TEACHING PROGRAMMING TO ENGINEERS

- THE CHOICE OF LANGUAGE

by

A. S. MORRIS

Dept. of Control Engineering,
The University of Sheffield,
Mappin Street,
Sheffield. S1 3JD
U.K.

RESEARCH REPORT NO. 168

This is a review paper, assessing the programming education needs of engineers and the attributes of the various programming languages which are usually considered in choosing a suitable language for teaching.

TEACHING PROGRAMMING TO ENGINEERS - THE CHOICE OF LANGUAGE

by

A. S. Morris, B.Eng., Ph.D., C.Eng., M.I.E.E.*

Abstract

The paper examines the merits of the major languages presently available for teaching computer programming to engineering students. Consideration of the ultimate application of programming by engineers is given prominence in this evaluation, which it is hoped will fill a gap in the information presently available to planners of engineering courses. Recommendations about the best language are made from engineering considerations, and offer an alternative to the much less application-orientated criteria typically adopted by University Computer Science departments, who are the traditional source of expertise in such matters.

1. Introduction

Some knowledge of computer programming is an essential part of every professional engineer's education and as microcomputers proliferate in industrial applications, this becomes of ever greater importance. However, planners of engineering courses always face a conflict between the amount of material which should ideally be taught and what can be reasonably fitted into the timetable, and the usual outcome is that programming instruction is limited to one language only. The normally accepted sources of expertise in the best language for teaching computer programming are University Computer Science departments. There is a tendency in the latter however to treat programming as a science in itself rather than as a tool, and languages are recommended on the basis of their goodness in structured aspects etc., with insufficient regard to application limitations. It is the purpose of this paper to take an engineering viewpoint in

---

*Department of Control Engineering, University of Sheffield.

unravelling the pertinent considerations in choosing a suitable language for teaching programming to engineers.

2. Choice of language - the pertinent considerations

The first consideration is to try to anticipate the programming requirements which an engineer will meet in industry and to choose a language which will best meet those demands. The minimum level of programming involvement which an engineer is likely to meet is to have to occasionally read programs written by other people for data analysis purposes etc., and perhaps to write small programs or modifications to existing programs himself. It is for this relatively large group of engineers with spasmodic involvement in programming that correct choice of language in programming teaching is most important. The chosen language should be that which such engineers are most likely to meet in industry, as it will be neither time nor cost effective for the practising engineer to have to start learning languages other than the one he has been taught, where the needs for such knowledge are infrequent. There are of course other engineers whose work involves a heavier programming content, ranging from writing off-line data analysis routines to on-line computer control algorithms. For such engineers, the criteria for choice of their initial training language are different. To such engineers, an initial teaching language encouraging good program design techniques is most approriate, as their heavier programming involvement will justify the cost of subsequent retraining in languages necessary for particular applications. The programming education needs of these two separate groups of engineers are substantially different and complicate the task of choosing a suitable language for programming training.

The speed at which programming skills can be assimilated is another important consideration in the choice of language, particularly in view of the lack of space in engineering timetables. This favours interpretive rather than compiled languages. In interpretive mode, each line of a program is executed immediately that it is entered from an interactive computer terminal, by an interpreter resident in the computer memory.

This means that any syntax errors are immediately rejected by the interpreter, and a working program is achieved much more rapidly by a novice programmer. The serious disadvantage of interpretive languages outside teaching applications is the slow speed of program execution, which is a direct result of the interpretive mode of operation. A compiled language, on the other hand, is more tedious for a novice programmer, as the whole program has to be entered before compilation, and it is only during this latter procedure that syntax errors are flagged. Program development thus becomes an interative procedure of compilation and editing until an error free version is obtained. This can be a relatively lengthy process during the early stages of learning to program. However, such a compiled language is often vastly superior in practical application, as the machine code generated executes at a much higher speed than interpretive code.

The qualities of the programming skills being taught are also very important. Programming is essentially a human-computer communication and the purpose of a high level language is to facilitate the communication to the computer of an algorithm which the programmer wishes to have evaluated. Compiled languages vary in their efficiency at doing this, both in the size of the object code produced for a given algorithm and also in the execution speed of the code. The other aspect of communication which a programming language should facilitate is human-human communication related to the program. In the case of long programs, it is often convenient to have several programmers working in parallel in developing separate parts of the program. These separate parts need to subsequently fit together with as few problems as possible, and the success of this is a measure of the quality of the language. Program readability is a prominent factor in this, and also very important to maintenance of programs and further development occurring at a later point in time than that when the program was initially written. It is in such realms of human-human communication related to programs that structured programming languages have particular

advantages. Structured languages also encourage a manner of programming which minimises logic errors and tends to minimise program development time, which by itself has important cost benefits. Structured languages constrain the flow of control in a program to predermined forms of looping and conditional branching, discouraging any arbitrary change of control such as allowed by the GOTO statements of unstructured languages. Furthermore, a problem solution is perceived as a series of identifiably separate steps, some small, some large, rather than a long list of single instructions, that is to say a structured language is very modular and thus easy to read.

