# A REVIEW OF PROGRAM PORTABILITY AND FORTRAN CONVENTIONS

by

D. T. Muxworthy
(University of Edinburgh)

PREFACE

This document was prepared under contract 625-76-06 SISPE at EUROCOPI, Euratom, Ispra. The author wishes to express his thanks to H. J. Helms and G. Gaggero of Euratom for providing all possible facilities and help and to Mrs. M. M. Barritt, Program Library Unit, University of Edinburgh for making time available to him. Thanks are also offered to members of ECSIR Working Group 4, especially J. D. Bevan, B. Ford, G. Gaggero, O. Murro and M. Sund, for supplying papers and suggesting areas of investigation, and to Mrs. A. Dorpema of Euratom for her extraordinarily quick and accurate typing of an indifferent manuscript.

David T. Muxworthy
4 August, 1976

CONTENTS Page

Contents (contd.) Page

Contents (contd.) Page

## INTRODUCTION

This report is intended to play a dual role. Firstly it is a review of Fortran Standards and of publications and practices relating to portability of programs and as such it is hoped that it will be of interest in its own right. Secondly it is intended to act as a discussion document for Working Group 4 of ECSIR (European Consortium for Software access and Information transfer in Research and teaching), whose remit is to investigate programming practices and conventions to ease the exchange of programs.

The two main sections on Fortran Standards and portability each have a short introduction outlining the areas they cover. A projected third section, on documentation standards, was abandoned when it was found that only a small proportion of the relevant literature was to hand; a summary of publications remains.

# 1. FORTRAN STANDARDS

## 1.1 Summary

This section relates the current Fortran Standards to the circumstances in which they were drawn up. It reviews publications about the Standards, both interpretations and comparisons with implementations, and describes those areas which have given particular problems. It also discusses observance of the Standards by implementors and by users. Finally, the revised proposed Standard of 1976 is outlined and discussed.

A distinction is drawn between those who design and implement Fortran compilers and those who use Fortran as a programming language, the groups being designated by the terms "compiler writers" and "programmers" respectively.

These terms are used for convenience only and no pejorative implications are intended.

## 1.2 Current Fortran Standards

A number of standards for Fortran exist; they follow an almost identical pattern and differ mainly in the level of language facilities incorporated. The most important one, and the one which is usually meant in the absence of any further designation, is the American National Standard for Fortran (ANSI, 1966a) as qualified by two sets of clarifications (USASI X3, 1969 and ANSI X3J3, 1971). A standard with fewer facilities, known as Basic Fortran, was drawn up concurrently by the same committee and was approved at the same time (ANSI, 1966b). Meanwhile the Fortran committee of the European Computer Manufacturers' Association, working in collaboration with the then American Standards Association, devised a standard midway in facilities between the two American ones (ECMA, 1965). Essentially these three standards, with insignificant word changes, were adopted in 1972 by the International Organization for Standardization as an ISO Recommendation for Fortran (ISO, 1972) with three levels numbered simply 1, 2 and 3 corresponding respectively to ANSI Fortran, ECMA Fortran and ANSI Basic Fortran. A note on the changes of name of the American Standards body, from ASA to USASI to ANSI, and further references are given by Muxworthy (1972a).

At the time of publication of the standards the compiler writers for several computers, particularly European ones, appeared to have difficulty in fulfilling their requirements and a number of compilers for second generation machines were issued which were markedly substandard. However, advances in hardware and software techniques have all but made the lower two levels of standard redundant. In practice the ISO work was a mere formality and the American Standards have been defacto international ones since 1966, or even since two years earlier when a draft was published in CACM (ASA, 1964).

Under the rules of ANSI a standard, once approved, lapses after five years, unless it is reaffirmed or unless notice is given that work is in progress to revise it. The Fortran standard was thus due to lapse in 1971 and work began in 1970 to prepare a revision with the expectation that it would take about two years. The draft proposed (revised) Standard for Fortran appeared for public review only in March 1976 (ANSI Subcommittee X3 J3, 1976) and as yet has not been approved as a standard. This document is considered in more detail below (section 1.8).

## 1.3 Pre-Standard Fortran

It is useful to consider briefly the background against which the first standards were drafted. A preliminary specification for Fortran was issued by IBM in 1954, a manual appeared in 1956 and the first software was released for the IBM 704 in 1957. This was succeeded in 1958 by Fortran II which incorporated the subroutine concept together with the common block. By 1961 IBM had 8 different compilers on various systems and already differences in dialects had become such a problem that IBM issued a manual contrasting the facilities available in the 8 compilers. The first Fortran system on a non-IBM machine had appeared in 1960; this was ALTAC on the Philco 2000 which was an extended Fortran II. The first system called Fortran on a non-IBM machine was Fortran I on the Univac SS 80 in 1961 and by 1963 all major manufacturers had either implemented or announced Fortran for their computers. The following year Oswald (1964) was able to state, in his overview of 16 Fortran systems, that 43 compilers existed in all.

Also during the early nineteen-sixties the SHARE organization and IBM designed a revised Fortran, known as Fortran IV, which had significantly more facilities than Fortran II and was similar in concept but which was not fully compatible with the earlier language; Fortran III existed within IBM but was never released. Fortran IV software first appeared on the IBM 7030 during the summer of 1962 and was followed within the year by systems for the IBM 7090/94 and the Univac 1107.

The ASA Fortran committee met for the first time in May 1962 with the chief objective of producing a standard which would facilitate the machine-to-machine transfer of programs; the standard was to be a reference document both for programmers and implementors (Heising, 1964). The bulk of their work was completed inside 18 months and, being concerned with language growth, they standardized on Fortran IV rather than Fortran II. Basic Fortran was designed to be comparable in power with Fortran II but to be a subset of full Fortran. Although it is often described as a proper subset of Fortran, it is not the case that a program written in Basic Fortran will necessarily produce the same results when run on a full Fortran processor; this is because of differences in the lists of supplied functions. Experience with the Standards in practice is discussed

below (section 1.7).

More detailed histories of language developments during these years are given by Bemer (1969) and Sammett (1969). Bemer in tracing the relative positions of Algol and Fortran in the USA and in Europe is especially interesting. In 1960 for example it was thought in the US that Fortran would shortly be eclipsed by Algol; in 1961 Bemer himself, then with IBM, was quoted as saying that Fortran had served its purpose and was to be deleted by IBM; as late as 1965 it was possible for the head of a well-known European software house to state that as there was no vested interest in Fortran in Europe, the superiority and availability of Algol would ensure that Europe would favour that language. Sanders and Fitzpatrick (1963), in an analysis which is still valid, show why they were wrong. Both Bemer and Sammett give a multitude of references.

## 1.4 Scope of the Standards

The Standards specify the syntax and semantics for a set of statements and the form of input and output data. They do not attempt to be exhaustive and they explicitly avoid defining the mechanism by which programs are transformed for use on a data processing system, the method of transcription of programs and input and output data, the results when the rules for interpretation fail, the maximum size or complexity of a program and the range and precision of numerical quantities. Further, they describe themselves as permissive; that is that "prohibited" implies "undefined", that it is open to any Fortran processor to define that which the standard leaves undefined and for any Fortran processor to be described as standard-conforming provided it will process the standard language as a subset.

The standard language is thus both a lower limit for a compiler writer and an upper limit for a program writer.

## 1.5 Interpretations of the Standard

It was a common complaint that the Algol Revised Report was incomprehensible to the average programmer and Rabinowitz (1962) was prompted to reply with Fortran II in Backus Normal Form. The Fortran Standards, although written in English with little symbolism, are also cryptic documents and may not easily be referenced by the uninitiated. Although some compilers and a few manufacturers' manuals (the IBM manual is excellent in this respect) indicate which facilities do or do not syntactically conform to the Standard, in general it is not easy for the average programmer to determine exactly what the provisions of the Standard are. The (British) National Computing Centre (1972a) therefore produced a manual which was intended to fulfil this function. Despite a number of flaws (see reviews), it has proved useful. Larmouth (1973) has given a less comprehensive but more sympathetic interpretation of the Standard.

A number of Fortran "standards" based on the wording and paragraph numbering of the American standards have appeared, usually for private use within commercial companies. Typically they describe the common subset of Fortran dialects in use on computers owned by the companies and are a superset of the Standard proper. A similar document by Friedrich (1975) for the DVM (Datenverarbeitung in der Medizin) Project in Germany serves the dual purpose of providing a German language version of the American standard and defining a recommendation for use within the Project.

## 1.6 Comparisons of Implementations and Standards

It was noted in section 1.3 that by 1961 IBM saw a need to publish a tabular overview of the facilities offered by their 8 compilers. More wide ranging comparisons were made by Oswald (1964; 16 compilers), McCracken (1965; 27 compilers and the draft Standards) and Wright (1966; 5 compilers). As experience with the Standard and with compilers grew and more subtle differences became apparent, these rather simplistic comparisons were joined by papers which gave more detail than an indication of the presence or absence of a facility. Schofield (1968) published a comparison of 12 compilers with the Standard in tabular form and the following year Stuart (1969) included as an appendix in a text book a massive table covering over 70 compilers. Berkowitz (1970) compared six compilers with the Standard in a reference list format and Muxworthy and Shearing (1970) also described six compilers in a form designed to be readable. Similar work is still in progress at Euratom, Ispra (G. Prinzivalli) and at CSATA, Bari (O. Murro, for minicomputers).

It is an indication of the diversification of Fortran dialects that amongst the 6 systems considered by Muxworthy and Shearing (CDC 6000, Honeywell 200, IBM 360, ICL 1900, ICL System 4, Univac 1108), the only Fortran statement found to be implemented without any extensions or contractions whatever vis-à-vis the Standard was the unconditional GO TO. Not even the CONTINUE statement qualified for this description. All the systems were non-standard, some more seriously than others, in the sense that it was possible to write a simple standard-conforming program which would not be processed in the prescribed manner on that system.

In general new features had been implemented in a similar manner although a single function sometimes had been achieved with statements of quite different syntax. A potentially much more troublesome matter for the transfer of programs was that extensions had been made in which a particular syntax had a different meaning on different systems (e.g. ENTRY, alternate RETURN).

Schofield's paper also showed that apparently trivial extensions, introduced without care, can bring ambiguities into Fortran. For example in one of the systems he considers, the statement

10 FORMAT(X9H) = FMT(X9H)

could be a FORMAT statement, a statement function definition, or an assignment statement. Despite the ad-hoc nature of the Fortran Standard, ambiguity has been successfully avoided.

## 1.7 Discussion

### 1.7.1 Relating to the Scope

The publication of the Standard did much to remove some of the differences between the systems of the mid-sixties; in 1964-66 there was considerable activity in upgrading existing systems (cf. McCracken, 1965) and many programmers were no doubt surprised that their supplier had gone to the trouble of implementing, say the G format code and issuing manual update pages when they had managed quite well without the feature and could see no necessity for it. But the Standard did give a firmer base, as was intended, for writing machine independent Fortran programs.

The Standard has frequently been criticized for its permissiveness but it is difficult to see what other course the committee could have taken. This was the first time that a language had been standardized and the committee were working at a time when a number of differing systems were already established and a larger number were still being developed. This contrasts with the case of Algol where although some systems existed, on the whole the design was published before the compilers were released and there was no entrenched body of users to satisfy; even so Algol programs have proved to be no more transportable than Fortran ones, and for similar reasons (cf. Brown et al., 1971). The Fortran Standard's permissiveness has deliberately allowed implementors to add completely new statements, to extend existing ones and to provide interpretations for situations which are "undefined" or "prohibited" according to the Standard. In general the programmer is given little indication by the compiler software or documentation as to whether his program is standard-conforming and the onus is usually on the programmer to ensure that it is. Most compilers are concerned only to process their dialect of Fortran and often they make few checks, if any, at execution time on whether the rules of the Standard or even those of the dialect are being observed. For example it is the exception rather than the rule to be able to check if array bounds are exceeded or if variables to which no values have been assigned are being used for computation.

While it is relatively easy to check the syntax of a program for standard-conformity at compilation time, it may however be impractical to check a program throughout its execution. The kind of checks mentioned above increase program sizes and running times dramatically and more subtle violations, such as the use of functions with side-effects, require the kind of scrutiny that incurs unacceptably high overheads in production systems.

The separable compilation of subroutines is one of Fortran's great strengths but it makes for difficulties in checking for consistency between subroutines. It would appear to be impractical to ensure mechanically that a given program, whatever data are supplied, will always conform to the standard. Even such a basic statement as A = B + C is non-standard if it causes overflow and even A = B can overflow if B is unnormalized and normalizing it causes overflow.

The standards committee excluded the range and precision of numerical quantities from the scope of the standard and implementors have naturally used the values which are most suitable for their hardware. No language has yet solved the problems of defining numerical range and precision satisfactorily and the problems of the various types of floating-point arithmetic available have hardly been discussed in the context of high-level programming languages. Again, the onus is on the programmer to ensure that the system on which an algorithm is processed is adequate for its needs. In practice surprisingly little use is made of environmental constants (Naur, 1967) or of self-checking programming techniques.

Nevertheless Fortran is put to many uses other than numerical computation and in a considerable proportion of Fortran routines this is not a problem. Possibly a greater hindrance to the transferability of programs between systems is the variation in the number of alphanumeric characters stored in a storage unit and the different ways in which characters are handled in Fortran. This applies particularly to more modern programs with user-oriented control statements. Given the lack of character-string handling in the Standard, programmers have developed different conventions for coping with characters and two essentially machine-dependent attributes, the number of characters per word and the internal value of the characters, are often assumed and their use may be deeply embedded in the coding.

