QUALITY - IN WHAT IMPORTANT WAYS DO PROGRAMMING LANGUAGES DIFFER?

Professor A. J. Perlis

Abstract:

Two courses in computing science, the second and third of a series, are described.

In one, attention is drawn to the varying qualities of programming languages by examining data structures and the features which are provided to handle them. Criteria by which languages may be compared are then discussed.

The other course examines the relevant properties of programs built to handle other programs. An exercise set during this course is described.

Rapporteurs:

Dr. J. Eve
Mr. A. Alderson

1. Introduction

In his first lecture, Professor Perlis described a first course in computing science. In that course the object was to produce a synthesis of programming issues through the use of only one language, arguing that if little is known about programming the choice of language is of no concern to the student. Having decided in that course not to raise the question of choice of language one is confronted with deciding whether the question should be raised at all.

Professor Perlis expressed the view that while using one language gives a common base for discussion, attempting to widen the scope of the problems beyond a set hand-tailored for solution in, for example, Algol 60 raises questions which are of no relevance to Algol 60. Thus, if one considers the rate of growth of scope and of problem generation, one cannot tell students that one language is all they need ever know. Indeed, it is instructive that there exists no language which is of equal facility in all fields of problem solving.

2. A second course in computing science

One object of the second course, which Professor Perlis described, is to convey to the student the dual purpose of a programming language: as a notation for human communication and as a carrier of algorithms which are to be executed. Another object of the second course is to show why certain problems should be programmed in a particular language. Part of students' education is to examine problems from different areas of the whole problem space and to make reasoned judgements as to which languages are most appropriate for attacking these problems. A number of points arise from this exploration.

1. How many languages should be taught?
2. Why are languages so disparate in their capabilities even though they may succeed one another in time? Why do new languages discard good as well as bad features of previous languages?
3. Since a choice between languages exist can their important features be characterised?
4. It must be realized that the environment in which a program is executed may be a factor as important as the characteristics of the problem when making the choice of programming language.

It is just beginning to be realised, Professor Perlis commented, that to design a language and hope that it will remain independent of the system is silly.

3. Data structures in programming languages

In discussing these matters one is led naturally to examine the qualities of programming languages. Data representations are one of the first features to which Professor Perlis draws attention. He introduces at an early stage problems which demand special forms of data representation. Data structures are examined from the point of view of the ease with which complexity may develop and to display the features which differentiate one class of data structures from another. Differentiation is made as follows.

A string is a collection of data which is linearly ordered and which has the property that any arbitrary subsegment can be replaced by any other arbitrary subsegment of any length and after the replacement the boundaries of the replacement are no longer determinable by examination of the string itself. That is, the ordinal position of the data is of no interest. Having discarded the use of ordinal position to locate data we must use some other method and the concept of patterns is utilized. Thus, if a programming language does not have this capability on a natural level then it is not a string processing language. The natural language for string processing is Snobol.

A list is a data structure in which the law of substitution is such that any primitive element in the structure may be replaced by an arbitrary structure. When the substitution has been made the process of finding that data is a simple reversal of the method of substitution. The natural way to handle lists is by recursion and since at any node in the list one wishes to obey conditional statements, these capabilities must be very near to the surface in a list processing language. Thus Lisp is the natural choice.

Faced with the unfortunate problem which involves both lists and strings, in Snobol one would suffer due to the former and Lisp for the latter. Professor Perlis commented that the most natural way to attack such a problem would be to program the list part in Lisp and the string part in Snobol and join them by transfer functions. Unfortunately, systems with this capability do not exist yet. Only in these sad cases, in

Professor Perlis' opinion, should one use a language such as PL/1 which attempts to provide both.

Professor Perlis felt that APL has in a sense provided a standard for the handling of arrays, and he thought it inconceivable that any future language allowing arrays as units of operation should be significantly inferior in this respect than APL. It is unforgivable, he said, that any new global language should ignore the good points of earlier languages, and the design of a new language should be a synthesis of the best parts of the four standards which we have today, namely, of Algol 60, Snobol, Lisp and APL.