The three languages commonly taught to engineers at present are Basic, Fortran and Pascal. There are of course very many other languages available which qualify for consideration in engineering software training. However, none have anything like the universal availability on most computers enjoyed by Basic, Fortran and Pascal and so are not considered here for that reason. An exception to this is ADA, which although not presently available, promises to become a major engineering language in the future and so deserves some mention.

3.1 Basic

The acronym BASIC stands for Beginners All- purpose Symbolic Instruction Code. It is an unstructured language which is available on some computers in both interpretive and compiled forms, although only as a interpreter in many manufacturer's implementations. Because of this lack of general availability of a Basic compiler, this discussion will be restricted to consideration of Basic as an interpreter, as this will be the restriction applicable to many users. Without commenting on the quality of the programming skills imparted, Basic in its interpreter form offers the fastest way of teaching programming, being more like plain English than any other high level language. The simplicity of the limited instruction set is a positive advantage in teaching, and this, together with the general advantages of an interpreter,

makes Basic a language which quickly produces students capable of writing programs of moderate complexity.

In engineering applications, Basic has achieved some popularity for programming in many microcomputer - controlled processes whose time constants are such that the slow execution speed of a Basic interpreter is not a disadvantage, and this application has seeded the growth of many extensions to the standard instruction set to allow file handling, floating point arithmetic and string variable manipulation etc., albeit at an elementary level. This usage has arisen largely because of the much smaller memory requirement of an interpreter compared with a compiler. However, as memory hardware costs are falling, and the addressing handwidth of microprocessors increases, this advantage of Basic is being rapidly eroded.

The disadvantages of Basic in engineering applications are considerable. Apart from the problems of execution speed already mentioned, the limited instruction set of Basic starts to become a difficulty in problems of any complexity, Whilst it is true that extensions to the standard instruction set have gone some way towards overcoming these limitations, such additions have generated a further problem of non-portability. There is now a proliferation of Basic dialects, each extended in different ways, and conversion of programs from one form to another to allow portability between computers can involve a very considerable amount of work. Compounding these difficulties are the general deficiencies of all unstructured languages.

3.2 Pascal

Pascal is the newest of the three languages commonly taught and is the only one of the three to be structured. It is a block structured language developed in 1971 by Prof. Niklaus Wirth in Zurich and intended, at least originally, for the needs of the academic teaching community. The first definitive text[1] on the language became available in 1974, though the

language has yet to achieve an internationally agreed standard. Whilst the International Standards Organisation has published a draft standard, this is not compatible with the UCSD version (University of California at San Diego), which has already been adopted by several computer manufacturers. These developments do not augur well for Pascal to ever achieve a universally agreed standard.

Both sequential and concurrent versions of Pascal exist, although in many computer manufacturers' implementations, only the sequential version is available. Concurrency is a particularly useful feature in real-time control applications and describes Pascal's ability to handle multi-tasking within the same program. Each separate process within the controlled system exists as a separate process within the one Pascal program, and these processes are executed in parallel with any necessary coordination and critical timing being handled by Pascal. This offers significant advantages over sequential versions of Pascal and other languages such as Fortran, where multi-tasking execution is the responsibility of the operating system, with each single process being written as a separate program.

Pascal consists of a relatively small instruction set of simple and easily understood data types and control structures. Particularly useful features are the facility of user-defined data types to suit the special needs of the task being programmed, and also the ability to limit the allowable range of scalar variables. Portability has been a major consideration in its design throughout. In common with most structured languages, code is automatically checked at compile time, avoiding many of the errors which do not emerge until execution time in other languages.

The advantages of Pascal's simplicity as a teaching language become its main limitations in engineering applications. There are no facilities for random disc access and dynamic array dimensioning , and no support for real-time input/output interfaces. Where language extensions allow these facilities

the extensions are non-standard and so the problem of non-portability is introduced.

Another deficiency of Pascal is the lack of availability of libraries of mathematical algorithms and functions which are essential in engineering application programming. This is largely due to the relatively young age of Pascal and may be put right in time. Several Pascal compiler versions allow linkage to Fortran libraries and while such implementations allow access to the necessary mathematical algorithms, we are again in the realm of compiler extensions and non-portability. A further disadvantage of Pascal at present is that it is not universally available on all computers, and very importantly not on IBM machines.

3.3 Fortran

Fortran stands for FORmula TRANslation and emerged in the late nineteen-fifties as a means of giving a form of organisation to an otherwise clumsy list of machine code instructions representing a mathematical algorithm. It thus represents in many ways a 'first attempt' at a high level language, and many of its undesirable features ensue from this. The standard for the Fortran 4 version dates from 1966, and in this form the language is very unstructured. Since 1966, the language deficiencies have been comprehensively analysed by committee and out of this has evolved a new standard, Fortran 77, published in 1978. This has several new features, particularly an IF-THEN-ELSE construct, which allow programs to be written in a more structured way, although admittedly this structuring does not have the elegance of Pascal. A very important point about the new standard Fortran 77 is that it is upwards compatible with programs written in Fortran 4 and so no software conversion work is necessary to existing programs when adopting Fortran 77.