A further omission from the scope was the rules for interpretation when a condition was violated. In some circumstances, such as the index of a computed GO TO being out of range, it could be argued that a program should be terminated but such a rule would clearly be inappropriate to the evaluation of a mixed-mode expression which a compiler supported but which was invalid according to the Standard. Again, the Standards Committee took the most sensible, and most general course, but it is one which has not satisfied everyone.

Finally, the omission of size limits for a program is inevitable and is common to most languages because so many factors relating to the processor and the operating environment are involved. Since it is not good practice to write massive subprograms few difficulties occur at compilation time but there are inevitably some problems at execution time with systems with small storage areas or with poor overlay systems.

### 1.7.2 Folk-lore

At a time when the IBM 7090 implementation was dominant, a number of conventions which were outside the Standard and were rarely documented, came to be implemented and assumed by programmers, both on IBM and on other systems. These included such things as the retention of local values in subroutines on execution of a RETURN statement, the use of 1 for the right-most dimension of an array passed as an argument to a subroutine, the ability to distinguish a blank and a punched zero data input field by using the SIGN function and the copying and comparison of characters in arithmatic and logical variables. Many of these are still assumed by most programs and programmers and Böck (1975) makes the excellent point that compiler writers who observe the letter of the Standard, but who are unaware of common practice may produce compilers of no use whatever for portability purposes.

### 1.7.3 Observance of the Standard

As discussed in section 1.6, observance of the Standard nowadays by compiler writers is on the whole good. The author is not aware of any compiler which complies fully with the Standard but usually the differences are so small it is difficult to understand why they should exist. The compilers which cause problems are not those which deliberately ignore the Standard, if indeed there are any such, but those which adhere to the written, but not to the unwritten rules of Fortran.

Observance of the Standard by programmers is unusual for the reasons given above. There is even an unfortunate tendency for programmers to think that because they are using a compiler which is stated to be standard-conforming that they are necessarily writing in standard code. It may be argued that it is good discipline to keep to standards but it is only necessary when a program is required to be run on more than one processor and even then, if the programmer knows which processors are involved he may be able to use the common subset of statements which are available on most large systems. These include such things as mixed mode expressions, ENTRY, IMPLICIT and direct-access input and output. A summary of common extensions is given by Muxworthy (1970). Most of these extensions are to be found in the draft proposed Standard of 1976 which implies that they were considered to have met the needs of users and to be in the Fortran style.

## 1.8 The Draft Proposed Standard for Fortran (1976)

In revising the Fortran Standard the Fortran Committee of ANSI, X3J3, have used the following criteria in deciding on changes:

- interchangeability of programs between processors,
- compatibility with existing practice and with existing standards,
- consistency and simplicity for users,

- efficiency of operation,
- allowance for future growth.

They have again produced two language levels, a full and a subset level. The latter was originally intended to be comparable in power to the ISO intermediate level, but it contains significant additions. It exists so that smaller computers may implement much of the new Standard without carrying the overhead necessary to cope with all the statements. The following description refers mainly to the higher level.

The committee's insistence on maintaining very high compatibility with the 1966 Standard in the sense that a program which conforms to the old Standard should conform to the new one, has alone added years to the committee's work and means that the essentials of Fortran are retained. In particular there is no change to the layout of statements, to the allocation of data storage or to the structuring of code, although there is a possibility that the selection mechanism (IF-THEN-ELSE) may appear in the final Standard.

The most notable incompatibility with 1966 Standard is the omission of type Hollerith (characters stored in reals, integers, etc.) although it seems likely that processors will continue to accept Holleriths; a recommendation for the form of their implementation is in the draft. Other incompatibilities are relatively trivial and are listed in section 19 of the document.

The draft incorporates many of the commonly implemented extensions such as 7-dimensional arrays, mixed-mode expressions, generalized subscripts, direct-access file-handling, free-format input-output, alternate returns, ENTRY and IMPLICIT, but not necessarily with existing syntax. It also includes some well-established extensions which are not on the majority of systems, such as the PARAMETER statement and the "zero-trip DO-loop", and some extensions which are not in generally available systems, such as the SAVE statement, giving an "own variable" and "own common block" facility and several new input-output and format facilities. The principal single extension is type CHARACTER which, so far as is logically possible, has all the attributes of a Fortran data type. The syntax is new and has been designed by the standards committee, based on that of existing implementations. Characters are stored in "character storage units" and arithmetic and logical entities in "storage units" and, to improve machine independence, great care has been taken to keep these separate. For example common blocks may contain character variables or arithmetic and logical variables but not both, characters may be equivalenced only to characters and so on. Unfortunately this separation has broken down in one place in that the length of an unformatted direct-access file record is measured in non-character storage units and characters may be written to such a record. In this case it may be necessary for the programmer to

know how many character storage units correspond to one non-character storage unit.

In general the language has been made more regular. Where before a variable was required an expression is now allowed and where an arithmetic entity of a particular type was required, an integer, real or double precision is allowed. The only major available extensions which are not in the proposal are varying length data types (INTEGER*2 etc.), NAMELIST and DOUBLE PRECISION COMPLEX.

A fuller summary of the draft Standard is given by Muxworthy (1976) and a more comprehensive one by Meissner (1976a).

## 1.9 Discussion of the Draft Standard

### 1.9.1 Development Phase

It was assumed at first, that is in 1970, that the revision of the standard would simply formalize some of then existing extensions. This cautious approach is reflected in the recommendations of a Working Party of the British Computing Society Fortran Specialist Group (1971a) to the American Standard Fortran Committee. When it was found that X3J3 had already approved in principle uncommon extensions like type CHARACTER and multiple assignment statements and was considering such unlikely proposals as dynamic storage allocation and a negative logical unit number implying backwards input-output, the reaction was one of pleasant surprise and the subsequent BCS report (1971b) was much more liberal. The latter two proposals were of course rejected and multiple assignment statements were subsequently removed. During the course of the revision a number of proposals were accepted which were later rejected, one of the most notable being array cross-sections and array arithmetic. Substantial changes, in some cases on several occasions, were made to the syntax of other new statements, especially the CHARACTER statements.

Throughout the development phase close contact was maintained between X3J3 and the Fortran Committee of ECMA and the BCS Fortran Group. Apart from the Boston Fortran Group there does not appear to have been a similar interest group in North America, despite attempts to generate discussion like those of Thorlin (1972) and Engel (1974), until the formation in 1975 of the Ad-Hoc Committee on Fortran Development which later came under the aegis of ACM-SIGPLAN. This committee's bimonthly newsletter FOR-WORD has carried a continuing discussion of X3J3 developments as has that of the BCS Group throughout the six years.

### 1.9.2 Content

The scope of attributes defined by the draft Standard is unchanged from that of the 1966 Standard. Nevertheless the proposals should have a bene-

ficial effect on the transferability of programs firstly because the increase of regularity, the general easing of restrictions and the various new facilities will make programs easier to write and should eliminate some of the present differences between implementations, and secondly because one of the main obstacles to transferability, the difference in the numbers of characters per word, may be made irrelevant in the majority of programs by the use of type CHARACTER.

Some transferability problems may remain with type CHARACTER because no minimum upper limit on the length of strings has been defined, leaving it possible for implementors to allow only single character strings, but it is more likely that problems could arise because no complete collating sequence has been defined. The characters A to Z are defined to be in ascending order as are 0 to 9, and blank is less than A and less than 0 but no other relationships are specified. ASCII code is recommended but is not mandatory so that 'A'.LT.'1' may be true as in EBCDIC or false as in ISO/ASCII and still be standard-conforming. Good programming techniques should be able to resolve portability problems in this area.

The committee did not accept proposals to include environmental enquiry functions in the language which would have facilitated the writing and interchange of numerical programs. It is of course possible for programmers to define such variables as are necessary in DATA statements but their provision as intrinsic functions would make authors more aware of them and increase their usage.

The committee also resisted proposed changes to the form of a Fortran source statement, considering it the function of a preprocessor, not a compiler, to accept free-format source. They took a similar attitude towards macros and accepted only the PARAMETER statement which is equivalent to EQU in most Assemblers and hence supplies a small but useful macro facility. The provision of the IMPLICIT statement and of generic functions removes the need for one of the uses to which macroprocessors are put.

The major possible change which the committee resisted was the introduction of structured coding. It seems likely that it was rejected because of fears that it would necessarily involve the dynamic allocation of data storage but during the time the committee has been working it has been overtaken by a flood of over 60 structured Fortran preprocessors (see section 2.6.2.3 and Appendix A) which have demonstrated not only that structured code and static storage may of course coexist but that structured Fortran is accepted by and appears to be beneficial for Fortran programmers. It is possible that the selection mechanism may yet appear in the next Standard due to a late change of opinion by the committee.

On the positive side, the addition of only two characters to the Fortran character set, apostrophe and colon, should aid portability and keep Fortran clear of some of the problems which have beset other languages. The greater than and less than characters were in the working proposals at one stage but not in the final document. Such trivial extensions as the ability to put comment lines immediately before continuation lines, the ability to specify an array name alone in DATA and EQUIVALENCE statements and the ability to have constants in output lists, all make for code which is more easily written and read.

The introduction of the zero-trip DO-loop, that is when the number of iterations of a loop as defined by the parameters in the DO-statement is less than one the body of the loop is not executed at all, may cause some problems at first, especially for those who incorrectly assume that the body of a DO-llop is always executed at least once, but this facility together with the fact that the DO-parameters may be expressions of type integer, real or double precision taking any value (positive, negative or zero) should make unnecessary the dummy assignment and IF statements which often clutter the entry to a loop. Similarly the introduction of non-unity lower bounds for arrays should eliminate the need for some of the artificial devices now used.

Another area which could cause minor problems at first is the reclassification of the basic external functions as intrinsic functions, the addition of new intrinsic functions and the replacement of the EXTERNAL by the INTRINSIC statement for these functions.

The SAVE statement is to be welcomed as it allows the old problem of the retention of local values in subprograms to be resolved by statements within the Fortran language. It also allows an overlay structure to be defined implicitly from within Fortran and although overlay might be thought by some to be obsolescent, it will be required for some years yet.

It is the area of input-output which has been developed the most. There are free-format (stream) and direct-access files and a new concept, character files. A character file is simply a character variable or array within a program which may be read or written under format control, thus giving character-internal representation conversion and vice versa. The possible forms in the format position of an input-output statement are extended to include a character expression, including a constant,and an integer variable to which a format label has been assigned. There are additions to the format codes themselves and OPEN and CLOSE statements for files are introduced. These statements have an optional reference to a file-name and are therefore potentially system-dependent; attention is drawn to this in the draft. Much criticism has been made of the INQUIRE statement which returns information on various attributes of a file and which uses a syntax which goes against Fortran, and most other language, conventions. A detailed criticism of INQUIRE is included in the review of

the draft Standard by Day et al. (1976).

The X3J3 committee have shown concern for compatibility by listing all known conflicts with the 1966 standard (section 19.1 of the draft Standard) and for portability by listing standard items which could inhibit portability (section 19.2). The latter concerns non-Fortran procedures, character collating sequence, non-standard characters, file names and input-output unit numbers and capabilities. The committee also suggest that producers of processors should provide some means of identifying the nonstandard syntax supported by their processors.

In summary, the X3J3 committee have shown the right priorities (compatibility, portability, acceptability, efficiency) throughout their work; some of their proposals are arguable and one or two are almost universally disliked, but the vast majority are to be welcomed. It is to be hoped that suppliers implement them in the same spirit.

# 2. PORTABILITY

## 2.1 Summary

This section is concerned with the portability of applications software in the scientific and technical areas. After discussing some aspects of Fortran, including the use of Fortran for portable programs, there is an overview of publications on portability and of software tools directly and indirectly used to aid portability.
After a brief consideration of multi-machine programming, there is a discussion and a conclusion.

## 2.2 Scope of Portable Software

The term "portable software" takes quite different meanings in different contexts. For example in "Macroprocessors and techniques for portable software" (Brown, 1974a) the author is concerned mainly with the transferability between computer systems of assemblers, or assembly language processors; in "Hints on distributing portable software" (Waite, 1975) the main concern is the mechanics (media, character sets, etc.) of moving software between computers without paying attention to the content of the programs themselves. Here the term is taken to mean programs, typically application programs, written in a higher-level language, which may without change be processed by different computer systems. The main discussion involves the design and production of such software although some mention is made of the transferability problems in the sense of Waite. Although the concepts are valid for most fields of application and for most computer languages, the discussion concentrates on the so-called scientific and technical fields.

## 2.3 The Trend towards Fortran

Rosen (1961) mentions that the ability of the Philco 2000 ALTAC compiler to process IBM 704 Fortran II programs provided a compiler, for the first time, with the power to assume the major burden of transition from one computer to another. At about the same time COBOL programs began to be run, with only minor changes, on two or more systems. As mentioned in section 1.3, the main concern of the American Standards Committee for Fortran, meeting for the first time in 1962, was to promote a high degree of interchangeability of Fortran programs for use on a variety of systems. Thus the concept of Fortran as a language for writing portable software was established in the early 1960's.

The need to move large numbers of Fortran II programs to Fortran IV systems led to the development of a number of source-to-source translation programs, the best known of which was SIFT - the SHARE Internal Fortran Translator (Allen et al., 1963). This was itself written largely in Fortran and demonstrated not only the viability of such translators, but

the viability of writing them in Fortran. Another translator from this period ALTRAN (Olsen, 1965) converted programs from ALTAC (an extended Fortran II) to IBM Fortran II.

