Functional programming

- gabrielmark.kuper@unitn.it
- Reception: Contact by email
- Books
  - Maurizio Gabbrielli and Simone Martini, "Linguaggi di Programmazione – Principi e Paradigmi", McGraw-Hill
  - Jeffrey D. Ullman, "Elements of ML Programming", ML97 edition. Prentice-Hall
  - Other material on esse3

Other books

- Louden, Kenneth C. and Kenneth A. Lambert, "Programming languages: Principles and practice", Cengage, 2012
- Scott, Michael L., "Programming language pragmatics", Morgan Kaufmann, 2019
- Paulson, L.C., "ML for the working programmer", Cambridge, 1992
- MacLennan, Bruce. "Functional programming: Practice and theory", Addison-Wesley, 1990

Objectives

- We asssume knowledge of an imperative language, such as C
- There are many similar languages, with different syntax
- Other languages are based on different principles
- We study characteristics of various programming language paradigms
  - Imperative
  - Object-oriented
  - *Functional*
- We start with a brief history of the development of programming languages

Course organization

- 2 lessons (4 hours) each week
  - Theoretical (usually Monday): Principles of programming languages
    - Execution of a program (interpreted or compiled)
    - Memory management
    - Control management
    - Abstraction of data
    - Different programming language paradigms
    - Theory of functional programming
  - Functional programming (usually Thursday)
    - Practical: The functional language ML
    - Theory: The lambda-calculus

ML installation

- Poly/ML: `http://polyml.org/index.html`
- Try to download and install by Thursday
- Requires compilation, but pre-compiled versions should be available online
- Should be installed on PCs in lab
  - For remote exams, you must have poly installed on your PC
  - For exams "in presenza" you should try the version in the lab, and make sure you can create a file and save it

History of programming languages

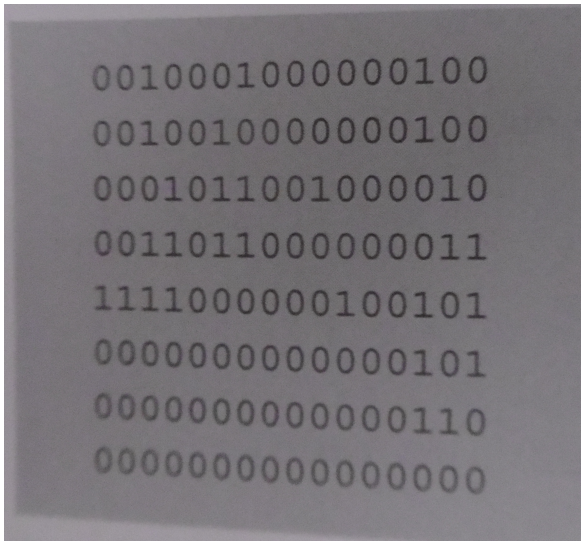
- Hilbert and Ackermann (1928): Entscheidungsproblem

  > Find an algorithm that considers, as input, a statement and answers "Yes" or "No" according to whether the statement is valid or not

- Negative answer:
  - Church (1932). Lambda calculus
    - $\lambda x.x^2$: the function that takes x to x^2
    - Computation is defined as applying functions to functions
  - Turing (1936): Computation defined via *Turing Machines*
- Church's lambda calculus is the foundation for functional programming

