Chapter 2 - Evolution of Major PLs

We emphasize

- the contribution of each language
  - focus on new features that most influenced subsequent languages
- and the motivation for its development

See Figure 2.1

Chapter contains listings of complete example programs, each in a different language.

1. Plankalkül (1945)

- **Never implemented**
- **Primitive (scalar) data types**
  - **single bit, integer and 2's complement floating-point type**
- **Composite (structured) data types**
  - **arrays, records (and records of records)**
- **Iterative (for), out of loop jump and selection (if) statements**
- **Each statement consisted of 2 or 3 lines of code:**

A(7) := 5 * B(6) ← Optional line

```
    | 5 * B => A
V   |     6    7      (subscripts)
S   |    1.n  1.n     (data types)
```


2. Minimal Hardware Programming: Pseudocodes (1949)

Early computers of 1940s & 1950s:

- slow, unreliable, and expensive
- having extremely small memories
- difficult to program due to lack of supporting sw.

Programming was done in **machine code**, which is tedious and error-prone:

- poor readability: numeric codes are used to specify instrs, e.g., 14 for ADD operation
- poor modifiability: absolute addressing (*study* the *example*)
- Expression coding was tedious
- Machine deficiencies--no indexing or fl. pt.

2.1 Short Code (1949)

- used to program UNIVAC I computer:
- 72 bit words, grouped as 12 six-bit bytes.
- Expressions were coded, left to right
- Variables, or memory locations, were named with byte-pair codes:

```
   X0  =  SQRT(ABS(Y0))
???
00 X0  03   20   06  Y0
```

- was implemented with a pure interpreter (simpler but 50 times slower than machine code)

2.3 Speedcoding (1954)

- **Developed for the IBM 701**
- **Interpreter converted the 701 to a virtual three-address f-p calculator**
  - pseudoinstructions for + - * / sqrt, since, atan, exp, log
- **Conditional and unconditional branching**
- **Input/output conversions**
- **Autoincrement registers for array access(allowed matrix multiplication in 12 instructions)**
- **But**
  - **Slow!**
  - **Only 700 words left for user program**

2.3 The UNIVAC "Compiling" System (1954)

- Compiling system expanded a pseudocode into machine code in the same way as macros are expanded into assembly language.
  - Primitive but much shorter source codes

2.4 Related Work

- Cambridge Un. - using blocks of relocatable addresses to solve the problem of absolute addresses
  - This idea was extended to design an **assembly language** (*no impact of AIs on HLLs*) that could combine chosen subroutines and allocate storage.

FORTRAN: The IBM **FOR**mula **TRAN**slating System (1957)

FORTRAN 0 & IBM 704 Machine

- first computer with index registers and f-p hw
  - So far all f-p ops had to be simulated in software which made interpretation an acceptable alternative to machine code execution (interp. cost used to be hidden by the high cost of f-p calcs)
- FORTRAN is credited with being the first compiled HLL. [*no consensus though*!]
  - The Laning and Zierler system (1954) of MIT was the first algebraic translation system: it translated arith. expressions, used function calls for math. functions, and included subscripted var. references.

FORTRAN I (1956)

- **Input/output formatting**
- **var. names of up to six characters (from 2 in v.0)**
- **user-defined subroutines(no separate compilation)**
- **IF selection statement,post-test counting DO loops**
- **Arithmetic IF (3-way branch) instead of logical IF!!**
- **No data typing (I to N: integer, otherwise: f-p)**
- **Compiler released in April 1957, after 18 worker/ years of effort**
- **Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of the 704**
- **Code was very fast**
- **Quickly became widely used**

Environment in which FORTRAN was developed:

- **Computers were small and unreliable**
- **Applications were scientific**
- **No programming methodology or tools**
- **Machine efficiency was most important**

FORTRAN II (1958)

- **Independent compilation of subroutines + linkers**
- **Fix the bugs**

FORTRAN IV (1960-62)

- **Explicit type declarations**
- **Logical selection statement**
- **Subprogram names could be parameters**
- **ANSI standard in 1966**

FORTRAN 77 (1978)

- **Character string handling**
- **Logical loop control statement**
- **IF-THEN-ELSE statement**

FORTRAN 90 (1990)

- **Modules-** PRIVATE **or** PUBLIC **data and subprograms**
- **Built-in array operations: SUM, MATMUL, etc.**
- **Dynamic arrays:** ALLOCATABLE
- **Pointers**
- **Recursion - recursive procedures**
- **CASE multiple selection statement**
- **Parameter type checking**

ALGOL - ALGOrithmic Language

ALGOL 58 (1958)

- Environment of development:
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being developed, all for specific machines
  - No portable language; all were machine-dependent
  - No universal language for communicating algorithms
    - FORTRAN could not become a universal language because at the time it was solely owned by IBM.

