

# Evolution of PL's

## I. Ways of Organizing the evolution of PL's

### A. Types of PL's [Sebesta p. 37]

1. Procedural (Imperative) [F&G, p. 39]
2. Functional (applicative) [F&G, p. 41]
3. Logical (declarative)
4. OOP
5. Special Purpose

### B. Generations of PL's

1. 1st Generation: e.g. FORTRAN, COBOL, BASIC
  - a. fixed format, machine-oriented control structures & data types
  - b. static storage allocation: done at compile time, optimizing run time but not storage usage or ease of writing programs
2. 2nd Generation: e.g. ALGOL 60
  - a. machine independent free-format syntax
  - b. higher level control structures & data types
  - c. hierarchically structured programs & name spaces
  - d. dynamic allocation of variables; recursion
3. 3rd Generation: e.g. ALGOL 68, PASCAL, C
  - a. focus on simplicity & extensibility
  - b. programmer-defined data types
4. 4th Generation (4GL): 2 views [MacLennan] vs. [F&G]
  - a. Some view this as application generator programs: e.g. Lotus 123, HyperCard
    - (1). powerful control structures or statements that allowed invoking in a few words some action that would otherwise take many lines of code
    - (2). viewed as *replacements* for PL's, not as new PL's
  - b. Alternate view: Powerful Data Abstraction Language: e.g. ADA
    - (1). encapsulation modules
    - (2). concurrent programming
5. 5th Generation: 2 views, as above
  - a. Some see it as a more natural-language-like interface, e.g. for database queries
  - b. Could be Functional, Logical, or OOP

## C. Programming Domains

1. Scientific Applications: FORTRAN & ALGOL 60 prdominated
  - a. Early Computers: Missile trajectories, log tables
  - b. Floating point processing important
2. Business Applications
  - a. Herman Hollerith's punch card census tabulation (1890?) -> IBM
  - b. Historical Sidebar: IBM debated whether to provided mixed-case I/O or faster processing
  - c. 60's: COBOL: report generating, file processing
  - d. 80's: micros: spreadsheets, databases, word processors
  - e. 90's: networking & connectivity: modems, WWW, email
3. Artificial Intelligence
  - a. Using computers to do things humans are currently better at doing
  - b. Inexact procedures, symbolic computation
  - c. '65: LISP, 70's: PROLOG
4. Systems programming
  - a. Higher level tools used so assembler not needed
  - b. Used to reference external devices, low-level code
  - c. E.g IBM: PL/S, DEC: BLISS; Burroughs: Extended ALGOL
  - d. UNIX & C
5. Scripting Languages (Very-High-Level-Languages)
  - a. shell -> .... ksh
  - b. awk (was a report generating language) & shell made pearl
  - c. tcl/tk scripting w/Xwindows
  - d. Some database 5GL's: allow some degree of natural language interface
6. Special Purpose Languages
  - a. RPG
  - b. Industrial control (MULTICS)

## II. Evolution according to Types of PL's

### A. Procedural (Imperative) [F&G, p. 39]

#### 1. Konrad Zuse's Plankalkül

- a. 1945 in postwar Germany: Z4 computer
  - b. Written in 1945, not published 'till 1972
  - c. data structures: "bit" type, 2's complement, arrays, records, recursion, for, selection stmt. but no "else",
  - d. Notation: up to 3 lines: 1. statement, 2. subscripts of arrays from line 1, 3. types from line 1
- e.g.  $A[5] = A[4] + 1$ , where 1.n is an integer of  $n$  bits

```
| A + 1 => A
v | 4      5
s | 1.n    1.n
```

#### 2. Early Languages (1st Generation)

##### a. FORTRAN

###### (1). Background:

- (a). Early 50's: pseudocodes for floating point masked cost of interpreters
- (b). 1954: IBM's 704 w/floating point hardware
- (c). Not intended to be portable, but IBM 709 announced before compiler was ready