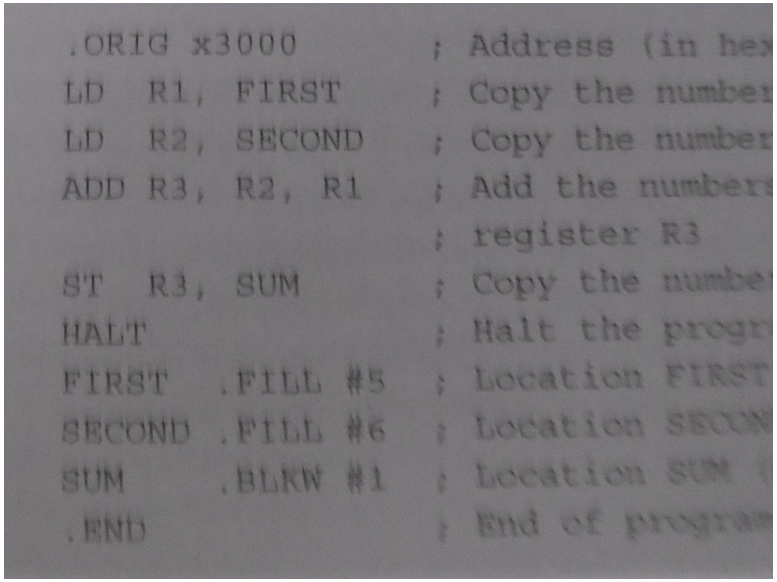
Early computers

- Von Neumann: Stored-program computer
- Programming in machine code



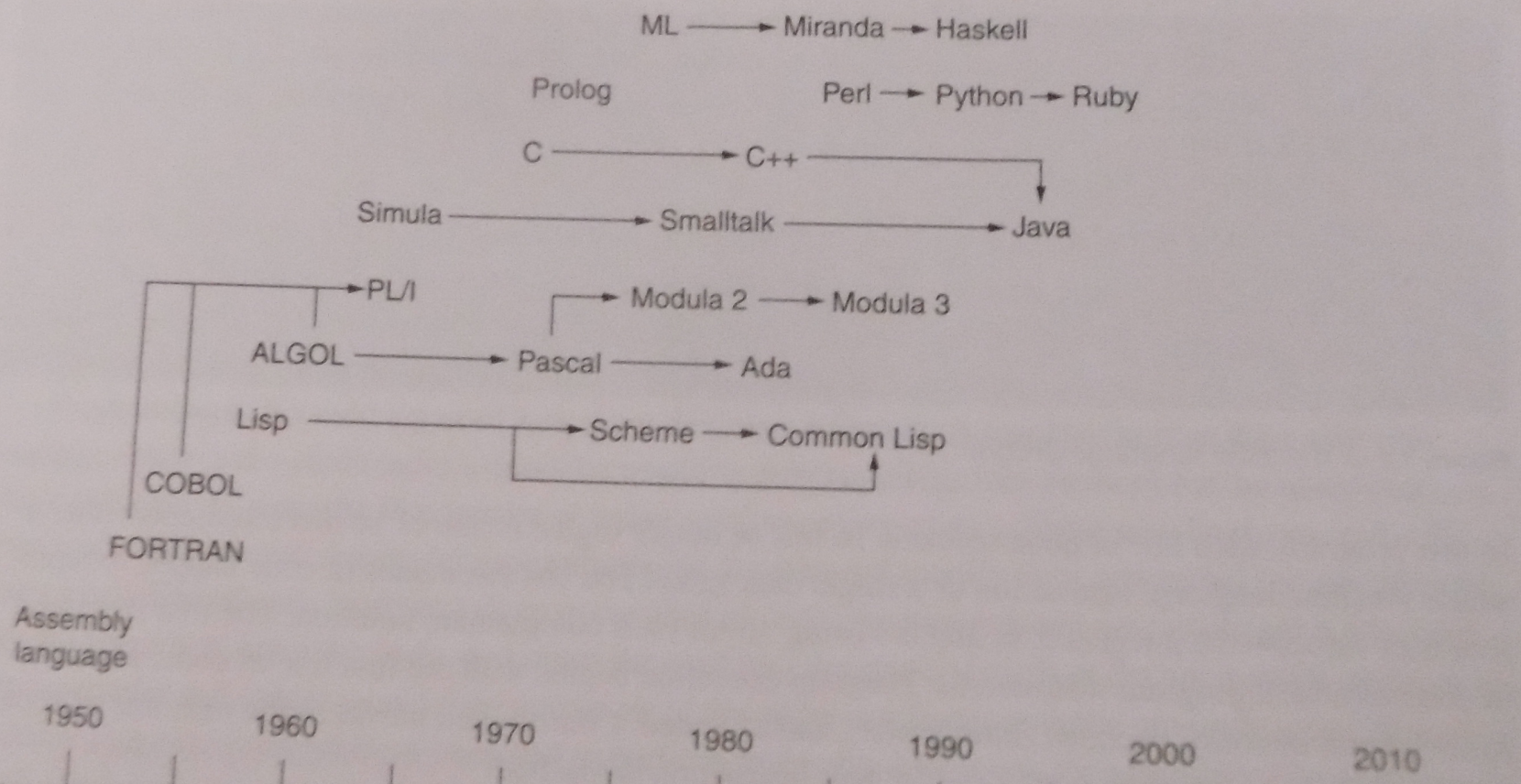
Assembly language

- More readable names for machine instructions



```
.ORIG x3000          ; Address (in hex
LD   R1, FIRST       ; Copy the number
LD   R2, SECOND      ; Copy the number
ADD R3, R2, R1       ; Add the numbers
                     ; register R3
ST   R3, SUM         ; Copy the numbe
HALT                 ; Halt the progr
FIRST  .FILL #5      ; Location FIRST
SECOND .FILL #6      ; Location SECON
SUM    .BLKW #1      ; Location SUM
.END                 ; End of progra
```