On the third generation of computers Fortran has become established as the principal language used for scientific and technical computing and virtually the only such language used for writing programs intended to be run on more than one computer. Its only serious rival has been Algol 60 which, notwithstanding its power and elegance and its machine independent design, has in practice suffered from the same hindrances to portability as Fortran and in addition has had its notorious problems with character sets and input-output conventions.
The only other potential rival, PL/I, despite its original informal name Fortran VI and despite its vigorous promotion by IBM in the late 1960's and early 1970's, has failed to supplant both Fortran and Cobol on IBM systems, as it was intended to do, and, in contrast to Fortran, it has been taken up only slowly by other manufacturers.

Thus Fortran, which was not intended to be a machine-independent language, has attained a position which its designers could hardly have imagined more than 20 years ago. It is not the intention here to discuss in more detail why this situation has arisen or whether a more widely accepted position could have been reached. Böck (1975) states the pragmatist's view, Bemer (1969) records the decline and fall of Algol 60 in the U.S., Hoare (1973) discusses some aspects of the main three (or four) languages and Ralston (1973) and McCracken (1973a) discuss trends in the teaching of languages.

The literature of portable software almost invariably assumes that Fortran is the language being used and this also is assumed below.

## 2.4 Programming Practice in Fortran

Although a number of studies have been made of the speed and facility with which programmers design, write and test out programs under certain experimental conditions, e.g. on-line/off-line, structured/unstructured etc., the chief conclusions of which appear to be that programmers are different from each other (cf. Sackman et al., 1968), surprisingly little had been published about statement usage in real Fortran programs until the classic paper of Knuth (1971), (Wichmann (1973) did similar work on Algol at about the same time). Moulton and Muller (1967) had included some Fortran statement statistics as a minor feature of their paper on a diagnostic compiler but it would appear for instance that much of the work on optimizing object code from compilers had gone on without any knowledge of the constructions programmers actually used and hence without sufficient information to decide whether a particular optimization was economically justifiable.

Knuth and his team of helpers made static and dynamic analyses of a large number of Fortran programs gathered from various sources at Stanford and from the Lockheed Missiles and Space Corporation; they also made a detailed study of seventeen of the programs. One of the most striking facts to emerge was that almost 50% (statically) of statements were assignment statements and of these no fewer than 68% were simple replacements, with no arithmetic operators, and a further 18% contained only a + or - sign. The paper makes a strong case for the use of statement frequency counts as a tool for improving the performance of programs and, even more basically, as a tool to allow the programmer to understand more fully what he has written. Knuth's paper had a number of indirect effects too: it gave impetus to the analysis of Fortran programs and hence to interest in programming aids and it turned attention to the practical needs of users of higher-level languages. All these have implications for portability; the better understood a program, the more likely it is to be transferable.

Robinson and Torsun (1976) repeated Knuth's static analysis for a smaller sample of programs and obtained broadly similar results and Kulsrud (1974) compiled some statistics on the various classes of problem for which compilers were used. Another observational study of programs which was carried out at NAG (Numerical Algorithms Group, Oxford) is discussed below (section 2.7.3).

## 2.5 Literature of Portability

### 2.5.1 General Papers

The literature of portability in the sense assumed here is small and recent but is growing. Past neglect and current interest are due to the rising cost of software relative to the cost of hardware. In 1967 Naur reviewed some of the problems but he was concerned with such things as data formats, data capacity and types of backing storage, items of lesser interest today. Naur also made a case for environmental enquiry functions (cf. section 2.6.3.2).

Of the general review papers Muxworthy (1972b) identifies some of the main problems and Böck (1975) makes a case for Fortran, warns that portability usually implies inefficiency and tilts very effectively at several windmills. Not having had the opportunity to see the work of Brown (1969) or Fries (1975), by far the most comprehensive document the author has read is that by Dahlstrand (1976) which is itself only an interim report.

Dahlstrand has been carrying out a preliminary investigation on the possibilities of running programs in an unchanged state on mutually different systems. Should the feasibility study prove positive, it is intended to undertake a larger project to design and implement a complete portable language system. The sources of imcompatibility, some of which are outside

the immediate scope of this report, are identified as:

(a) differences in memory structure and capacity (disks, drums, etc.),

(b) differences in the form of input-output (character sets, symbol representation in Algol),

(c) differences in wordlength,

(d) differences in the implementation of languages (dialects),

(e) differences in operating systems and command languages.

These differences are said to have caused the following problems, some of which were inevitable but many of which have been brought on needlessly:

(a) the unnecessary writing of programs which already exist at other installations,

(b) the work of transferring programs when an installation changes its machine,

(c) difficulties with evening out the load in a concern with several installations,

(d) serious hindrances to network development.

These incompatibilities are estimated to cost, in Sweden, a minimum of 3-4 million Kronor per year.
The difficulties of incompatibility may be overcome in two ways: to define a truly independent programming language or to limit oneself to those parts of existing languages which are machine-independent. The latter is rejected because it does not solve the basic problem and has a major weakness: in using what is common to all machines neither hardware nor software is used to full advantage.

Dahlstrand thus proposes a logical solution. He cites the ALMO project in the Soviet Union which has produced Algol compilers for the three different machines at Novosibirsk University which are said to appear to be so similar that the user does not need to know on which machine a job will be run. ALMO (Kamynin and Lyubimskiy, 1967; Luchovitskaya, 1974; Yershov, 1971, describes related work in a more accessible reference) uses the UNCOL (machine-independent intermediate language) concepts of the late 1950's; ALMO has also produced compilers for Fortran and ALPHA. Dahlstrand proposes adding precision statements, bit-handling and character-handling statements to Fortran and building these into a complete subsystem which will have its own command language and its own Algol compiler, also compiled to an UNCOL. Basic and APL would be added later but Cobol and PL/I would require less attention because they are already defined in a machine-independent manner. With such a system installed on major machines in Sweden, it would be possible to transfer programs with a minimum of effort. Rather surprisingly Dahl-

strand allows that local compilers could still be used for local running.

The goal for full portability is summarized thus: a complete language system, i.e. a system containing both programming language and command language should be defined. Properties which are important for an algorithm, e.g. precision, should be defined in the framework of the language. A program written solely within these language rules should, on execution, either give the correct result within the latitude accepted because of the defined precision etc., or be cancelled because the program reaches some implementation limit. A program which breaks one of the language system rules should be cancelled when an attempt is made to use it. The diagnostics ought in both cases to be given if possible at compilation time.

Dahlstrand would go so far as having error actions for this subsystem identical across machines, implying that additional interrupts would be needed in some cases. He says that if the stage is ever reached where a program can be run on absolutely any machine, no one will suggest changing that situation. This might be true for users but it is not so certain that manufacturers would agree.

### 2.5.2 More Specialized Papers

It is interesting that Dahlstrand devotes considerable space to problems of character sets. In English-speaking countries the attitude is often that these cause little difficulty provided a one-to-one correspondence exists. In some European languages however, extra letters (five in Swedish) are needed to be able to print personal names and addresses correctly and these have to be taken from the special characters often already used for specific purposes by the manufacturer. A further complication arises when different manufacturers ignore standards and allocate national characters to varying special characters. Zeckendorf (1973) also gives a comprehensive description of the problems involved.

Distribution of software is described by Mongini-Tamagnini and Gaggero (1974) and Waite (1975) and the many recent papers which describe some aspect of the master source or composite file concept include Krogh (1972) and Ford et al. (1974); other master source references are in section 2.7.3.

Papers describing particular software tools are referenced in their proper place below but the one on PFORT (Ryder, 1974), a standard Fortran syntax checker, caused such interest that it should be mentioned in an overview. PFortran (Whitten and deMaine, 1975) - an unfortunate clash of names - is an attempt at defining an extended, portable Fortran which works by translation to ordinary Fortran and uses run-time utility functions. Papers advocating particular programming practices are also described at the appropriate section below.

Traub (1971) in a relatively early paper expounds the benefits of writing in "intersect Fortran" but curiously, in the context of numerical software, does not mention precision differences. Intermachine comparisons for Algol 60 have tended to concentrate on implementation techniques and benchwork speeds (v. Wichmann (1973) and the numerous references given therein). Brown et al. (1971) make some remarks on the portability of Algol programs and van de Riet (1973) describes a portable Algol compiler which appears to satisfy some of the demands of Dahlstrand at the expense of run-time efficiency.

Portability is here considered to be an essentially practical problem and those papers discussing it in terms of abstract machines, advocating translation at the assembler language level, and other chimera are omitted.

### 2.5.3 A Caution

The 1966 American Fortran Standard very carefully refrains from using the word "compiler" and always uses the word "processor". This leaves the way open for implementors to employ such hardware and software techniques, e.g. compiling, interpreting, etc., as they wish (section B 1 of the Standard). The 1976 proposed Standard retains the same emphasis and suggests that a processor could also be a human with paper and pencil (section 20.1 of the proposal). Fortran for interchange is defined only at the level of the coding form; no mechanical representation of a program is defined.

Similarly the 1966 Standard (section 4) puts little restriction on representation of arithmetic quantities. Integers must be exact representations, real data are approximations to a real number and double precision data are approximations with a greater degree of approximation than reals. These concepts are retained in the 1976 draft (section 4). At no point is a binary digit mentioned. Yet many papers discussing Fortran assume aspects of implementations with which their authors are familiar, for instance that positive integers are stored as a right-justified string of bits representing digits in a binary number. This is not justified: ICL and Telefunken machines have stored Fortran integers as floating point numbers and Fortran has been implemented on several machines with decimal arithmetic.

It is helpful when considering some aspects of Fortran to remember that a human with a calculator is a valid Fortran processor, just as it is helpful when considering transferring programs to remember that incompatibilities between different compilers on the same computing system can cause as much difficulty as differences between computers.

## 2.6 Approaches to Portability

### 2.6.1 Introduction

Since the simplest of programming language statements, while standard-

conforming syntactically, cannot be guaranteed to be standard-conforming on execution (cf. section 1.7.1), it is impossible to state unequivocally that a particular program, per se, is portable. Given a program, its data, the environment in which they will run and the history of any similar runs, it is possible to make a reasonable prediction as to whether the run will be "successful" but the demands made by a program on its environment, the support provided by the environment and their interaction are so complex that it is not possible, or practicable, to document them in such a way that a certain prediction can be made. Indeed at the present state of knowledge of program validation and program proving few would attempt to make certain predictions about the running of any given program with all possible data within a single environment. In the domain of real machines and real programs therefore rigorous portability, that is universal guaranteed portability, is unachievable.

Dahlstrand (section 2.5.1) sees the solution as the standardization of language processors and the standardization of environments. This is at best a long term solution and is subject to the usual criticism of standardization, especially over such a large area - that it is the enemy of progress. Most workers have taken the opposite course and have endeavoured to produce programs which will be processed in as similar manner as possible on different systems. Effort has been concentrated on methods of ensuring this, of investigating the nature of the programs and of documenting them so that their external needs (processors, run-time systems, numerical precision, etc.) are known as fully as possible. In a perfect world much of this would not be necessary, but in practice it is common for programs to go undocumented, for programs to be maintained by others than their authors, for programs written for one computer to be pressed into service on another, and so on. In any case Knuth (1971) made clear that even as programmers were writing, they were ignorant of some most important aspects of their programs such as the relative efficiency of various sequences.

The following three sub-sections discuss some of the ways advocated for the individual programmer to improve portability, directly or indirectly, (a) improved programming practice, (b) defining the environment of the program, and (c) software tools. Documentation standards are so far as possible separated from programming conventions and are covered in section 3.1. The centralized viewpoint, that of the software distributor, is dealt with in section 2.7.

### 2.6.2 Improved Programming Practice

#### 2.6.2.1 Introduction

Advocates of improved programming practice tend to concentrate on one aspect of coding. One of the major divisions is between those discussing programming in the present Fortran context and those proposing struc-

tured programming techniques; the latter is considered in section 2.6.2.3. Within the present context there are many different emphases; most have some concern with transferability but one of them argues that legibility should be paramount and specifically recommends the use of statements such as

DO 12 X=1.2, G+9.1, B/2.0

if available.

2.6.2.2 The Present Fortran Context

Apart from their advocacy of legibility at the expense of portability and their unusual but useful suggestion that the last digit of a label be a count of the number of GO TO's referencing that label, McCracken and Weinberg (1972) give a list of suggestions on what is now generally accepted to be good coding practice, e.g. liberal comment cards, few GO TO's, DO-loops end on CONTINUE's, indentation, liberal parentheses especially in relational expressions, meaningful names, variables not used in different contexts, increasing order of labels and so on. This paper is typical of most such sets of guidelines and many installations make similar recommendations to their programmers; they have the advantage that if followed they make for more easily read, and hence maintained, programs without placing any serious restrictions on the author's style.

The same concern for legibility without restricting style is expressed by Banks, Percival and Wilson (1972a) in their set of documentation conventions called FORDOC1; these are not simply documentation conventions as they affect the form of the program. They are in effect a long checklist of what items should be commented and in what order and what manner and they lay down detailed conventions for orders of subprograms within a program, orders of statements within a subprogram, indentation and spacing within a comment and within a statement, classes of variable name, statement numbering etc.,etc. They were conceived in the context of the need to communicate subroutine packages between physicists and, as demonstrated by the authors (Banks et al., 1972b) they make for a very readable program; a program consisting of 90% comments should be readable. The whole has been done with admirable thoroughness and it is unfortunate that so much work has gone into something which flits around the problems of machine-dependence without attempting to attack them. More attention is paid to whether an equals sign should have spaces around it than to whether the numerical precision used by a routine should be declared in a comment and variations in language dialects are covered in two sentences. Most important, the conventions require more motivation for their use than is possessed by most programmers.

