplicity
elegance
perfection
intellectual honesty

Edsger W. Dijkstra

Sir Tony Hoare

Dijkstra: The competent programmer is fully aware of the limited size of his own skull, so he approaches the programming task in full humility, and among other things, he avoids clever tricks like the plague.

Hoare: Two ways to write a program:
(1) Make it so simple that there are obviously no errors.
(2) Make it so complicated that there are no obvious errors.

Axiomatic Basis for Computer Programming. Tony Hoare, 1969

Provide a definition of programming language statements not in terms of how they are executed but in terms of proving them correct.

{precondition P}
Statement S
{Postcondition Q)

Meaning: If P is true, then execution of S is guaranteed to terminate and with Q true

Assignment statement **x= e;**

{true}	{x+1 >= 0}	{2*x = 82}
x= 5;	x= x + 1;	x= 2*x;
{x = 5}	{x >= 0}	{x = 82}

Definition of notation:
P[x:= e] (read P with x replaced by e) stands for a copy of expression P in which each occurrence of x is replaced by e

Example: (x >= 0)[x:= x+1] = x+1 >= 0

Definition of the assignment statement:
{P[x:= e]}
x= e;
{P}

Assignment statement x= e;

Definition of the assignment statement:
{P[x:= e]}
x= e;
{P}

```
{x+1 >= 0}          {2*x = 82}
x=   x + 1;         x=   2*x;
{x >= 0}            {x = 82}
```

```
{2.0xy + z = (2.0xy + z)/6}          x = x/6
x=   2.0*x*y + z;
{x = x/6}                            2.0xy + z = (2.0xy + z)/6
```

If statement defined as an “inference rule”:

Definition of if statement: If

{P && B} ST {Q} and
{P && !B} SF {Q}

Then

```
{P}
if (B) ST
else   SF
{Q}
```

The then-part, ST, must end with Q true
The else-part, SF, must end with Q true

Hoare's contribution 1969:

Axiomatic basis: Definition of a language in terms of how to prove a program correct.

But it is difficult to prove a program correct after the fact.

How do we develop a program and its proof hand-in-hand?

Dijkstra showed us how to do that in 1975.

His definition, called "weakest preconditions" is defined in such a way that it allows us to "calculate" a program and its proof of correctness hand-in-hand, with the proof idea leading the way.

Dijkstra: *A Discipline of Programming*. Prentice Hall, 1976.
A research monograph

Gries: The Science of Programming. Springer Verlag, 1981.
Undergraduate text.

How to prove concurrent programs correct.
Use the principle of non-interference

```
Thread T1
{P0}
S1;
{P1}
S2;
{P2}
…
Sn;
{Pn}
```

```
Thread T2
{Q0}
Z1;
{Q1}
Z2;
{Q2}
…
Zm;
{Qm}
```

We have a proof that T1 works in isolation and a proof that T2 works in isolation.
But what happens when T1 and T2 execute simultaneously, operating on the same variables?

How to prove concurrent programs correct.

```
Thread T1
{P0}
S1;
{P1}
S2;
{P2}
…
Sn;
{Pn}
```

```
Thread T2
{Q0}
Z1;
{Q1}
Z2;
{Q2}
…
Zm;
{Qm}
```

Turn what previously seemed to be an exponential problem, looking at all executions, into a problem of size n*m.

Prove that execution of T1 does not interfere with the proof of T2, and vice versa.
Basic notion: Execution of Si does not falsify an assertion in T2: e.g. {Pi && Q1} S2 {Z2}

```
Thread T1
{P0}
S1;
{P1}
S2;
{P2}
…
Sn;
{Pn}
```

```
Thread T2
{Q0}
Z1;
{Q1}
Z2;
{Q2}
…
Zm;
{Qm}
```

Interference freedom. Susan Owicki's Cornell thesis, under Gries, in 1975.

A lot of progress since then! But still, there are a lot of hard issues to solve in proving concurrent programs correct in a practical manner.

Prove that execution of T1 does not interfere with the proof of T2, and vice versa.
Basic notion: Execution of Si does not falsify an assertion in T2: e.g. {Pi && Q1} S2 {Z2}