Timeline

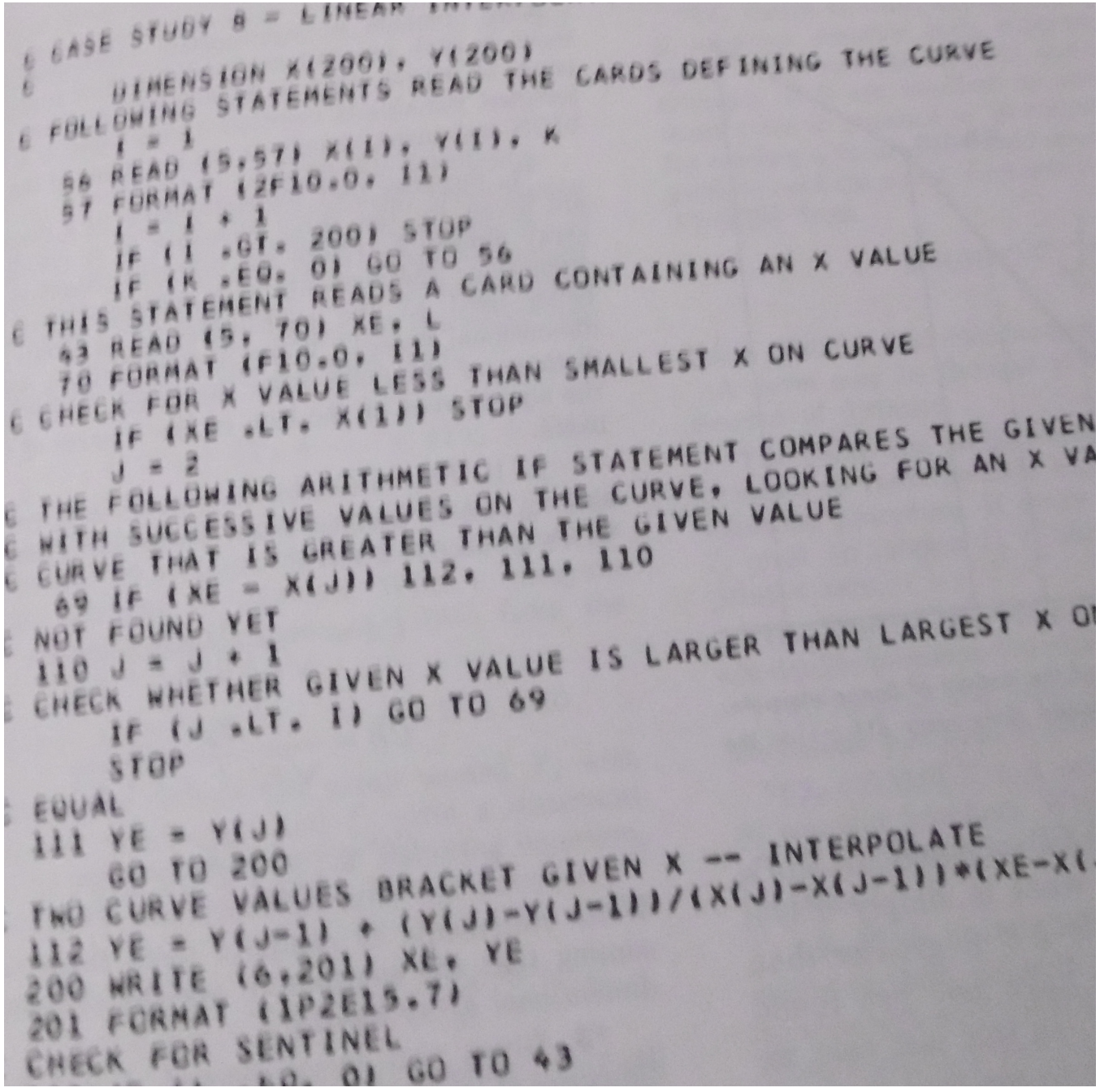


gure 1.1 A programming language timeline

FORTRAN

- FORmula TRANslation language
- Limited structured control sequence
- Limited data support
- Very good for floating point and algebraic notation
- Introduces variables and arrays

