Niklaus Wirth

Pascal, and its Successors
Pascal, and Structured Programming (1970)

- Pascal, a language appealing to systematic thinking, mirroring conventional, mathematical notation, satisfying the needs of practical programming, and encouraging a structured approach.
Pascal was built up on the notational grounds established by mathematics and Algol

- Examples:
  \[ x - y - z \quad (x-y) - z \quad x - (y-z) \]
  \[ x = y \quad x = = y \]
Bridging the gap between scientific community with Fortran and Algol, and business community with Cobol.

- Burroughs B5000 (1964)
- PL/I (1965)
- Pascal (1970)
Structured language supports structured programming

- **Statement structures:**
  - If, While, Repeat, For, Case

- **Data structures:**
  - Array, Record, Sequence, Set

- **Dynamic data structures:**
  - type-bound pointers
Deficiencies of Pascal

- Syntax
- GO TO statement
- No dynamic arrays
- Lack of full type specification of parameters
- Variant record as loophole for type casting
IF a THEN IF b THEN P ELSE Q

- IF a THEN {IF b THEN P ELSE Q}
- IF a THEN {IF b THEN P} ELSE Q

PROCEDURE P(b: BOOLEAN; q: PROCEDURE);
VAR i: INTEGER;
PROCEDURE Q; BEGIN i := i+1 END Q;
BEGIN i := 0;
  IF b THEN P(FALSE, Q) ELSE q;
  Print(i)
END P
Modula-2, and Modular Programming (1979)

A language adequate for describing entire systems

The module concept: information hiding

Definition and implementation parts
DEFINITION MODULE Files;
    TYPE File; (*opaque type*)
        Rider = RECORD eof: BOOLEAN END ;
    PROCEDURE Set
        (VAR r: Rider; f: File; pos: INTEGER);
    PROCEDURE Read
        (VAR r: Rider; VAR ch: CHAR);
    PROCEDURE Write
        (VAR r: Rider; ch: CHAR);
    PROCEDURE Length(f: File): INTEGER;
END Files.
Separate Compilation:
Static type checking across module boundaries

Deficiencies of Modula:
• type casting functions as loophole
• language too “large”
• INTEGER vs. CARDINAL
• lack of strings
• variant records as loopholes
Oberon and Object-oriented Programming (1988)

Two features characterize OOP:
- type-bound procedures (methods)
- type extension (subclassing)

Only type extension need be added
Several other features removed
Unification of def. and impl. parts
Consequence of its simplicity:
Implementation with few resources
- entire system: 200 Kbytes
- two persons, 2 years part time

• Power of language lies in its regularity and not in its abundance of features
• Character of language is defined by what it prevents more than by what it allows to be expressed