Where Fortran gains over Pascal in its language features are its comprehensive I/O facilities, its random disc access capabilities and, in

Fortran 77, the string-handling functions. Probably the biggest advantage which Fortran holds over Pascal though is the wealth of engineering support subroutines and program packages which have been developed over the last twenty-odd years. This software will take a very long time to translate to Pascal, even if the advantages of Pascal make it cost-effective to do so, and in the meantime the existence of this software investment will continue to favour Fortran over Pascal in the choice of language for industrial applications. Whilst it is true that, recognising the need to access this software investment, the facility now exists in some Pascal implementations for linking into Fortran subroutine libraries from Pascal programs, this feature is by no means universal.

3.4 Ada

The initiative for the development of Ada came from the United States Department of Defence (DOD) in 1975, who were concerned at high software costs and wished to standardise on a language which would improve programmer performance and aid program compatibility and portability. Four years were spent in defining the requirements of the language and Ada was finally born in specification form in 1979. It was named after Ada Lovelace, a colleague of Charles Babbage. Ada is a highly structured language, and in essence it can best be described as a substantially expanded version of Pascal, overcoming many of the deficiencies of that language in engineering, particularly real-time, applications. Whilst it has been hailed as a very important advance in computing by many commentators, it has also been condemned in some quarters as misconceived, poorly specified and over-complicated[5].

No compilers are yet available, though a version conforming to the DOD specification is expected from Intel in Mid-1982. The anticipated size of the compiler is in excess of 500K bytes[6] which is likely to prove a major disadvantage of the language in both application and teaching environments.

Further comments on its suitability as a teaching language must await a future time when compilers become generally available and the qualities of Ada in serious use can be assessed.

4. Language Evaluation

Published evaluations of these languages is scarce. Rundle[2] has carried out an evaluation in the form of a questionaire to industrial users, and in each of the categories of 'ease of use', 'efficiency' and 'overall satisfaction', the ranking was Fortran top, Pascal second and Basic last. A more numerate comparison between Fortran and Pascal has been carried out by Anderson[3] using a benchmark devised by Curnow[4]. This showed Fortran to be approximately 10% better than Pascal, both in terms of object code size and execution speed, with the same level of run-time support. If the better run-time debugging support of Pascal is included, execution speed is made even slower and object code size becomes even larger.

Ada has to be ruled out immediately as no compiler is yet available. Pascal and Basic are similar both in the facilities offered by their instruction sets and in the rate at which the language can be learnt. Pascal is vastly superior in terms of the structured programming skills which it gives the novice programmer. In the past, Basic has enjoyed an advantage in respect of its lesser demands on computer memory, but this now carries little weight with present-generation microcomputers. These facts provide good reason for favouring Pascal over Basic as a teaching language.

The choice then remains between Fortran and Pascal, where only one language can be taught. However, this decision is not clear-cut in anything like the same way as between Basic and Pascal. Strongly in favour of Pascal is its structured approach, whereas Fortran gains in terms of its extra facilities in input-output and random disc access. Fortran also appears to be superior in terms of the efficiency of the object code produced, and it has a large advantage in respect of the past software investment in subroutine

libraries. Additionally, the newer Fortran 77 has some pseudo-structured features, whilst retaining upwards compatibility with programs and libraries written in Fortran 4. For these reasons, Fortran is likely to continue to be the language most generally used in industry for some time to come.

5. Conclusions

Section two outlined the pertinent considerations in choosing the best language for inclusion in the computer programming aspect of engineering courses. These considerations are to a large extent mutually exclusive, and the problem becomes one of deciding on suitable weighting factors for these considerations in order to arrive at the best choice. It is felt that the heaviest weighting should be placed on the requirement that the chosen language should be that which engineers will be most required to use for the forseeable future in industry. This therefore leads to the recommendation that, where timetable restrictions limit programming instruction to one language only, that language should be Fortran, but in the Fortran 77 version which encourages some form of structured programming.

The conclusion ignores the considerable advantages in teaching a properly structured language, but is felt to be the best compromise at the present time within the normal timetable restrictions. Should the timetable be able to cope with teaching two languages, however, then the recommendation would be that Pascal should be taught first, in order to instil good structured programming skills, followed by Fortran to fulfil the need of industrial engineering applications.

Finally, it is very important to stress that this analysis only applies to the present time. Developments in the computing field are rapid, and reassessment of the position should be made annually in order to judge the continuing validity of the arguments used in making the present recommendation.

References

1. JENSEN, K. and WIRTA, N. 'Pascal user manual and report', Springer-Verlag, 1974.

2. RUNDLE, A. Microcomputer analysis, Nos. 3-5, 1979.

3. ANDERSON, R.E. et al, 'Computer language evaluation'. U.S. Department of Energy, Lawrence LIvermore Laboratory, Internal report.

4. CURNOW, H.J. and WICHMANN, B.A. 'A synthetic benchmark', The Computer Journal, 19, 1, 1975, pp.43-49.

5. THOMAS, M. Systems International, Aug.1981, pp54-56.

6. Infomatics, June 1981 p.6.