Example



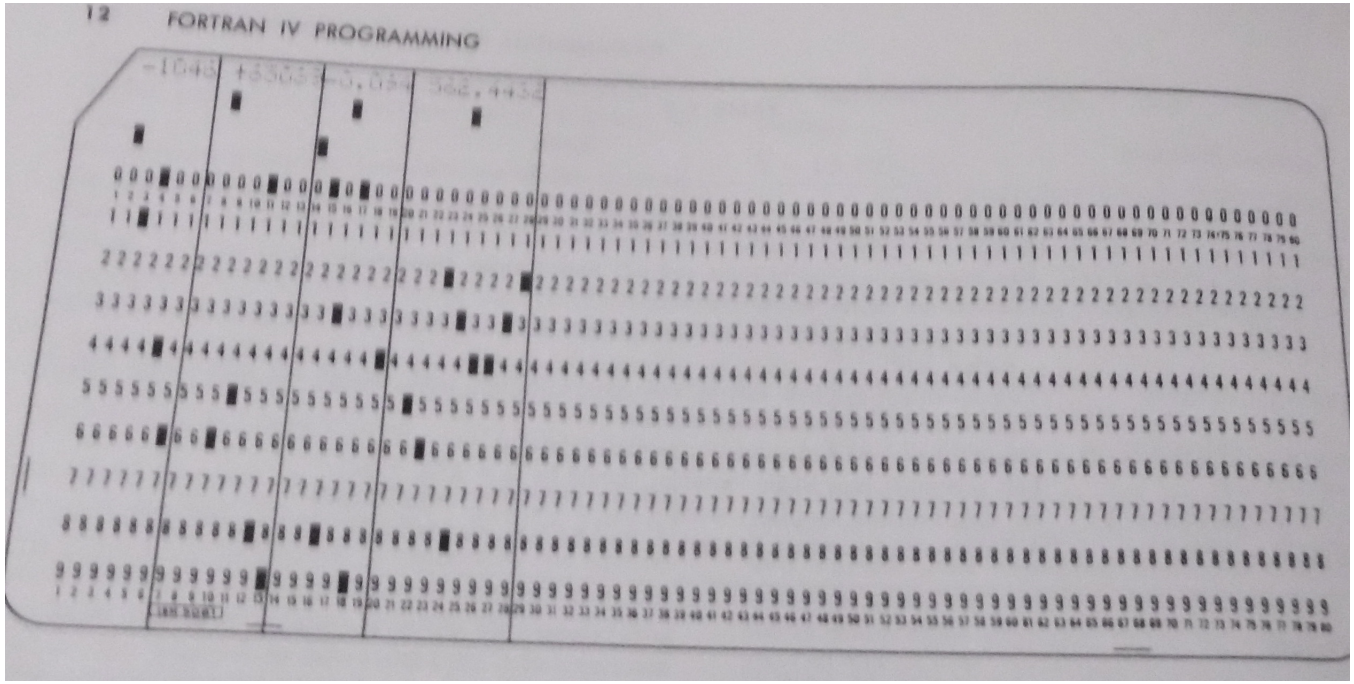
```
C CASE STUDY 8 = LINEAR
C     DIMENSION X(200), Y(200)
C FOLLOWING STATEMENTS READ THE CARDS DEFINING THE CURVE
      I = 1
   56 READ (5,57) X(I), Y(I), K
   57 FORMAT (2F10.0, I1)
      I = I + 1
      IF (I .GT. 200) STOP
      IF (K .EQ. 0) GO TO 56
C THIS STATEMENT READS A CARD CONTAINING AN X VALUE
   43 READ (5, 70) XE, L
   70 FORMAT (F10.0, I1)
C CHECK FOR X VALUE LESS THAN SMALLEST X ON CURVE
      IF (XE .LT. X(1)) STOP
      J = 2
C THE FOLLOWING ARITHMETIC IF STATEMENT COMPARES THE GIVEN
C WITH SUCCESSIVE VALUES ON THE CURVE, LOOKING FOR AN X VA
C CURVE THAT IS GREATER THAN THE GIVEN VALUE
   69 IF (XE - X(J)) 112, 111, 110
C NOT FOUND YET
  110 J = J + 1
C CHECK WHETHER GIVEN X VALUE IS LARGER THAN LARGEST X O
      IF (J .LT. I) GO TO 69
      STOP
C EQUAL
  111 YE = Y(J)
      GO TO 200
C TWO CURVE VALUES BRACKET GIVEN X -- INTERPOLATE
  112 YE = Y(J-1) + (Y(J)-Y(J-1))/(X(J)-X(J-1))*(XE-X(
  200 WRITE (6,201) XE, YE
  201 FORMAT (1P2E15.7)
C CHECK FOR SENTINEL
      IF (L .EQ. 0) GO TO 43
```

FORTRAN

- Arrays have fixed size (compile-time)
- Procedures, but no recursion.
  - Both facilitate data management
- Procedures can be independently compiled
- FORMAT for I/O

Input

- Programs usually run in batch mode
- Input: Stack of punched cards



ALGOL

- Not designed for a particular application domain
- Three main versions
  - 1958
  - 1960
  - 1968
- Directly influenced most imperative languages
- Algol 60 report:

  Backus et. al. "Report on the algorithmic language ALGOL 60", CACM 1960
- Introduced BNF syntax definitions
- A lot of subsequent work to improve this document