###### (2). FORTRAN I (1957)

- (a). first compiled language. Uppercase characters only, no "<", algebraic notation w/operator precedence, had IF (ELSE), had DO (no WHILE), used GOTO, numeric types only, 6 char. identifiers (only 2 previously), identifiers beginning w/ I-N: integers (subscripts), the rest were real
- (b). Max 3-D arrays, all variables statically allocated
- (c). Control statements based on 704 machine instructions
- (d). e.g. 3-way arithmetic IF corresponded to CAS (compare AC w/ Storage)

```
IF (J-1) 21, 76, 76
```

Equivalent to modern IF:

```
IF J-1 .LT. 0 THEN
    <block1>
ELSE
    <block2>
ENDIF
```

- (3). FORTRAN II ('58): independent compilation. Previously machine would fail before getting through compiling progs. > 3-400 lines
- (4). FORTRAN IV (1962): type declarations, logical variables, logical IF
- (5). FORTRAN 77: character strings, IF-ELSE
- (6). FORTRAN 90: dynamic array allocation, built-in array fcn's., records, pointers, modules, CASE, loop EXIT & CYCLE, recursion, certain features (COMMON, GOTO) marked for future deletion
- (7). [Example of FORTRAN prog., MacLennan p. 36] finding absolute mean

## b. COBOL

Problem: everyone was developing their own business language (RCA, Sylvania)

- (1). DOD sponsored mtg. in '59, design goals: as much English as possible, easy to learn & use, mechanically translatable.
- (2). "Short Range Committee": Initial specs. in '60. Program divided into 1. Data description & 2. executable operations
- (3). no control structures, no parameters in subprograms
- (4). COBOL II in '85: control structures, parms., no user-defined types
- (5). [COBOL prog., Pratt p. 273] Find total # items and sum of prices

## c. BASIC

- (1). Dartmouth '64: remote access, first time-sharing system, user time more important than computer time
- (2). 2 char. identifiers, line numbers, GOTO
- (3). [Example Prog., Pratt p. 71] Compute sum of  $1^2+2^2+\dots+10^2$
- (4). Language has been denigrated, although DEC PDP-11 RSTS written in a version of BASIC
- (5). Visual Basic

## 3. Block Structured Languages

FORTRAN could have been universal lang., but owned by IBM

### a. ALGOL

- Designed by an international committee, designed to be platform independent  
- Goals: close to mathematical notation, suitable for algorithm description in publications, mechanically translatable

- (1). ALGOL 58: 8 people 8 days in Zurich
  - (a). Introduced free-format syntax, unlimited length identifiers, if-then-else, for (step, while), switch w/goto, hierarchical nesting of control structures, compound statements, data types, arrays w/out bounds & any number of indices, := for assignment
  - (b). Not supported by IBM
- (2). ALGOL 60: 13 people for 6 days in Paris
  - (a). described using BNF in 15 pages
  - (b). introduced nested scopes (block structure), recursion, pass by value (& *name*), string type, no standard I/O
  - (c). [Example prog. MacLennan p. 154] finding absolute mean
- (3). ALGOL 68: complete orthogonality, programmer-specified data types - overly complex, used an unknown metalanguage
- (4). Burroughs B5000 ... built to accommodate ALGOL. Had stack to implement block structure & recursion

b. PL/1

- IBM's answer to combine business (using regression) and scientific groups (managing files): combine ALGOL, FORTRAN, & COBOL

- (1). 3x3 committee: 5 months, meeting every other week
- (2). Introduced concurrent tasks (but no synchronization method), exception handling, pointers, operations on array cross-sections
- (3). Too large & complicated: Dijkstra in '72 Turing Award Lecture:  
"I absolutely fail to see how we can keep our growing programs firmly within our intellectual grip when by its sheer baroque-ness the programming language - our basic tool, mind you! - already escapes our intellectual control."
- (4). [Example prog., Sebesta p. 71] Find # values > mean

c. Pascal

- Niklaus Wirth: part of ALGOL committee, disagreed w/ALGOL 68 as being too unmanageable.