The work of Banks et al. was stimulated by that of Roberts (1969). Roberts argued for the establishment of an international literature of published scientific programs so that information on programs could be published

as quickly and efficiently as that on other research projects. To this end he suggested a few coding conventions and mentioned without giving detailed references that some aspects of program documentation could be automated. The coding conventions include a number of eminently sensible suggestions (use variables for logical units and limits, segregate input-output, do not use internal character values, provide a Fortran equivalent of any non-Fortran routine, etc.) and only the suggestion for variable naming conventions could be regarded as inhibiting. These principles were built into OLYMPUS (Roberts, 1974; Christiansen and Roberts, 1974; Hughes, Roberts and Roberts, 1975; Hughes, Roberts and Lister, 1975) which is a partially automated system for writing, storing and running programs. This interesting system was developed for codes to solve partial-differential equations of a particular form but was found to have wider applicability; it consists of a set of utilities and standard subprograms and even has a fixed skeleton of subroutines for a program. It is claimed that once a programmer has learned all the rules, he can develop programs more quickly partly because he is relieved from the need to make ad-hoc decisions and partly because the code is so formalized it is easier to understand other peoples' programming. It remains to be seen to what extent a typical Fortran program can be forced into a rigid subprogram framework and to what extent programmers can accept the discipline involved.

Publication of programs can be a strong motivating force for adopting particular conventions (cf. Hill et al., 1975) and the rules laid down for algorithms printed in journals can offer good guidelines for coding. When these are backed up by strong editing and refereeing, as for example in the case of Applied Statistics (Working Party on Statistical Computing, 1975), the resulting algorithms achieve a high degree of portability.

The conventions above aim at transferability, primarily through increased legibility and documentation of the code. McCormick (1974) reviews and evaluates the literature on detailed coding suggestions for making programs more portable. Some of the published suggestions have been far too restrictive (e.g. avoidance of DATA, labelled COMMON, LOGICAL) and McCormick says so, but he advocates large machine common subset, so-called "intersect Fortran" programming rather than strict Standard-adherence. His paper contains a detailed, but not exhaustive, list of good coding advice. He also makes the point that efficient coding techniques for non-virtual systems may be inefficient on virtual systems and vice versa.

The sentiment is often expressed in the literature that the American, or sometimes the ISO, Standard should be followed by programmers (and occasionally that it should not be). This ignores the fact that the Standard is a cryptic document which deliberately left certain sections open to different interpretation and other passages which were thought to be sufficiently explicit were found not to be. It has been described as more a document for

a lawyer than for a programmer. As the experience with the Standard and with implementations has grown, and following the publication of two sets of clarifications not often referenced when the Standard is mentioned, the scope for theological debates on the interpretation of a particular text have been reduced but not yet eliminated. A manual intended to show "programmers how to write Standard Fortran programs and to bridge the gap between the few who understand Standard Fortran and the many who do not" was published but was flawed by typographic errors. Even the second edition (NCC, 1972a) still contained some printing errors and a few errors of fact and insufficient distinction was made between fact and opinion. Programmers following the advice in this manual would produce programs not far from Standard-conforming but they would gain little insight into why they were writing as they were. For this they would have to read Larmouth (1973). Larmouth's excellent paper discusses both programs conforming to the Standard and the use of non-standard features.

It is noticeable that both these major works of interpretation of the American Standard emanated from Europe and it is interesting, but profitless, to speculate why this should be. As noted above, compiler manuals generally give little help to the programmer to decide whether a program is standard-conforming; they usually claim that the compiler is standard-conforming and give in an appendix a list of extensions, and, less often, restrictions. At about the same time as the NCC manual first appeared IBM redesigned their Fortran language manual to indicate clearly exactly which of their facilities are non-standard at the place they are described. This is an excellent system and it is unfortunate both that it is not more widely adopted and that the IBM software itself does not have a corresponding feature.

Lastly on dialects there exist a number of in-house standards which recommend programming conventions within companies (e.g. Shell Standard Fortran of 1971 - the content of which was confidential) or within other installations (e.g. Friedrich, 1975).

Differences in wordlength have received relatively little attention in discussions of portability. The modification of programs which assume a particular number of Hollerith characters per word seems to be accepted as a tedious but inevitable chore if efficiency is to be maintained. The question of numerical precision commonly devolves, possibly after a serious analysis, onto a simple choice between single and double precision arithmetic. Actively programming for a particular precision is mentioned in section 2.6.3.2; Schonfelder (1976) gives a review of the problems involved.

### 2.6.2.3 Structured Programming in Fortran

Interest in structured programming in Fortran has grown tremendously in the past two years. Although Algol 60 had structured programming facilities it was not until 1966 that Böhm and Jacopini proved that it was possible

to express any required program with only three control.structures: sequence, selection (IF-THEN-ELSE) and iteration (DO-WHILE or DO-UNTIL), and that given these the GO TO was unnecessary. Other mechanisms, especially SELECT-CASE, have since been shown to be desirable to improve efficiency and ease of expression. In 1968 Dijkstra in a letter given the title "GO TO statement considered harmful" caused considerable debate and not a little puzzlement but on the whole programmers did not allow it to affect their coding styles. Then stories began to leak out about IBM's project for the New York Times in which they used both their chief programmer team system - a formalized scheme of inter-programmer communication for designing, writing and debugging programs (Baker, 1972; Baker and Mills, 1973) and structured programming, the whole being known as IPT - Improved Programming Techniques -, which had been completed ahead of schedule and which had achieved unheard-of programming error rates; one error per 10 000 lines of code was mentioned. Finally, Datamation gave special attention to structured programming with five expository articles in its issue of December 1973 (McCracken, Donaldson, Miller and Lindamood, Baker and Mills, and Clark, the last having a different emphasis from the others). Now that IBM had been seen to have embraced structured programming it became a topic for general discussion, just as virtual operating systems had done 18 months earlier. The journal Computing Surveys followed up with a special issue on programming (December 1974) which was largely devoted to structured programming.

There were immediate demands that structured facilities be incorporated in the revised American Standard for Fortran and from 1975 onward there have been numerous articles and letters discussing whether and how Fortran should be structured in the pages of Computer, Datamation, FORWORD, SIGPLAN Notices etc. (inter alia: Meissner 1975, 1976b; Horowitz, 1975; Reifer, 1976). So far the Standards Committee has refused to accept structured Fortran but many individuals have gone ahead and produced structured Fortran preprocessors which translate a program written in a structured Fortran (possibly an extended subset rather than an extended Fortran) to a form processable by their normal Fortran compiler. FORWORD (August 1975) listed 51 such preprocessors and Appendix A to this report lists these and a further 16; Horowitz (1975) contrasts the facilities of six of them.

It appears that not a few of these preprocessors were written as experiments rather than as real working tools and little experience on their use has been described. Fortunately two of the few European preprocessors, SHELTRAN (Croes and Deckers, 1975; Croes, 1975) and STRUFORT (Cardetta et al., 1975; Cardetta et al., 1976) have been used and reported.

SHELTRAN has taken a drastic attitude to GO TO-less programming: it does not allow any form of GO TO, or any statement labels other than those on format statements and it has also done away with the usual For-

tran IF-statements, the DO, the ASSIGN and the alternate RETURN. The constructs IF-THEN-ELSE, SELECT-CASE, WHILE-loop, loop-UNTIL and FOR-loop replace them, and a COBOL-like PERFORM-PROCEDURE has been added. Other Fortran statements are unchanged. SHELTRAN has been in production use within the Shell company since 1974 and has apparently met with great success. Writing in Fortran has virtually been eliminated and programmer productivity has gone up by 50%. Although SHELTRAN or similar preprocessors do not of themselves make for increased portability, their use does open up possibilities to incorporate standard-conformity checking into the preprocessors as an option. Another possibility being explored in Shell's case is to be able to vary the processor output, from a fixed input, according to the target computer for the Fortran.

To the user STRUFORT appears very similar to SHELTRAN, but the syntax is not identical. STRUFORT too has removed the need for using GO TO statements but still allows them.

Further aspects of preprocessors are discussed below (section 2.6.4.4.4). Structured Fortran usage is established in a few areas and it remains to be seen whether there will be any harmonization of syntaxes so that structured Fortran programs themselves become portable. At present the Fortran output from preprocessors is often as opaque to human eyes as the worst of direct Fortran programming and its transfer between installations should be avoided.

It would make an interesting study to investigate why Fortran programmers apparently take readily to structured Fortran when they have resisted other structured languages. Essential points must be that little new has to be learned, that none of the facilities and advantages of Fortran are lost and that some facilities are gained. In particular the structuring pertains only to the code, not to the data storage, so that programs with individual subprograms originally written in either structured or ordinary Fortran are possible.

It is not the intention here to attempt to review the literature of structured programming, which is very large, but attention should be drawn to the following: Dahl, Dijkstra and Hoare (1972), Knuth (1974), Wirth (1974), Rogers (1975) and Neely (1976). Knuth gives over a hundred references.

### 2.6.3 Defining the Environment

#### 2.6.3.1 Introduction

Rather than standardizing the environment of a program (section 2.5.1), an alternative approach would be to define the environment to the program, or to allow the program to request such information on the environment as it needs. Another alternative would be to select those areas of Fortran which are not consistent between machines and to provide parallel facilities by for example supplying utility functions which are tailored to the computer

but which present a standard interface to the program. This latter course is often taken for handling non-standard input-output devices, such as graph plotters, but only rarely for manipulative processing and the former is not used as much as would be expected.

2.6.3.2 Environmental Enquiries

Naur (1967) suggested using environmental constants as an aid to machine-independent programming. A rather limited number of environmental constants are recommended in the Algol 68 report (section 10.1 of van Wijngaarden et al., 1969); they include the values of maximum integer, maximum real, relative precision and allow for character-integer conversions. The Working Party on Fortran extensions (1971b) suggested also that the number of characters per word, the decimal accuracy of real numbers, and constants such as pi be provided by calls to a standard function.

A more comprehensive list is that by Ford (1976) which specifies several more values for arithmetic limits than the Algol 68 set and attempts to parameterize the representation of integer and real numbers and the type of arithmetic used. Further it seeks to specify the page size for virtual systems, the standard Fortran input-output units and other device attributes such as characters per line.

As mentioned above, such a set of values may easily be incorporated in a program in a DATA statement, or in a more uniform way for several programs by putting them in a common block defined in a BLOCK DATA subprogram, but their provision as standard functions with a description in a compiler manual would tend to increase awareness and usage.

2.6.3.3 Parallel Facilities

Algorithms to perform basic arithmetic to arbitrary precision are well described in the literature (cf. Hill, 1968; Dekker, 1971) but guaranteeing precision to programs by using external functions simply in order to make them portable, is usually considered to be inefficient and is not used although it is proposed by Whitten and deMaine (1975).

A more obvious field for this technique is character string handling. Although several string handling packages exist (e.g. Hertweck, 1970; Macleod, 1970) only a few (e.g. Reynolds, 1976) have been deliberately written for and successfully used on several different computers. In general it would appear that programmers are content to write within the confines of environments provided for their programs and not to be concerned with other possible environments their programs might encounter; even the names of such universally provided functions as those to find the date and time are not fixed.

## 2.6.4 Software Tools for Fortran Programs

### 2.6.4.1 Introduction

Tools which increase knowledge of a Fortran program are an aid to portability either directly or indirectly. This section describes software tools in the following classes: syntax and other checkers, analyzers, preprocessors, debugging aids, documentation aids, extensions to Fortran, translators, compiler testers. A checklist is shown in Appendix B. The groups are used for convenience only and some programs could have been allotted to one of several classes. Corresponding aids exist for other languages.

It should be noted that some of the individual programs mentioned may be limited to processing one dialect of Fortran; this of course does not affect the principle involved.

### 2.6.4.2 Syntax and Other Checkers

Syntax checker in this context is taken to imply checking that the syntax of a program conforms to a particular set of language statements. Baron, Schiffman and Fenves (1974) give an overview of some software tools in use at the University of Colorado and mention an ANSI Fortran syntax checker without giving any details. They state that this "unit of software" will flag non-ANSI code but that future versions would alter the input program to ANSI text. This is an extraordinary claim: if this could be done without altering the semantics of a program, all difficulties with dialects would vanish immediately.

A better known syntax checker, and the only other one found by the author, is PFORT (Ryder, 1974) which checks a program for conformity with a subset of ANSI Standard Fortran. Both individual program units and inter-unit communication, including use of common blocks, are checked. The language subset is sub-Standard only in a few areas: a common block may be initialized in at most one BLOCK DATA subprogram, a function may not change its arguments, the order of statements within a subprogram and the order of variables within a common block are more restricted, at most one character may be stored in a storage unit and characters may be stored only in integers, a common block name may not be also a variable name and extended ranges of DO-loops are not allowed. The program is itself written in this subset and hence is easily transferable. Ryder quotes a processing time of 20 lines per second, presumably on a 370/165, so the program is not particularly fast.

A number of compilers, often as an option, flag statements which do not syntactically conform to the Standard but few make the necessary checks at run time. One compiler (Rohl et al., 1975) claims to implement the Stan-

dard language only.