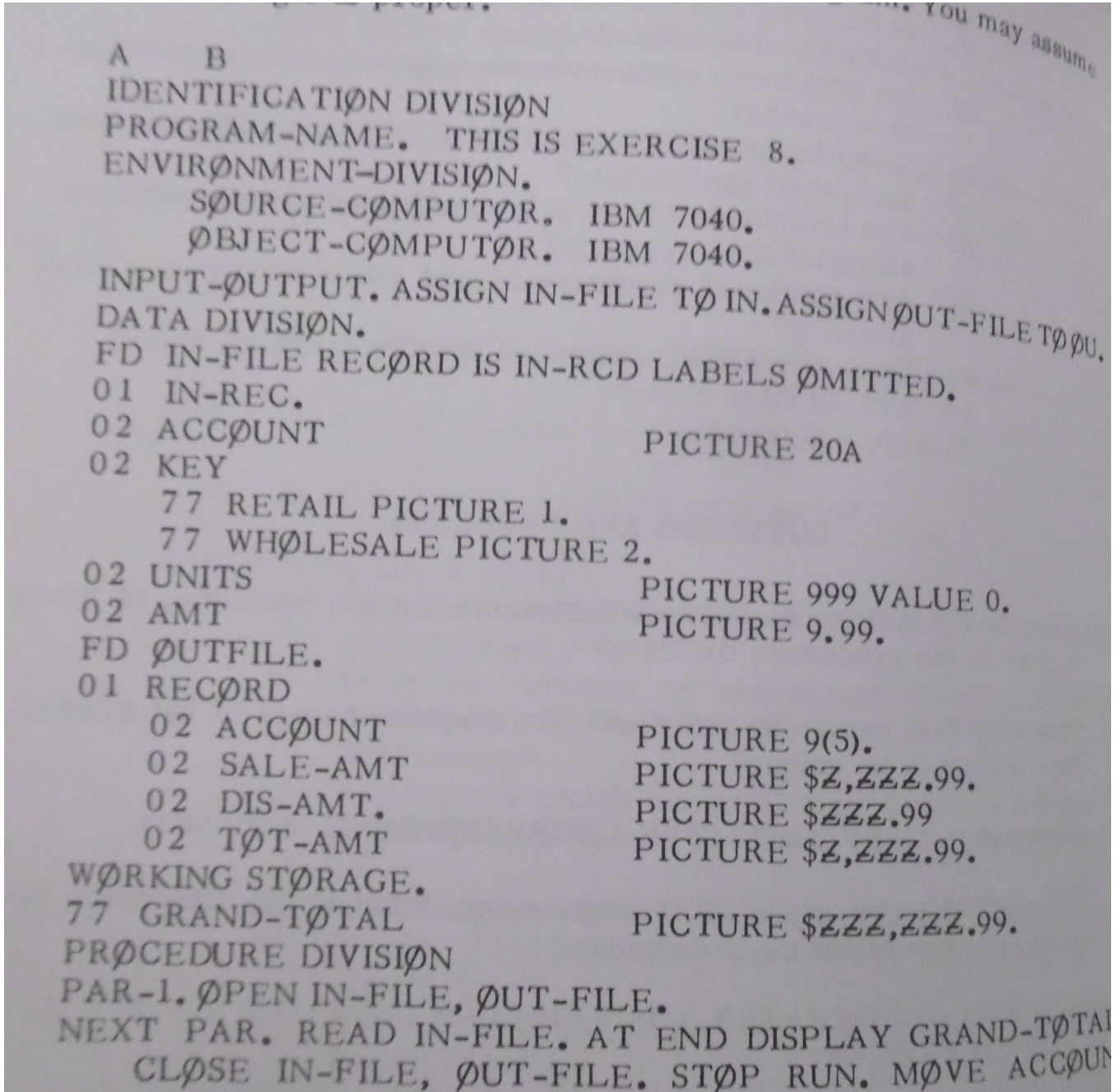
ALGOL

- Block structure, with scope for local variables
- Explicit type declarations
- Dynamic vs. static lifetime for variables
- If-then-else
- Call by value and call by name
- Recursion
- Arrays with dynamic bounds

ALGOL

- Key point: Designed around a model of implementation using a stack
- Early implementation in 1960

