

Programming Languages Design Principles

I. Simplicity-

- Einstein: "Language should be as simple as possible, but no simpler."
- Edgar Allan Poe: "eschew surplusage"-
- Economy: a wide range of programs can be expressed using a relatively small vocab. & simple grammar.
 - A. Should not contain exceptions to rules or special cases, or multiple ways of doing same thing.
 1. COBOL: ADD 1 TO N and COMPUTE N = N + 1
 2. C: $n = n + 1$, $n += 1$, $n++$, $++n$
 3. static: means something different when it is global vs. when it is inside a function
 - B. Operator overloading could be problematic: single symbol (e.g. "+") could have more than one meaning
 - C. Too much simplicity: Assembly Lang. Prog.

II. Semantic completeness & Definedness

- Lang. complete enough to express intended semantics exactly (without having to "kludge" it)
- Lang. is unambiguous

III. Abstraction

- An abstraction presents an interface to an operation, object, or type of object which hides the implementation.
 - A. subprogram is an abstraction of a procedure, ADT is abstraction of an interpretation of data and relevant operations
 - B. abstractions allow a programmer to be clear succinctly
 1. Boolean types more clear than using 1 for TRUE and 0 for FALSE
 2. FORTRAN 77: no record type, so must use a collection of arrays with like indices
 3. FORTRAN 77: parallel arrays for binary tree as opposed to C node structs.

IV. Readability factors

- More time spent in maintenance than coding
- simplicity and abstraction make programs more readable
 - A. Lexical & Syntactic structure of program.
 1. FORTRAN identifiers could only have up to 6 upper case characters, could use reserved words as identifiers
 2. PASCAL's use of BEGIN & END -vs- C's use of { }- Language easier to read if it is familiar, i.e. English-like (as opposed to LISP or APL)
 - B. - Locality: the further away the effect of an action occurs, the harder it is to understand.
 1. This "distance" could be in execution time or lines of code.
 2. global vars. (BASIC), goto's & spaghetti code
 - C. - Lexical Coherence: code that logically belongs together should be placed together.
 1. code sections should be marked (PASCAL BEGIN & END)
 2. PASCAL's poor lexical coherence: variable declared at top of prog., but not used 'till much later

V. Distinct Representation

Each semantic construct should be represented by a single, different syntactic construct

- A. BASIC: line numbers used for both ordering lines & as branch destinations. Problem: if inserting many lines, can run out of numbers

- B. ALGOL 68 (?) constant: int num = 1; variable: int num := 1; They are very similar-
- C. Frequency: the more often a feature is used, the more convenient it should be
 1. area codes
 2. #include <stdio.h> in C is usually used, but not easy to use

VI. Writeability

Depends on design goals of program.

- A. Matrix operations simple in APL, but not in COBOL. Printing a financial report much easier in COBOL
- B. Simplicity: won't inadvertently use something incorrectly, so aids writeability
- C. Abstraction makes it easier to write progs.
- D. Powerful operators help (e.g. lack of "^" in C), APL

VII. Orthogonality

Orthogonality means the degree to which lang. consists of a set of independent primitive constructs that can be combined as necessary to express a program. (1st class in C or PASCAL)

- A. Features are orthogonal if there are no restrictions on how they may be combined.- Orthogonal means "at right angles" Envision a m x n grid. To represent m*n distinct combinations, you only need to learn m + n items. This assumes symmetry.
- B. 0-1-∞ [MacLennan, p. 124] Don't want to have to remember special cases.
 1. FORTRAN: 6 char. identifiers, max 19 continuation cards, max 3 dimension arrays
- C. IBM assembler vs. VAX:
 1. IBM: need 2 separate instructions (R+M->R, R1+R2-> R1)
 - a. A Reg1, memory_cell
 - b. AR Reg1, Reg2
 2. VAX: same instruction for both: ADDDL op1, op2 ; opx could be Reg. or Mem
- D. PASCAL: functions can't return structured types, operators usable only w/built in types (can't add two arrays or two records)
- E. Can be overly orthogonal
 1. BASIC goto a nonsensical line
 2. ALGOL 68 allowing expression on l.h.s. of assignment
 3. Functional Lang. highly orthogonal, but are inefficient
- F. Exceptions to a rule (restrictions) can prove helpful
 1. PASCAL array bounds check
- G. Exceptions to a rule may lessen flexibility or efficiency of code
 1. PASCAL array bounds check less efficient than C array indexing via pointer

VIII. Portability

Portable program can be run on various platforms, but loses architecture-specific enhancements

- A. Backwards compatibility: maintaining weak constructs from older architectures
- B. pre-ANSI C: return type of function that returns int may be omitted

IX. Reliability

Invalid prog. should be detected by the translator

- A. Type checking helps: it is easier & cheaper to detect an error earlier
 1. PASCAL array type & range check errors are caught, but not in C (well - lint)
- B. Exception processing makes a prog. more robust, e.g. processing input file and entire thing bombs on invalid format specifier for scanf
- C. Software development life cycle using modular design ends up with more reliable code

X. Cost of:

- A. training programmers: e.g. ALGOL 68 was too complex
- B. developing & maintaining programs (development environments, OOP)
- C. compilation & execution (LISP)

XI. Efficiency

FORTRAN had to compete with assembly lang.

- A. Ada & C++ are large & that limits (or used to limit) availability of compilers
- B. Lang. features themselves dictate efficiency: interpretation, range checking, optimization levels of compilation

XII. Design Tradeoffs

Cannot satisfy all the above.

- A. APL: great writeability, very poor readability: Daniel McCracken: 4hrs. to read & understand a four-line APL program
- B. Ada is complete, but is large & complex
- C. Reliability & efficiency conflict with one another