2.6.4.3 Analyzers

This section includes a number of different programs for analyzing Fortran programs either statically or dynamically or both. The most basic is the algorithm of Sale (1971) for identifying Fortran statements. The most common aids in this group are those which ultimately produce frequency counts for statements as a program is executed; references include Fosdick (1974), Paige and Benson (1974), Lyon and Stillman (1975) and frequency counts are one of the facilities offered by programs FUS, RXVP-1 and TAP (ICP Software Directory, July, 1975).
These aids typically involve the Fortran program to be measured having count statements inserted into it by means of a preprocessor which is often itself in Fortran. Shaw (1972) has shown that when the preprocessor is in SNOBOL it need be no longer than 200 statements. A variant on frequency counting is program sequence timing, provided for example by FUS.

Both RXVP-1 and STAN (ICP, 1975) claim to perform extensive (structural, syntactical and semantic) static analyses oriented to improving the quality and reliability of Fortran programs. From its summary description it is impossible to tell what RXVP-1 does. STAN makes a number of inter-subprogram checks which a compiler cannot do and it reports on possible use of a variable before it has been assigned; a number of intra-subprogram tests and reports are also made. Neither program summary mentions the ANSI Standard.

One of the most basic analyses that is required is a cross-reference listing of variables, showing in which statements they are used, and of labels, showing where they are defined and where referenced. Some compilers provide this information as an option and several utilities are available to perform the same function, e.g. FORTREF, STAN, XREFIV (ICP, 1975); see also section 2.6.4.6. This type of information is especially useful for independent validation or maintenance as well as for debugging a program. The same qualification could be made for path analysis, which involves the static analysis of all possible paths through a program and examines variable usage along each path. It is immediately seen for example if a variable is defined and never used or if one is defined and then redefined before use - both instances possibly indicating logical weaknesses. A program for path analysis, DAVE, is described by Fosdick and Osterweil (1976); some preliminary work is in Fosdick and Osterweil (1974).

The intra-subprogram structure of a program is analyzed by the Fortran-to-Fortran optimizing compiler of Schneck and Angel (1973). The analysis is of course an essential preliminary to code optimization. This feature too could be a useful aid to validation or maintenance but the prime objective of the program puts it more logically in section 2.6.4.4.2 where it is discussed in more detail.

The inter-subprogram structure of a program is analyzed by a program of Murison (1974) which, knowing the calls made by each subprogram and the subprograms' lengths, can produce a variety of possible overlay structures to meet criteria specified by the user. The analysis part is an independent program with input in character form. The user will typically wish to use a preliminary program to extract subprogram calls and lengths from an existing load module; such a program exists for the IBM 360/370. There is of course no restriction on the language in which the various subprograms are written.

Attention is also drawn to the work of Reifer (1975) which describes twenty automated tools for reliability and that of Ramamoorthy and Ho (1974) and Ramamoorthy, Cheung and Kim (1974). These are samples of the growing number of papers on the related subject of program reliability.

### 2.6.4.4 Preprocessors

#### 2.6.4.4.1 Introduction

Two groups of general purpose preprocessors are readily identifiable - macroprocessors and structured Fortran preprocessors; the remainder defy classification. There exist a number of well-known packages, for example in the fields of simulation and of civil engineering, which involve the translation of an extended Fortran to Fortran as part of their operation but as they are oriented towards a particular application and the Fortran they produce is rarely the concern of the user, they are not considered further.

#### 2.6.4.4.2 Miscellaneous Preprocessors

The Fortran-to-Fortran optimizing compiler of Schneck and Angel (1973) makes optimizations which are familiar in the context of conventional Fortran compilation such as elimination of dead variables, of common subexpressions and of redundant variable definitions, removal from DO-loops of fixed expressions, replacement of division by multiplication and so on. It thus provides certain automatic optimization facilities independent of any system-bound compiler, but more importantly it analyzes the structure and flow within each program unit and it can rearrange the statements to match this; this can be a powerful tool for investigating a program. Inevitably the new variables generated by the program have meaningless names and if the output is to be retained it may be preferable to replace them for example by using a context editor or by using one of the utilities described in section 2.6.4.6.

Two preprocessors in the literature are concerned with the layout of variables in large programs. Winske (1975) describes a preprocessor to dimension arrays and Ghan (1971) discusses CEB (Common Equivalence Builder) which will pack variables nominated in command statements into

common blocks and produce the appropriate equivalence and dimension statements and a variable dictionary. Papakonstantinou (1975) discusses a preprocessor for recursive Fortran.

Warburton (1976) has a preprocessor SHORTRAN, intended for use at keyboard terminals, which accepts free form input with all Fortran keywords replaced by one or two letters and which sets up Fortran statements in the standard form.

A preprocessor specifically intended to aid portability is that which will form part of the PFortran system; the distinguishing characteristic of this system is the use of run-time utility functions and it is therefore described in section 2.6.4.7.

2.6.4.4.3 Macroprocessors

The use of macros is well established in assembler languages but macroprocessors for Fortran, while well used in certain areas such as tailoring a program for a particular environment, seem not to have attracted the attention of the average Fortran programmer.

There are now also available completely language-independent macroprocessors and those intended for other languages, e.g. the IBM 370 PL/I preprocessor, have been used for Fortran. Macroprocessors have obvious applications for portability as a suitably written macro program can, within limits, be designed to generate a required numerical precision for a given computer, a required number of characters per word, given sizes for array lengths and even appropriate machine-dependent statements. This is tending towards the master source/composite file concepts which are described below (section 2.7.3). Some of the potential uses have been eroded by the introduction of such Fortran facilities as PARAMETER, IMPLICIT and generic functions, the introduction of simple macro facilities in compilers (e.g. INCLUDE on the Univac 1100) and the more general availability of context editors.

Fortran macroprocessors include those by Macleod (1970, 1971), Day (1971) and Dasenbrock (1974).

2.6.4.4.4 Structured Fortran Preprocessors

The form of this report demonstrates one of the disadvantages of structured programming viz. that the level at which an item is described is not related to its importance. Another disadvantage, at least at present, is that the Fortran output from preprocessors is crude and opaque. This is remarked by Kernighan (1975) and is the author's own experience with SHELTRAN. For example SHELTRAN produced the following not untypical sequence at the start of a main program:

```
15000 CONTINUE
      IF(.NOT.(
     -.TRUE.
     -)) GO TO 15001
      GO TO 10003
10002 STOP
10003 CONTINUE
      .
      .
      .
```

It is understandable that to tidy up such code would not in general be worthwhile but if such programs ever need to be read by humans the tidying should be possible as an option. (It may be remarked that a well-known optimizing compiler did not optimize the .NOT..TRUE. but it did optimize a subsequent .NOT..NOT. generated by SHELTRAN and it did reconstruct a WHILE-loop which had been demolished by the preprocessor). Some advocates of top-down programming have argued that unstructured coding is logical spaghetti and seem to imply that even computer hardware finds this more inefficient to process. If this is indeed so it will have a hard time with structured Fortran. Another remark of Kernighan's, that it is difficult to relate input to output does not apply to SHELTRAN whose authors have been to some trouble to define the vagaries of statement numbering of certain compilers. Of course, if the output statements are rearranged in any way identification becomes much more difficult.

A further disadvantage and one unfamiliar to Fortran programmers is that a single syntactic error, such as the mispunching of an ENDIF or equivalent, often stops the useful production of diagnostics and starts a sequence of spurious error messages. There is no unanimity on the question of whether the preprocessor should detect "pure" Fortran errors. This is clearly desirable from the programmer's point of view, especially where the output is not easily related to the input, but it puts an extra burden on the preprocessor and given the changing status of some compilers it would be impossible to guarantee error-free output without withdrawing some of the facilities in the Fortran. Further, the translation of structured to ordinary Fortran is in principle machine-independent and a number of preprocessors have been implemented on several different systems (cf. Kernighan).

It remains to be seen how the above disadvantages and the most obvious one of requiring an extra stage in processing, moreover a stage which destroys information useful to a compiler, balance with the potential advantages in designing, writing, debugging, validating and maintaining programs. That these preprocessors are attracting much attention is shown by the list in Appendix A. It is to be hoped that the syntaxes of the new control statements, at least for the production preprocessors, converge rather than diverge but it is still possible that many of them may be made redundant by

the incorporation of structured control statements into the new Standard, or failing that, into the compilers which implement the new Standard.

2.6.4.5 Debugging Aids

The most striking feature about most commercially available debugging aids, and some of those described in publications, is that most or all of their features have been available for years in commonly implemented systems, especially in universities. Some do not even meet the basic requirement that a programmer be addressed at the level at which he addressed the computer; there is no excuse, no need at least on a large system, for a Fortran programmer to be given an octal or a hexadecimal dump to decipher.

For Fortran work in batch mode general requirements appear to be: selective trace of variable assignments, branches and subroutine calls and, at a normal or abnormal termination or whenever requested, a dump of variable names and values at that level and at intermediate levels back to the main program. Further, checks need to be made on the use of unassigned variables. From here it is a short step to having frequency counts of the execution of all statements (section 2.6.4.3).

In interactive mode the above facilities are needed but there are further possibilities for setting breakpoints and for altering the values of variables and for altering code when control reaches a breakpoint or an abnormal termination. The ability to change code in this way does not necessarily lead to a well-structured program.

In view of the tenuous connection between portability and debugging aids and of the very wide variation in the details of the facilities offered as standard by manufacturers and by independent suppliers it is not proposed to discuss them here; the only tool offering significantly different features from those already outlined is the reversible execution in PL/1 of Zelkowitz (1973). Poole (1973) and Satterthwaite (1972) have produced general papers and Brown and Sampson (1973) devote a chapter to the subject.

2.6.4.6 Documentation Aids

Utilities which are offered as documentation aids fall into two groups - those concerned with locating and reporting on usage of variables, labels etc. and those which draw flowcharts, possibly printing variable cross reference lists as well. Both groups have two further subgroups according as the utility is passive, i.e. it accepts the program as given, or active, i.e. it alters the program to make it more readable.

The first group contains programs such as COMFORT, DOCUM, FORDOC, FORTREDIT, FORTREF, FXREF, XREFIV (ICP, 1975), DCE 263 (NEA Program Library no. 327), that described by Merrill (1974) and numerous

programs named RENUMBER or RENBR, one of which is in the SHARE Program Library (no. 360D-99.0.009). The principal concern is with statement numbers: the passive programs report where labels are defined and referenced and the active programs renumber the labels into ascending order. Otherwise the functions vary but include some of the following:

- rename variables
- report where variables are used, including type of usage
- rename subprograms and subprogram references
- renumber continuation card numbers
- report location of STOP, RETURN, CALL or other nominated statements
- report on logical units used
- move all FORMAT statements to the bottom of a program unit
- move all declaration statements to the top of a program unit
- tidy array declarations between DIMENSION, COMMON and type statements
- delete unused labels, FORMAT statements and variable declarations
- reform statements by putting blanks around operators, etc.
- insert comment statements
- indent DO-loops
- sequence lines in columns 73-80
- make some syntax checks

Both active and passive programs produce reports on the actions they have taken and report on the form of the final program.

The flowcharting programs produce a flowchart of a Fortran program on a line printer or on a graph plotter or similar device and often produce some of the reports mentioned above. Programs in this class include AUTOFLOW, FLOWGEN/F, FORFLO, FORTPAK and QUICKDRAW (ICP, 1975) and FFL (Hirose et al., 1974).

It may be remarked that one of the most notorious programming aids ever to appear in Britain was a flowcharting program for Fortran: it deleted any statement in the input program which it deemed to be non-standard and proceeded to draw a chart of the remainder. Often more than half the program was lost. It was quickly withdrawn from the market.

2.6.4.7 Extensions to Fortran

Extensions to Fortran is taken in this context to mean utility programs which duplicate facilities already in some manufacturers' Fortrans, such as free-format input-output, or which provide facilities which may be considered natural language extensions, such as string handling. In as much as some of these programs enable facilities to be matched across different systems, it may be said that they aid transferability but as the statement syntax is likely to be different from machine to machine, it could not be recommended that, say, free format input-output be included in a program intended to be portable. It is more useful to consider the utility programs which are offered in program libraries and described in the literature simply as an indication of the deficiencies of Fortran. This section also includes a summary of an integrated run-time package explicitly aimed at portability.

It appears that free-format input-output, the ability to reread an input record with a different format and the ability to use a format for binary-character conversion and vice versa, within a program, are three of the most-felt wants. All of these facilities are proposed for inclusion in the new Standard although the rereading would have to be done by first copying a record from an input file to a character file. Other popular utilities include allowing Fortran programs to access file-handling functions normally available only at the assembler language level. These, of course, can increase the efficiency of a program at the expense of portability.

There are a number of character and string handling routines available. Again, many of these utilities will be made obsolescent by the new standard but variable-length string handling will not be directly catered for. References to some of these packages are given in section 2.6.3.3 and references to commercially available ones are in ICP (1975); that of Reynolds attracts special attention as a package successfully implemented on several different computers. Multilength arithmetic is also mentioned in section 2.6.3.3; there are no proposals in the new standard to change Fortran in this area.

Other commonly supplied subroutines, such as those for sorting, are less likely to be thought a natural extension of a language and others allowing a Fortran program to access operating system functions are inimical to portability. There are even functions to allow a Fortran program to access a single hardware instruction, such as those to copy or compare arrays; these indicate scope for compiler optimization.