COBOL



```
                                                    You may assume
A    B
IDENTIFICATIØN DIVISIØN
PROGRAM-NAME.  THIS IS EXERCISE 8.
ENVIRØNMENT-DIVISIØN.
     SØURCE-CØMPUTØR.  IBM 7040.
     ØBJECT-CØMPUTØR.  IBM 7040.
INPUT-ØUTPUT. ASSIGN IN-FILE TØ IN. ASSIGN ØUT-FILE TØ ØU.
DATA DIVISIØN.
FD  IN-FILE RECØRD IS IN-RCD LABELS ØMITTED.
01  IN-REC.
02  ACCØUNT                          PICTURE 20A
02  KEY
    77  RETAIL PICTURE 1.
    77  WHØLESALE PICTURE 2.
02  UNITS                            PICTURE 999 VALUE 0.
02  AMT                              PICTURE 9.99.
FD  ØUTFILE.
01  RECØRD
    02  ACCØUNT                      PICTURE 9(5).
    02  SALE-AMT                     PICTURE $Z,ZZZ.99.
    02  DIS-AMT.                     PICTURE $ZZZ.99
    02  TØT-AMT                      PICTURE $Z,ZZZ.99.
WØRKING STØRAGE.
77  GRAND-TØTAL                      PICTURE $ZZZ,ZZZ.99.
PRØCEDURE DIVISIØN
PAR-1. ØPEN IN-FILE, ØUT-FILE.
NEXT PAR. READ IN-FILE. AT END DISPLAY GRAND-TØTAL
    CLØSE IN-FILE, ØUT-FILE. STØP RUN. MØVE ACCØUN
```

Features

Block structure `BEGIN,END`

- Introduces the `RECORD` type (in C, `struct`)
- Very verbose; intent was to program in natural language
- File description and manipulation
- Focus on data: Most databases (until the relational model) are in COBOL

PL/I

- Synthesis of previous languages (very large definition)
- New:
  - Exceptions
  - Pointers and lists
  - Separately compiled procedures
  - Multitasking

Simula 67

- Introduces the notion of “classes”

LISP

- Designed for AI
- McCarthy: Based on the lamda-calculus
- Not a pure functional system, but a step in this direction
- Key feature: List processing
- Example

```
(defun foo (a b c d) (+ a b c d))
```

- Relies on garbage collection

APL

- A Programming Language
- Prime numbers from 1 to R

```
(~R∈R∘.×R)/R←1↓⍳R
```

- Limited control (just goto)
- Very powerful mathematical operations
- No block structure or explicit types

Smalltalk

- From XEROX PARC
- Fully object-oriented language (even constants are objects)
- Graphic programming language

PASCAL

- An extension of ALGOL (originally ALGOL W)
- Recursive data types
- Strong typing
- Very successful for education

ADA

- In the ALGOL tradition
- Designed for embedded and real-time systems
- Very strong typing
- Modular programming support
- Pragmas: compiler directive that conveys information to the compiler (ignore array bound checking)

C

- In the Algol tradition
- Designed for system programming
- Has constructs that map efficiently to machine instructions
- Static type system

C++

- Originally preprocesser for C
- Adds classes
- Later fully object-oriented

Logic programming (Prolog)

- Write programs in logic
- Inference done automatically

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y)       :- parent_child(Z, X),
parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

```
?- sibling(sally, erica).
```

Functional programming

- Subject of this course
- John Backus "Can Programming Be Liberated From the von Neumann Style? A Functional Style and its Algebra of Programs" (1977)
- Some aspects of functional programming in LISP and other languages
- Other languages: Scheme, OCaml, Haskell, Miranda, ML
- Miranda and Haskell are purely functional. Most others allow some side-effects.

Java

- Object-oriented language, designed to be fully portable
- Programs compiled into "bytecode" that can be run on any Java Virtual Machine
- Early implementations interpreted bytecode
- Better performance with "just-in-time" compilers
- Originally for embedded systems, they are now used widely on the Internet

Scripting languages

- Started wih shell languages
  - JCL, MS-DOS
  - Unix: `csh`, `sed`, `awk`
  - Followed by: `perl`, `python` etc.
  - characteristics
    - Compiled and interpreted
    - Economy of expression
    - No declarations, simple scoping rules
    - Flexible dynamic typing
    - Easy access to system facilities