- (1). Designed to be simple and extensible
- (2). Most of ALGOL features, subprograms (but not blocks), pass by value & reference, char. & bool types, programmer-defined types (arrays, records, pointers)
- (3). No separate compilation, variable-size array arguments, or dynamic arrays
- (4). [Example prog., MacLennan p. 180] finding absolute mean
- (5). Modula-2: successor w/modules, short-circuit evaluation, premature exit from a loop

d. C

- Based on BCPL, intended for portable system programming, done originally via teletype.

- (1). Low level, built-in types may be treated as bit-strings, pointer arithmetic, poor type-checking
- (2). Avoids keywords and uses various special characters, making it algebraic-looking
- (3). macro pre-processor for text substitution before compilation, supports modules which define namespaces
- (4). K&R: '78, ANSI '89

B. Functional (applicative) [F&G, p. 41]

1. Background

- a. Automated theorem proving desired as well as symbolic data processing using lists
- b. McCarthy: summer job @ IBM in '58 working on a symbolic processing lang., then at MIT w/Marvin Minsky. Wanted automatic garbage collection

2. LISP

- a. Has only atoms & lists. e.g. (A (B C) D (E (F G)))
- b. Internal representation (see Sebesta p. 50)
- c. [E.g. Sebesta p. 51] Defining a function to compare 2 lists
- d. Descendants of Lisp:
  - (1). Scheme: static scoping, functions are 1st class
  - (2). COMMON LISP: also allows dynamic scoping, data structures: records, strings

3. ML: Functional Lang., but also supports imperative prog. paradigm, type-checking @ compile time

- 4. Miranda: Lazy evaluation
- C. Logical (declarative)
  - Declaring *what* is true rather than *how* to do something. Most notable example: PROLOG
  - 1. Colmerauer & Roussel at U. of Marseilles in early 70's
  - 2. Clauses expressed in Horn clause form: the "then" is on the left, the "if" conditions are on the right
  - 3. Deduction follows in order [F&G p. 351 ff.]
- D. OOP
  - 1. Simula
    - a. Simulations: Modeling real-world objects, requires restarting subprograms where they left off. These subprogs. called *coroutines*.
    - b. Coroutines implemented as *objects* of a *class*
    - c. Introduced *inheritance* (class, superclass)
  - 2. Ada
    - a. Commissioned by DoD ('74: 450 languages in use for DoD projects), particularly for reusable code & mission critical applications
    - b. Designed in stages, competitive process
    - c. Based on Pascal. Introduces loop w/exit; in, out, and in-out parameters; default parameters; procedure, function & operator overloading; concurrent execution of tasks using rendezvous mechanism; expands on Pascal's numeric type constructors. E.g. [Sebesta, p. 89] Prog. to find # greater than mean
    - d. *packages* used to define modules or ADT's; Can write generic packages, which allows the type used in the package to be a parameter
    - e. Problem: extremely large: compilers were slow in coming. Hoare in '85 stated that Ada should not be used for any application where reliability is critical [due to its unwieldy size]
    - f. Ada 95
  - 3. Smalltalk
    - a. Alan Kay at Xerox PARC in '71: Dynabook. Desktop model, pointing paradigm of interaction, uniform interface, programming browser
    - b. Smalltalk-80 based on: data abstraction & inheritance of Simula, the dynamic typing & functional semantics of LISP, and the graphical interactive environment of LOGO
    - c. Introduced the "object" paradigm of message passing
    - d. Everything is an object. E.g. The number one is an object that can respond to the message: "What is the result of adding one to yourself?"
  - 4. C++ & hybrid OOP's
    - a. OOP extensions have been added to Pascal, C, LISP, Modula 2, Ada, etc.
    - b. Added to these languages: information hiding (abstraction), encapsulation, inheritance, & dynamic binding (polymorphism)
    - c. C++: Bjarne Stroustrup et. al. at AT&T during 80's. Was initially a front end to translate into C (Cfront)
    - d. Features: argument types in function prototypes, default function args., function overloading, *inline* specifier, named scopes, exception handling
    - e. OOP features: public, private, & protected access specifiers for class members; allows execution-time *method* resolution. Programmer-controlled initialization & destruction of objects. Generic functions & classes, multiple inheritance.
- E. Special Purpose