WIRTH74

Wirth, N.; On The Design of Programming Languages; Procedures of the IFIP Congress 74, pp. 386-393, North Holland, Amsterdam, North-Holland Publishing Co.

SYNOPSIS

The paper presents the author’s arguments for simplicity in programming languages. A goal which should be achieved through "transparency and clarity of its (the language’s) features and by a regular structure, rather than by utmost conciseness and unwanted generality".

The author first considers some previous attempts to introduce simplicity through generality, in particular his experience with the language Euler and concludes that such an approach leads to a language where

1) the compiler cannot help the programmer in detecting mistakes. i.e. nearly everything is a legal program.

2) conciseness and lack of redundancy in the program make it difficult for the human reader to grasp its meaning.

He then considers three features commonly included in high level languages which he considers a throwback to machine and assembly coding, and how a more restricted version of these features could be included without the disadvantages of the fully generalized versions. These features are

1) goto’s: The programmer’s desire for goto’s can be satisfied by more flexible control statements. e.g. the ’while’ and ’repeat’ statements for repetition and the ’if’ and ’case’ statements for selection.

2) pointers or references: Here full generality leads to the aliasing problem. His more restricted version allows pointers to be bound to only a single type, and pointers may only refer to anonymous variables (partially solving the aliasing problem).

3) data types: Instead of allowing untyped operands that force the programmer to know the underlying machine representation , the record (as in Pascal) shifts this burden to the compiler while the language retains the conceptual simplicity of the data type.

The main point in these sections is that simplicity can be achieved through carefully chosen restrictions of more general concepts. These examples also provide justification for some of the features of Pascal since the restrictions he proposes are all present in Pascal. The author also considers some other pitfalls of language design. The restrictions chosen as above must be compatible, " the combination of two seemingly harmless and well understood features may have disastrous effects". There should be no hidden inefficiencies, the programmer should have a good idea of the computational effort involved in a particular feature.

The paper ends with a list of hints, conclusions and goals that the prospective language designer should consider.

COMMENTS.

Seen from 1984, Wirth’s arguments for simplicity in programming languages (advocated in the early 70’s) seem almost obvious since many of these approaches are now the ’accepted wisdom’. The paper however contains many valuable insights from an experienced language designer which are still valid today and it gives a good overview of programming language design at that time elucidating some of the ’historical’ reasons for features seen in today’s programming languages.