The use of run-time utilities is an essential part of the PFortran system proposed by Whitten and deMaine (1975). This consists of an extended Fortran dialect, PFortran, in which it will be possible to write programs which are machine and configuration independent. A program written in the extended language will be first translated to a Fortran dialect which is

the common subset of the Fortrans on seven major computers. (The paper uses the word PFortran to describe both the input and output for this translation process as well as to describe the run-time system, which is rather confusing, and it claims incorrectly that the common subset is a proper superset of the ANSI standard.) It is proposed to allow not only the normal Fortran storage unit but also a generalized data unit, the kernel, which may be an arbitrary number of words or bytes long, or alternatively may be shorter than a byte or word. Bit types would also be supported within kernels. As well as ordinary arrays of words and kernels the system would support extended arrays, possibly larger than the primary storage available to the program, which would be paged in and out by the PFortran system. Using these facilities, it would be possible for the programmer to nominate the numerical precision desired for each variable and constant separately. Data manipulation and arithmetic that could not be handled by ordinary Fortran language facilities would be performed by the utility routines.

The run-time support package would need to be informed of the available storage and peripheral devices and various other environmental matters at the start of execution of a program. It is proposed that all input and output be performed by utility functions rather than by Fortran itself. An error recovery facility is proposed but it is not clear whether PFortran would attempt to capture control after an interrupt normally processed by the Fortran run-time system or by the operating system.

The preliminary run-times quoted by the authors are quite encouraging; it will be interesting to see if the whole system is viable. It will also be interesting to see if the input-output control package can "be used to assign space in .....a box, shelf, room or building" (p. 119), a task which has defeated many computing centre managements.

2.6.4.8 Translators

Source translators have had a long and honourable history in processing Fortran code since the days of SIFT-Fortran II to IV - (Allen et al., 1963). There have been translators into Fortran, for example from ALTAC, itself really a Fortran dialect (Olsen, 1965), from BASIC (Bevan, 1975) and from EMA (Aitken, 1970) and there have been translators from Fortran, for example to Algol 60 (Pullin, 1964) and to PL/I (IBM). There has even been a Fortran IV to II translator. The Schneck-Angel optimizer could be termed a translator. Source translators have also been found useful in other languages (e.g. McEwan, 1967; Hopgood and Bell, 1967; Bommas, 1968; ICP, 1975, pp 119-126).

Current interest is in programs which convert code between dialects of Fortran or between a dialect and the ANSI Standard language. Such programs have obvious uses for portability but tend to be actually acquired

and used more when an installation is about to change its machine. Beckman (1976) describes a CDC to IBM translator and IBM themselves have a program product which is claimed to convert many dialects of other manufacturers to IBM 360/370 conventions. Jones and Taylor (1973) describe a program which converts certain Univac 1100-dependent statements to ANSI form and the ASA Fortran Translator of Becker (ICP, 1975) converts non-standard statements of "virtually-all Fortran II through Fortran V compilers" to ANSI Fortran.

Translators have generally worked by aiming to convert the bulk of the code accurately and drawing the programmers' attention to items they could not cope with. It is impossible, at the present state of the art, to believe that a translator could successfully convert every program in a dialect to standard form. Apart from concepts not in the Standard, such as direct-access files and assignment to an arbitrary subfield of a word on the Univac 1100, and problems with syntactically identical extensions having different semantics on different systems (v. Muxworthy, 1970), even the use of quite simple extensions may require non-trivial conversion, e.g. some uses of ENTRY, or a statement such as: WRITE(3) (X(L(I)), I=1, N) especially if N is large.

Translators tend to be expensive to write, to rent or to buy and it is not likely that an installation will have them available for occasional use for sending or receiving programs from other machines. This is unfortunate as translators offer a good short term solution to the dialect aspects of the portability problem.

2.6.4.9 Compiler Testers

Compiler testers are not in the same class of Fortran software tool as the above and are included only for completeness. The U.S. Navy set, subsequently taken over by the U.S. Federal COBOL Testing Service, consisted of some 90 short Fortran programs with data, and running them demonstrated whether a system compiled and executed correctly in the sense of the 1966 American standard. While such a test can not be a complete one, it can show up odd errors in compilers. For example on one system a statement which fortunately is unlikely to occur in practice, I=1.6, was found to be compiled as I=2, and some abstruse errors were found even in systems which had been in service for several years.

These programs have now been superseded by a set of 116 programs from the U.S. National Bureau of Standards (Holberton and Parker, 1974); the same set is also available in the form of 14 programs for use on larger systems.

## 2.7 Multi-Machine Software and Distribution

### 2.7.1 Introduction

This section discusses portability from the point of view of the software distributor or the program library. Most libraries act as a depository for programs. The programs may be maintained on a number of nominated computer systems but, if the library is large, the ability to supply any program adapted ready for any computer would require more human and computer resources than could be economically justifiable. This is of course why most of the programs for sale or rent in the ICP Software Directory (1975) specify a list of systems on which they will work. In recent years the idea of multi-machine software based on a single master source file, has gained ground as a tool particularly for distributors of subroutine packages and this is discussed in section 2.7.3. Program libraries per se are outside the scope of this report and section 2.7.2 draws attention to references.

### 2.7.2 Software Distribution

The SHARE program library has been well known for over 15 years and there are now a large number of libraries maintained by computer manufacturers and their user groups, industry, universities, disciplinary user groups, research institutes, government departments and professional societies. Schiffman (1974) gives an overview of libraries and program distribution in North America and Mongini-Tamagnini and Gaggero (1974) describe corresponding activities in Europe. Langenheder (1976) describes available libraries from the point of view of the user. The shipping and subsequent maintenance of software are covered in some detail by Waite (1975) and the paper by Zeckendorf (1973) on character sets acts as a reminder of additional problems in Europe.

### 2.7.3 Master Source Files

The basic concept of the master source or composite file is that Fortran code explicitly tailored for a number of different systems is held in a single file and that a program for a particular computer may be selected from the master file by a preprocessor. This system has the advantage that maintenance of multiple versions is much simplified and also, when many versions are held together, it is relatively easy to identify and eliminate differences which are not essential, thereby tending towards more portable software.

This system tends to have been used most in the context of subroutine packages for numerical mathematics, where the main problems are to do with numerical precision and dialect variation, but it has also been used for single very large programs such as P-STAT (Buhler, 1975), thus also

coping successfully with variations in number of characters per word, file-handling facilities and primary storage sizes. Buhler gives examples of statements which caused particular problems on particular computers and also describes how other differences such as those in overlay systems were dealt with.

The actual form of composite file varies from implementation to implementation. Some systems use command statements intermingled with lines of code so that the whole file looks rather like a Fortran program with additional lines which are not compilable. Others embed the command statements and variant statements as Fortran comments so that the master file is compilable. Others again use replacement techniques so that for example &SIN may be expanded to SIN or to DSIN.

Baron, Schiffman and Fenves (1974) describe a whole set of software designed to enable programs to be submitted, checked for Standard-conformity, put in a master file and subsequently extracted and "customized". Other recent papers discussing some aspects of master files are Krogh (1972), Boyle and Dritz (1974), Ford et al. (1974), Taylor (1974), Hague and Ford (1976) and Schonfelder (1976).

An interesting study has been conducted by NAG (Numerical Algorithms Group) at Oxford who use a master source system for mathematical software on a large number of different computers (cf. Ford et al., 1974). They have supplied Fortran source code to programmers using the various computers, have noted the changes to the software made on implementation and have attempted to classify the reasons for the changes. A surprisingly large number of different reasons occur, as do a disturbingly large number of changes for no apparent reason. Changes made for reasons of efficiency were generally made without balancing the gain received against the cost of making the change. The results of this study have not yet been published.

## 2.8 Discussion

The conventions and software tools outlined above are only aids to the programmer. Each of them addresses only a part of the portability whole and it is the responsibility of the programmer to use the aids intelligently and to write the final program. The overriding need in the production of a program intended to be portable is the desire of the programmer that it be so.

Although legibility and portability of programs are independent attributes legibility is generally considered to be a desirable characteristic of a portable program and is the lowest level of condition mentioned above (section 2.6.2.2). Legibility should not simply imply cosmetic treatment of statements but should imply a well-designed, logically structured program using good, efficient coding techniques. There is remarkable unanimity in

the various published sets of conventions and advice on techniques and the differences between them tend to be complementary rather than contradictory. Programs written in accordance with these rules are eminently readable; some of the best examples are published in the algorithms section of journals or in journals such as Computer Physics Communications. The main criticisms that can be made of the published conventions are firstly that some of them do not go far enough and aim at a superficial legibility with little regard for structure or for portability, and secondly that some of them seek to restrict unduly the programmer's style. The latter may have limited validity but must be doomed to failure generally. There are a variety of good programming styles just as there are a variety of good literary styles and the imposition of arbitrary restrictions on creative programmers is likely to be counter-productive.

It should be emphasized that these conventions are simply conventions: they cannot be enforced mechanically and need the active cognizance and cooperation of programmers to be obeyed. Apart from some companies where coders are very strictly supervised the nearest approach to enforcement would be in the context of IBM's chief programmer team where a group of people "walk through" a sequence of code. In Europe at least to criticize someone's coding is analogous to criticizing his driving and is not done; indirect criticism by editors and referees of journals is acceptable.

The next level of advice published relates to the dialects of Fortran. Some authors advocate the use of the common subset of generally available Fortrans for writing portable programs and a few have gone so far as to document a version of this. This subset is often used in practice with fair success. It does however suffer from the disadvantage that it is ill-defined (and may even vary with new software releases) so that there is no sure way of checking that a program conforms to the subset. More authors suggest adherence to the ANSI Standard which is relatively well defined, documented and described and gives a much firmer foundation on which to build a portable program. Moreover it is possible to check mechanically that a program conforms syntactically to the Standard. Third in this progression, it is possible to use a subset of the ANSI Standard language such as that defined for PFORT. PFORT is the most comprehensive software tool available to overcome differences in dialect and in wordlength so far as the number of characters per word is concerned. Moreover it is a tool which is not associated with a particular computer and is generally available. However, the portability is gained at the expense of a slight loss of language facility and a potentially significant increase in run-time space requirement and execution time if there is much character handling.

The problems of wordlength so far as numerical precision is concerned are common to all computer languages and the analysis of floating-point arithmetic forms a well defined subbranch of mathematics. Possibly for these reasons this topic is not so widely discussed as it could be in the context of portability in general but it naturally occurs in the literature

of numerical software.

Other spheres such as differences in input-output unit capabilities are of lesser import with most modern operating systems and there is little that can be usefully said in detailed terms over such a wide field. Programmers writing portable programs should read at least one of the general review papers, one of the Fortran dialect comparisons and some direct implementation experience (such as Buhler) to get a better understanding of the gross and the subtle variations between computer systems and thus to appreciate what is machine independent and what is not.

Apart from the syntax checker the other main tool of direct relevance to portability is the translator which offers a quick and accurate solution to dialect problems; its disadvantage is usually the cost. Translators tend to be expensive and a heavy throughput, which is usually over a limited period of time, is necessary to justify their acquisition. Their use for occasional inter-machine transfer is therefore not common and there could be advantages in the setting up of central translation services.

The final direct tool, strictly for multi-machine rather than for portable software, is the master source & macroprocessor system which tends to be used by cooperative groups, although there is no reason why an individual should not use it.

The bulk of the software aids discussed in section 2.6.4 are not directly related to portability but help the programmer to gain a better understanding of a program. The more important of the static analyses are the cross-reference listing, some of the tidying facilities, the structural analysis (e.g. Schneck-Angel) and the path analysis (e.g. DAVE). The more important dynamic analyses are frequency counts and the run-time tests made by some checking compilers such as WATFIV. All these help to indicate structure, flow of control or observance of the standard. Some of the other software tools may be useful in limited applications: those which generate Fortran programs often produce code which is ugly and obscure and a few of the others are of dubious utility.

The two current major developments in Fortran, the new draft Standard and structured programming, are more recent than most of the portability literature and may make some of it obsolescent. The new Standard and the corresponding compilers will give a new aspect to the dialect problem; hopefully variations will not be so strong as in the past. Even if they are the new Standard marks a significant advance in facilities and programmers should have less cause to stray outside its limits.

Structured Fortran offers another significant advance in facilities which may or may not be included in the new Standard. If they are, Fortran will be strengthened in its current position; if they are not, there seems likely to be a period of increasing use of structured Fortran preprocessors which could aid portability at the Fortran level through incorporation of syntax

checkers into the preprocessors, but which would be chaotic at the level of structure command syntax.

Two current developments in portability which it will be interesting to follow are those of Whitten and deMaine, and Dahlstrand. To use the railway analogy so often applied to standards, most workers are content if trains run throughout most of Europe on one gauge (4 feet 8 1/2 inches, 1435 mm) and have to have bogies changed for Russia (5 feet, 1524 mm) or Spain (5 feet 6 inches, 1676 mm); Whitten and deMaine would have all trains equipped with multiple wheels to run on any track, Dahlstrand would have all countries adopt the same gauge and would widen tunnels and clearances as necessary. (Russia's railways were deliberately made different for defence reasons)

## 2.9 Conclusions

There is a widespread desire for programs to be portable for reasons of cost and general efficiency and there has been increased awareness of this particularly in the past three years. With real systems, portability in all aspects is something to be approached rather than achieved and this possibly explains the paucity of relevant software tools. A literature has grown in recent years but this too, while it contains much useful advice, can offer no guarantee of success. Given the wide range of environments in which a program may have to operate, portability is more a skill than a science. Its best exponent is an intelligent informed programmer.