Goals of the Language

- American ACM and European GAMM determined the goals: *[after endless discussions]*
  - Close to mathematical notation (good for sci.prog.)
  - Good for describing algorithms
  - Must be translatable to machine code

ALGOL 58 Features

- **Concept of type was formalized**
  - **vars. not f-p numbers required explicit declaration**
- Names could have any length
- **Arrays could have any number of subscripts**
- Parameters were separated by mode (in & out)
- **Subscripts were placed in brackets**
- Compound statements (begin ... end)
- **Semicolon as a statement separator**
- Assignment operator was :=
- if **had an** else-if **clause**
- a for statement for counting loops
- **a** do **statement for subprogram calls.**

ALGOL 60 (1960)

- New Features:
  - Block structure (local scope)
  - Two parameter passing methods: by value/by name
  - Subprogram recursion
  - Stack-dynamic arrays
  - Still no i/o and no string handling -subscript range is specified by vars during execution.
- Successes:
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative langs are based on it
  - First machine-independent language
  - First lang. whose syntax was formally defined (BNF)

- Failure:
  - Never widely used, especially in U.S
  - **Reasons:**
  - No i/o and the character set made programs non-portable
  - Too flexible--hard and inefficient to implement
  - Entrenchment of FORTRAN
  - Formal syntax description
  - Lack of support of IBM

ALGOL 68 (1968)

- **follows ALGOL 60, but it is not a superset of it.**
- **Design is based on the concept of orthogonality:** a few primitive types and structures and allow the user to combine those primitives into a large number of different structures.
- ***Contributions:***
  - User-defined data types (structures)
  - Reference types
  - Dynamic arrays (called `flex` arrays)
- ***Comments:***
  - Had even less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, C++, PL/I and Ada.

Pascal (1971)

- **Designed by Wirth, who quit the ALGOL 68 committee (didn't like the direction of that work)**
- **Designed for teaching structured programming**
- **Small, simple, nothing really new**
- **Still the most widely used language for teaching programming in colleges (but use is shrinking)**

Weaknesses:

- **A subprogram can not take an array of variable length as argument**
- **Lack of any separate compilation capability**

C (1972)

- **Designed for systems programming (at Bell Labs by Dennis Richie)**
- **Evolved primarily from B, but also ALGOL 68**
  - for and switch statements
  - assignment operators
  - treatment of pointers
- **Powerful set of operators: high expresiveness**
- **But poor type checking: esp. function parameters**
- **Too flexible & very insecure**
- **Initially spread through UNIX: inexpensive or free compiler support**

C++ (C with Classes) (1985)

- **Developed at Bell Labs by Stroustrup**
- **C with Classes(1983): Evolved from C & SIMULA 67:**
  - function parameter type checking and conversion
  - classes
  - derived classes
  - public/private access control of inherited components
  - constructor and destructor functions
  - friend classes

  And later in 1981:
  - inline functions
  - default parameters
  - overloading of assignment operator

And later in 1984: (It has become C++)

- virtual functions
- function name and operator overloading
- reference types
- **And later** multiple inheritance, abstract classes, type-safe linkage**.**
- **Downward compatible with C**
- **Also has exception handling**
- **A large and complex language, in part because it supports both procedural and OO programming**
- **Rapidly grew in popularity, along with OOP**
- **ANSI standard approved in November, 1997**

Java (1995)

- **Developed at Sun in the early 1990s**
- **Based on C++**
- **Significantly simplified**
- **Supports *only* OOP**
- **Has references, but not pointers**
- **Includes support for applets and a form of concurrency**