4. The comparison of programming languages

Once the student is aware of these reference points for data structures, he is encouraged to examine other languages to see with what facility they are capable of handling these structures. He is also asked to classify a number of languages in terms of the following qualities.

1. Generality. Even though the student may not have written many programs he acquires a feeling for the generality of a language.

2. Conciseness. Although it is commendable that comments should be made in programs, Professor Perlis expressed the view that aids to writing should take precedence over those of reading programs. Indeed, languages which are conveniently concise provide the necessary comment.

3. Consistency. Van Wijngaarden has introduced the concept of orthogonality, in that a concept of a language should be easily and naturally used everywhere that it ought to be used.

4. Simplicity. All of the apparent complexity of the language should be on the surface.

5. Familiarity. If there exist notations which are adequate does the language use them in a familiar manner?

6. Intuitiveness. The language should reflect human thought processes permitting ideas to be written in an order and form which is natural when thinking about them.

7. Legibility. The language should be a good vehicle for the communication of algorithms.

8. Implementability. What in the language, would make compilation or interpretation difficult? Some of the better students are asked to consider what would have to be removed or inserted to transfer the language from one form of implementation to the other.

9. Efficiency. The language should be constructed and implemented to take advantage of efficient constructions in the target machine.

10. Machine-independence.

11. Best treatment for common cases. Do those things most commonly needed come easily in the language?

12. Acceptability. Does the language have properties which will attract users?

13. Adaptability. Can the language be extended?

14. Richness. Does the language have within it a wide variety of programming techniques?

Professor Perlis summarized the aim of this second course in Computing Science as trying to have the student realize that there are certain features which one should look for in programming languages and that these features divide into data structures, operations and control.

5. A third course in computing science

Having acquainted the student with a point of view which it is hoped will make him confident in the use of several programming languages in general problem solving situations, the third course concentrates upon a specific problem area, namely, structures which are built to manage other programs, i.e. assemblers, compilers, interpreters, etc. The only convenient language for communicating the algorithms and structures encountered in this course was, in Professor Perlis' view, APL. It alone is sufficiently concise for this purpose. In this course the major programming entities are examined to uncover their relevant properties. For example, the properties which assemblers should have are as follows:

1. It should be possible to write in the machine code into which the programs are assembled, within the assembler language.

2. There should be psuedo-operations.

3. Macros should be available.

4. It should be possible to have macro definitions and macro calls within other macros.

5. It should be possible to instruct the assembler to map output into various kinds of storage maps.

6. Partial assembly should be available.

7. Linking of the assembler code with other programming languages should be possible.

8. One should be able to write an assembly program and put it into a library.

9. Programs should be easily altered.

10. Tracing and monitoring facilities should be available.

Each of the major program management systems are examined in this manner, and it is found that all of them are very similar in that they all have certain standard features.

A typical exercise set during the course is the following. Snobol is available on a machine under a batch operating system. Define and build a Snobol interactive system so that the programmer may interact with the running program in a number of ways, and so that he may keep his program stored in a number of different representations. Students quickly realize that the obvious way in which to attack the problem is to program in Snobol, since one has available the statement 'compile this string'. Eventually the students find that they would like a facility similar to the quad in APL which allows entries from the terminal in the middle of a statement. Unfortunately, Snobol did not anticipate this with the consequence that, since the Snobol executor assumes that all parsing will be done before execution commences, the parser has disappeared from view and a piece of program text cannot be entered in the middle of a statement.

With experience, such as may be afforded by the above problem, by the end of the course the students are realizing that in structuring systems at present we never guess correctly and so decomposition of them will be necessary because of new contingencies. The final theme of the course is that processors should be designed in small pieces which are 'screwed' rather than 'welded' together.