# 3. PROGRAM DOCUMENTATION

## 3.1 Overview of Documentation Paradigms

The literature of program documentation is remarkable, firstly because it contains an American Standard which cannot possibly be enforced and secondly because published reviews have almost invariably been censorious. Publications on documentation are relatively recent, first appearing not long before those on portability; Gray and London (1969) was described as the first book commercially published devoted to documentation. Gray and London gave a full set of guidelines, with examples, for analytic, system, program, operational, user and management documentation. This work was followed by Walsh (1969), Kuehne et al. (1972), van Duyn (1972), NCC (1972 and 1973) and Robinson and Graviss (1973). The American Standard (sponsored by the American Nuclear Society) appeared in 1974. It addresses itself to the program abstract, user manual, problem definition and programmer manual but it is only a five page checklist of headings, some of which are cryptic in the extreme, e.g. discuss any man-machine interactions, so that while it provides useful guidelines, most junior programmers would have to be referred to one of the more complete guides, perhaps the NCC ones, or to one of the many local standards in use at installations, or to the semi-public models of program libraries or journals which publish programs.

Overview papers include those by Flores (1972), Goos (1973), Brown (1974b) Sheaks (1974) and Harper (1975); that of Goos is highly recommended. Case study papers include Katzenelson (1971) and Henrywood (1973).

All the above papers tend to concentrate on documentation about the program without necessarily affecting the program. Roberts (1969) and Banks et al. (1972a) begin from comments in the program building up to separate user documents. Roberts outlines possibilities of automatic documentation of programs; while some of the program documentation aids (section 2.6.4.6) have proved useful, a software tool cannot get more information out of a source program than the programmer put in and for this reason experience with them has been generally disappointing.

## 3.2 Discussion

Criticism of the published guidelines has not been because they are misleading, or offer contradictory or controversial advice, but more because they omit some fields, e.g. fail to recognize the introductory guide/reference manual distinction for users, because they are biased to particular applications, because they are too obvious and underestimate the reader or simply because they are too verbose. Without making a detailed comparison it is not possible to come to any conclusion; the American Standard is at least succinct and reasonably comprehensive.

REFERENCES

Aitken, L.D. (1970); An EMA to Fortran Translator. Program Library Unit, University of Edinburgh

Allan, J.J., D.P. Moore and H.P. Rogaway (1963); SIFT-SHARE Internal Fortran Translator. Datamation, vol. 9 (March), pp 43-46

American National Standards Institute (1966a); American National Standard FORTRAN (ANS X3.9-1966). ANSI, New York

American National Standards Institute (1966b); American National Standard Basic FORTRAN (ANS X3.10-1966). ANSI, New York

American National Standards Institute Subcommittee X3J3 (1971); Clarification of Fortran Standards - Second Report. CACM, vol. 14, pp 628-642

American National Standards Institute Subcommittee X3J3 (1976); Draft Proposed ANS FORTRAN. ACM SIGPLAN Notices, vol. 11, no. 3 (March)

American Nuclear Society (1974); American National Standard Guidelines for the Documentation of Digital Computer Programs, ANSI N413-1974. ANS, Hinsdale, Illinois

American Standards Association X3 Committee (1964); Fortran vs. Basic Fortran. CACM, vol. 7, pp 591-625

Baker, F.T. (1972); Chief programmer team management of production programming. IBM Systems Journal, vol. 11, pp 56-73

Baker, F.T. and H.D. Mills (1973); Chief programmer teams. Datamation vol. 19 (Dec.) pp 58-61

Banks, D., I.C. Percival and J.M. Wilson (1972a); Stirling Fordoc 01 : A set of documentation conventions for Fortran packages and routines. Computer Physics Communications, vol. 3, pp 180-196

Banks, D., I.C. Percival and J.M. Wilson (1972b); Classical relative motion of 2 particles (EVAR edition 02). Computer Physics Communications, vol. 3, pp 197-220

Baron, M.L., R.L. Schiffman and S.J. Fenves (1974); A national software center for engineering. pp 615-631 in Pilkey et al. (q.v.)

Beckman, A (1976); Automatic translation between programming language dialects (submitted to Software Practice and Experience)

Bemer, R.W. (1969); A politico-social history of Algol. Annual Review in Automatic Programming, pp 151-237. Pergamon Press, Oxford

Berkowitz, R. L. (1970); A comparison of some Fortran languages. U.S. Naval Research Laboratory Memorandum Report 2191, October 1970

Bevan, J. D. (1975); BASIC to Fortran automatic translation. Mimeographed note, August 1975. National Development Programme in Computer Assisted Learning, London

Böck, R. K. (1975); Program Portability in High-Energy Physics. Computer Physics Communications, vol. 9, pp 221-229

Böhm, C. and Jacopini (1966); Flow diagrams, Turing machines and languages with only two formation rules. CACM, vol. 9, pp 366-371

Bommas, W. (1968); An Algol 60 to PL/1 translator. Proc. SEAS XIII Conference, Scheveningen. SEAS, Nijmegen

Boyle, J. and K. Dritz (1974); An automated programming system to facilitate the development of quality mathematical software. IFIP 74 Proc., pp 542-546

Brown, A. R. and W. A. Sampson (1973); Program debugging. Macdonald, London

Brown, F. D., V. J. Calderbank and M. D. Poole (1971); Some comments on the portability of a large Algol program. Software Practice and Experience, vol. 1, pp 367-371

Brown, P. J. (1974a); Macroprocessors and techniques for portable software. Wiley, London

Brown, P. J. (1974b); Programming and documenting software projects, Computing Surveys, vol. 6, pp 213-220

Brown, W. S. (1969); Software Portability. Proc. NATO Conf. Techn. Software Eng., Rome

Buhler, R. (1975); P-STAT Portability. Proc. Computer Science and Statistics, 8th Annual Symposium on the Interface, UCLA, February, 1975

Cardetta, L., O. Murro, F. Stea and E. Uva (1976); Un precompilatore per la codifica strutturata in ambiente Fortran. CSATA Internal Report, April 1976, Bari

Cardetta, L., O. Murro and E. Uva (1975); Una esperienza di implementazione ed utilizzo di un precompilatore Fortran per la diffusione di tecniche di programmazione strutturata. Congresso AICA 1975, Genova, October 1975

Christiansen, J. P. and K. V. Roberts (1974); OLYMPUS - A Standard control and utility package for initial value Fortran programs. Computer Physics Communications, vol. 7, pp 245-270

Clark, R. L. (1973); A linguistic contribution to GO TO-less programming. Datamation, vol. 19 (Dec.), pp 62-63

Croes, G. A. (1975); A user's concern with programming methodology. Proc. SEAS Spring Technical Meeting, Aalborg, Denmark, April 1975, pp 31-48. SEAS, Nijmegen

Croes, G. A. and F. Deckers (1975); Aspects of structured programming in Fortran. Informatie, vol. 17, pp 121-131

Dahl, O. J., E. W. Dijkstra and C. A. R. Hoare (1972); Structured programming. Academic Press, London

Dahlstrand, I. (1976); Portabilitet inom teknisk-vetenskaplig ADB. STU-rapport 74-4022. Lunds Datacentral, Lund

Dasenbrock, R. R. (1974); An editing routine for Fortran programs. Naval Research Laboratory report AD-779841/6

Day, A. C. (1971); FORMAPRO - Fortran Macroprocessor. University College London Computing Centre, Technical Report no. 2

Day, A. C., P. A. Clarke, D. Hill and J. K. Reid (1976); An Examination of Fortrev (to be published in the Computer Journal)

Dekker, T. J. (1971); A floating point technique for extending the available precision. Numerische Mathematik, vol. 18, pp 224-242

Dijkstra, E. W. (1968); GO TO statement considered harmful. CACM, vol. 11, pp 147-148

Donaldson, J. R. (1973); Structured programming. Datamation, vol. 19 (December), pp 52-54

Engel, F. (1974); Revise Standard Fortran? Datamation, vol. 19 (May), pp 164-169

European Computer Manufacturers Association (1965); ECMA Standard on Fortran (ECMA-9), ECMA, Geneva

Flores, I. (1972); Intraprogram documentation. Software Practice and Experience, vol. 2, pp 353-358

Ford. B. (1976); Machine characteristics and their parameterisation in numerical software. Mimeographed note. 22 January, 1976, NAG, Oxford

Ford, B., S. J. Hague, J. A. Prentice and D. B. Taylor (1974); The development and maintenance of multi-machine software in the NAG project, in D. J. Evans (editor) Software for Numerical Mathematics, Academic Press, London

FOR-WORD, Fortran Development Newsletter. Edited by L. P. Meissner, 50-B 3239 Lawrence Berkeley Laboratory, Berkeley, California 94720

Fosdick, L. D. (1974); BRNANL - A Fortran program to identify basic blocks in Fortran programs. University of Colorado report PB-235 294/6, March 1974

Fosdick, L. D. and L. J. Osterweil (1974); Automated input-output variable classification as an aid to validation of Fortran programs. University of Colorado report PB-235293/8

Fosdick, L. D. and L. J. Osterweil (1976); DAVE - A valuation, error detection and documentation system for Fortran programs. (Submitted to Software Practice and Experience)

Friedrich, H. J. (1975); Fortran Standards - Empfehlungen für die standardisierte Anwendung von Fortran für Programme zur Auswertung medizinischer Datenbestände. Beiheft zum Statistical Software Newsletter B 1, February 1975, University of Giessen, Germany

Fries, G. (1975); The problem of compatibility - an introduction. Report LU/CS 75:07, 1975. Department of Computer Science, Lund University

Ghan, L. (1971); Better techniques for developing large scale Fortran programs. Proc. ACM 1971 Annual Conference, New York, pp 520-537

Goos, G. (1973); Documentation. pp 385-394 in F. L. Bauer (editor) Advanced Course in Software Engineering, Springer, New York

Gray, M. and K. R. Loudon (1969); Documentation Standards. Brandon/System Press, Princeton, New Jersey

Hague, S. J. and B. Ford (1976); Portability - prediction and correction. Software Practice and Experience, vol. 6, pp 61-69

Harper, W. L. (1975); Building EDP success by standing on shoulders. Computer, vol. 8, pp 50-56

Heising, W. P. (1964); History and summary of Fortran standardization development for the ASA. CACM, vol. 7, p 590

Henrywood, R. K. (1973); The design, development, documentation and support of a major finite element system. Computer aided design, vol. 5, pp 160-165

Hertweck, F. (1970); String Handling in Fortran. Proc. SEAS XV Conference, Munich. SEAS, Nijmegen

Hill, I. D. (1968); Procedures for the basic arithmetical operations in multi-length working. Computer Journal, vol. 11, pp 232-235

Hill, I. D., R. S. Scowen and B. A. Wichmann (1975); Writing algorithms in Algol 60. Software Practice and Experience, vol. 5, pp 229-244

Hirose, K., K. Utsunowiya, M. Sakakura and F. Hamma (1974); Design of Fortran-oriented Flowchart program FFL and its implementation. Information processing in Japan, vol. 14, pp 20-25

Hoare, C.A.R. (1973); High level programming languages - the way behind. pp 17-26, in "High level programming languages - the way ahead", NCC publications, Manchester

Holberton, F.E. and E.G. Parker (1974); NBS Fortran test programs, vols. 1-3. NBS Reports COM-75-50138/7, 50139/5, 50140/3, Washington, D.C.

Hopgood, F.R.A. and A.G. Bell (1967); The Atlas Algol preprocessor for non-standard dialects. Computer Journal, vol. 9, pp 360-364

Horowitz, E. (1975); Fortran - Can it be structured - should it be. Computer, vol. 8 (June), pp 30-37

Hughes, M.H., K.V. Roberts and G.G. Lister (1975); OLYMPUS and utility package for the CDC 6500. Computer Physics Communications, vol. 10, pp 167-181

Hughes, M.H., K.V. Roberts and P.D. Roberts (1975); OLYMPUS and utility package for an IBM 370/165. Computer Physics Communications, vol. 9, pp 51-58

International Computer Programs (1975); ICP Software Directory, vol. 1, July 1975. ICP, Carmel, Indiana

International Organization for Standardization (1972); Programming Language Fortran ISO Recommendation no. 1539. ISO, Geneva

Jones, R.A. and D.P. Taylor (1973); FR 1108 - A program to transfer certain machine dependent Univac 1108 statements to ANSI Fortran. University of Colorado Computing Center Report 73-25

Kamynin, S.S. and E.Z. Lyubimskiy (1967); Algoritmicheskiy mashino-orientirovannyy yazyk ALMO. no. 1 in the series Algoritmy i algoritmicheskie yazyki. Akademia Nauk, Moscow

Katzenelson, J. (1971); Documentation and the management of a software project. Software Practice and Experience, vol. 1, pp 147-157

Kernighan, B.W. (1975); RATFOR: A preprocessor for rational Fortran. Software Practice and Experience, vol. 5, pp 395-406

Knuth, D.E. (1971); An empirical study of Fortran programs. Software Practice and Experience, vol. 1, pp 105-133

Knuth, D.E. (1974); Structured programming with GO TO statements. Computing Surveys, vol. 6, pp 261-302

Krogh, F.T. (1972); A method for simplifying the maintenance of software which consists of many versions. S 914 Technical Memorandum 314. Jet Propulsion Laboratory, Pasadena, California, 15 September 1972

Kuehne, R.S., H.W. Lindberg and W.F. Baron (1972); Manual of computer documentation standards with forms. Prentice Hall, Englewood Cliffs, New Jersey

Kulsrud, H.G. (1974); Some statistics on the reasons for compiler use. Software Practice and Experience, vol. 4, pp 241-249

Langenheder, W. (1976); The present state in Europe of software access and information transfer in the field of research and teaching. pp 14-22 in W. Langenheder and H. Züllighoven (editors) Proceedings of the 1. ECSIR Workshop, February, 1976. GMD, Bonn

Larmouth, J. (1973); Serious Fortran. Software Practice and Experience, vol. 3, pp 87-107 and pp 197-225

Luchovitskaya, E.S. (1974); Analiz kachestvo translatorov, razrabotannykh na baze ALMO (Analysis of the quality of compilers developed using ALMO). Akademia Nauk, Moscow

Lyon, G. and R.B. Stillman (1975); Simple transforms for instrumenting Fortran decks. Software Practice and Experience, vol. 5, pp 347-358

Macleod, I.A. (1970); SP/1 - A Fortran integrated string processor. Computer Journal, vol. 13, pp 255-260

Macleod, I.A. (1971); MP/1 - A Fortran macroprocessor. Computer Journal, vol. 14, pp 229-231

McCormick, J.M. (1974); Programming for effective interchange. pp 651-667 in Pilkey et al. (q.v.)

McCracken, D.D. (1965); How to tell if it's Fortran IV. Datamation, vol. 11 (October), pp 38-41

McCracken, D.D. (1973a); Is there a Fortran in your future? Datamation, vol. 19 (May), pp 236-237

McCracken, D.D. (1973b); Revolution in programming: an overview. Datamation, vol. 19 (December), pp 50-52

McCracken, D.D. and G.M. Weinberg (1972); How to write a readable Fortran program. Datamation, vol. 18 (October), pp 73-77

McEwan, A.T. (1967); An Atlas Autocode to Algol 60 translator. Computer Journal, vol. 9, pp 353-359

Meissner, L.P. (1975); On preprocessors. ACM SIGPLAN Notices, vol. 10 (December) p. 39

Meissner, L.P. (1976a); Proposed ANS X3.9 Fortran language revision. Forword, vol. 1, no. 6 (January, 1976)

Meissner, L.P. (1976b); A compatible structured extension to Fortran. ACM SIGPLAN Notices, vol. 11, no. 2

Merrill, R.G. (1974); Fortran concordance program. National Oceanic and Atmospheric Administration report COM-74-11187/3

Miller, E. F. and G. E. Lindamood (1973); Structured programming: the top-down approach. Datamation, vol. 19 (December) pp 55-57

Mongini-Tamagnini, C. and G. Gaggero (1974); Software Exchange within the European Community. pp 633-649 in Pilkey et al. (q. v.)

Moulton, P. G. and M. E. Muller (1967); DITRAN - A compiler emphasizing diagnostics. CACM, vol. 10, pp 45-52

Murison, J. M. (1974); Program in internal use at Edinburgh Regional Computing Centre. (not yet published)

Muxworthy, D. T. (1970); Dialects of Fortran. Proc. SEAS XV Conference, Munich, pp 235-244. SEAS, Nijmegen

Muxworthy, D. T. (1972a); Standard Fortran - a short history. Computer Bulletin, vol. 16, pp 211-213

Muxworthy, D. T. (1972 b); On the portability of Fortran programs. Management Informatics, vol. 1, pp 125-127

Muxworthy, D. T. (1976); The new standard Fortran. SEAS Newsletter, June 1976. SEAS, Nijmegen

Muxworthy, D. T. and B. H. Shearing (1970); A review of some dialects of Fortran. Alcock, Shearing and Partners, London

National Computing Centre (1972a); Standard Fortran Programming Manual (Second Edition). NCC, Manchester. Reviewed in Computer Journal (1974), vol. 17, p 123 and Computing Reviews (1974), vol. 15, p 19

National Computing Centre (1972b); Programming Standards: vol. 1 Documentation. NCC, Manchester

National Computing Centre (1973); Systems Documentation Manual (Third Edition), NCC, Manchester

Naur, P. (1967); Machine dependent programming in common languages. BIT, vol. 7, pp 123-131

Neely, P. (1976); The new programming discipline. Software Practice and Experience, vol. 6, pp 7-27

Olsen, T. M. (1965); Philco/IBM translation at problem-oriented, symbolic and binary levels. CACM, vol. 12, pp 762-768

Oswald, H. (1964); The various Fortrans. Datamation, vol. 10 (August), pp 25-29

Paige, M. R. and J. P. Benson (1974); The use of software probes in testing Fortran programs. Computer, vol. 7, pp 40-47

Papakonstantinou, P. (1975); A portable preprocessor for recursive Fortran. Angewandte Informatik, vol. 17, pp 492-494

Pilkey, W., K. Saczalski and H. Schaeffer (editors) (1974); Structural Mechanics Computer Programs. University Press of Virginia, Charlottesville

Poole, P.C. (1973); Debugging and testing. pp 278-318 in F.L. Bauer (editor) Advanced Course in Software Engineering, Springer, New York

Pullin, D. (1964); A Fortran to Algol Translator. Computer Journal, vol. 7, pp 24-27

Rabinowitz, I.N. (1962); Report on the algorithmic language Fortran II. CACM, vol. 5, pp 327-337

Ralston, A. (1973); The future of higher level languages (in teaching). International Computing Symposium, pp 1-10

Ramamoorthy, C.V., R.C. Cheung and K.H. Kim (1974); Reliability and integrity of large computer programs. University of California Berkeley, Report AD-779339/1. 12 March, 1974

Ramamoorthy, C.V. and S.F. Ho (1974); Fortran Automatic Code Evaluation System (FACES). University of California Berkeley. Report AD/A-004168/1. 25 July,1974

Reifer, D.J. (1975); Automated aids for reliable software. Aerospace Corporation Report AD-A014821/3. 26 August, 1975

Reifer, D.J. (1976); The structured Fortran dilemma. ACM SIGPLAN Notices, vol. 11 (February) pp 30-32

Reynolds, R.A. (1976); String Manipulation in Fortran. (Submitted to the Computer Journal)

Roberts, K.V. (1969); The publication of scientific Fortran programs. Computer Physics Communications, vol. 1, pp 1-9

Roberts, K.V. (1974); An introduction to the OLYMPUS system. Computer Physics Communcations, vol. 7, pp 237-243

Robinson, J.P. and J.D. Graviss (1973); Documentation Standards Manual for Computer Systems. Associated Systems Management, Cleveland

Robinson, S.K. and I.J. Torsun (1976); An empirical analysis of Fortran programs. Computer Journal, vol. 19, pp 56-62

Rogers, J.G. (1975); Structured programming for virtual storage systems. IBM Systems Journal, vol. 14, pp 385-406

Rohl, J.S., H.D. Ellison and R.J. Collins (1975); An incore batching standard Fortran compiler for large 1900 machines. Software Practice and Experience, vol. 5, pp 19-28

Rosen, S. (1961); ALTAC, Fortran and compatibility. ACM 16th National Conference Preprints, pp 2B-2(1)-(4)

Ryder, B.G. (1974); The PFORT Verifier. Software Practice and Experience, vol. 4, pp 359-377

Sackman, H., W.J. Eriksen and E.F. Grant (1968); Exploratory Experimental Studies comparing on-line and off-line programming performance. CACM, vol. 11, pp 3-11

Sale, A.H.J. (1971); The classification of Fortran Statements. Computer Journal, vol. 14, pp 10-13

Sammett, J. (1969); Programming Languages: History and Fundamentals. Prentice Hall, Englewood Cliffs, New Jersey

Sanders, N. and Fitzpatrick (1963); Algol and Fortran revisited. Datamation, vol. 9 (January) pp 30-32

Satterthwaite, E. (1972); Debugging tools for high level languages. Software Practice and Experience, vol. 2, pp 197-217

Schiffman, R.L. (1974); Current software dissemination practices and organizations. pp 591-614 in Pilkey et al. (q.v.)

Schneck, P.B. and E. Angel (1973); A Fortran to Fortran optimizing compiler. Computer Journal, vol. 16, pp 322-330

Schofield, C.F. (1968); A study of Fortran Compatibility. University of London, Atlas Computing Service

Schonfelder, J.L. (1976); The production of special function routines for a multi-machine library. Software Practice and Experience, vol. 6, pp 71-82

SHARE Program Library Agency User's Guide and Catalog of Programs. Annual Publication. SHARE Inc., Chicago

Shaw, A. (1972); Program in internal use at University College, London Computing Centre

Sheaks, D. (1974); A blueprint for documentation standards. Data Processing, vol. 16, pp 382-390

Stuart, F. (1969); Fortran Programming. Wiley, New York

Taylor, D.B. (1974); Management practices in the development and distribution of mathematical software with emphasis on computational aids. pp 373-382 in D.J. Evans (editor) Software for Numerical Mathematics, Academic Press, London

Thorlin, F. (1972); What's new with DO? Datamation, vol. 18 (December), pp 142-144

Traub, J.F. (1971); High quality portable numerical mathematical software. pp 131-139 in J.R. Rice (editor) Mathematical Software, Academic Press, New York

United States of America Standards Institute X3 Committee (1969); Clarification of Fortran Standards - Initial progress. CACM vol. 12, pp 289-294

van der Riet, R. P. (1973); ABC Algol - a portable language for formula manipulation systems. Mathematisch Centrum, Amsterdam. Tracts 46 and 47

van Duyn, J. (1972); Documentation Manual. Auerbach

van Wijngaarden, A., B. J. Mailloux, J. E. L. Peck, C. H. A. Koster (1969); Report on the algorithmic language Algol 68. (inter alia:) Numerische Mathematik, vol. 14, pp 79-218

Warburton, B. (1976); SHORTRAN - program in internal use at the London School of Pharmacy

Waite, W. M. (1975); Hints on distributing portable software. Software Practice and Experience, vol. 5, pp 295 - 308

Walsh, D. (1969); A guide for software documentation. Advanced Computer Techniques Corporation and McGraw-Hill, New York

Whitten, D. E. and P. A. D. deMaine (1975); A machine and configuration independent portable Fortran (PFortran). IEEE Transactions on Software Engineering, vol. SE-1, pp 111-124

Wichmann, B. A. (1973); Algol 60: Compilation and Assessment. Academic Press, London

Winske, P. (1975); Variable Dimensionierung von Feldern in grossen Programmen. Angewandte Informatik, vol. 17, pp 479-482

Wirth, N. (1974); On the composition of well-structured programs. Computing Survey, vol. 6, pp 247-259

Working Party on Fortran Extensions (1971a); The next Standard Fortran? Computer Bulletin, vol. 14, pp 312-314

Working Party on Fortran Extensions (1971b); Some comments on extending Fortran, 48 pp. Mimeographed note

Working Party on Statistical Computing (1975); The construction and description of algorithms. Applied Statistics, vol. 24, pp 366-373

Wright, D. C. (1966); A comparison of the Fortran language implementations for several computers. CACM, vol. 9, pp 77-79

Yershov, A. P. (editor) (1971); The ALPHA Automatic Programming System. Academic Press, London

Zeckendorf, L. J. (1973); Character sets en codes. Informatie, vol. 15, pp 653-661

Zelkowitz, M. V. (1973); Reversible Execution. CACM, vol. 16, p 566.

# APPENDIX A

## STRUCTURED FORTRAN PREPROCESSORS

There follows a list of structured Fortran preprocessors which have been mentioned in the literature. It does not claim to be exhaustive.

| Name | Author or Contact | Organization |
|---|---|---|
| ATHENA-ML | J. Perrine | Jet Propulsion Lab, Pasadena |
| B4TRAN | L. P. Meissner | Lawrence Laboratory, Berkeley, CA |
| COM-SP | R. E. Jeffries | COMSHARE, Ann Arbor |
| DAY-SMP | A. C. Day | University College, London |
| DEFT | T. E. Hull | University of Toronto |
| DO-OD IF-FI | J. L. Wagener | State University of New York, Brockport |
| ELESSAR | R. Bond | Essay Corp., Oklahoma City |
| ESFOR | W. L. Bearley | Citrus College, Azusa, CA |
| FDP-5798-CDW | G. McCool | IBM, Glendale, CA |
| FLECS | T. Beyer | University of Oregon, Eugene |
| FORTRAN-S | T. E. Hull | University of Toronto |
| FORTRAN-4S | R. S. O'Bryant | Texas Instruments, Dallas |
| FORTSP | T. Saisho | (Tokyo) |
| HIGGINS | D. S. Higgins | Florida Power Corp., St. Petersburg |
| IFP | R. E. Kottler | Intermetrics, Cambridge, MA |
| IFTRAN-1 | R. O. Parker | General Research Corp., Santa Barbara |
| IFTRAN-2 | R. O. Parker | " " " " |
| IFTRAN-C | W. R. Bezanson | Carleton University, Ottawa |
| IFTRAN-LA | M. Cohen | University of Southern California, Marina del Rey |
| IFTRAN-W | D. L. Dietmeyer | University of Wisconsin, Madison |
| IFTRAN-X | E. F. Miller | General Research Corp., Santa Barbara |
| IPLFOR | A. Schwartz | Jet Propulsion Lab., Pasadena |
| LINUS | C. M. Bernstein | Bowling Green State Univ., Ohio |
| MELTRAN | J. S. Miller | Intermetrics Inc., Cambridge, Mass. |
| MORTRAN-1 | A. J. Cook | Stanford LAC, Stanford |
| MORTRAN-2 | A. J. Cook | " " " |
| MORTRAN-X | R. H. Ault | University of Utah, Salt Lake City |
| MTUFP | J. Lowther | Michigan Tech. University, Houghton |
| NSF-TRAN | F. Friedman | Temple University, Philadelphia |
| PMDS | D. J. Reifer | Aerospace Corp., Los Angeles |

| Name | Author or Contact | Organization |
|---|---|---|
| PREFOR | W. M. Bradley | IBM, Houston |
| PREST4 | N. Kaffer | Ohio State University, Columbus |
| PSST | L. Stucki | McDonnell Douglas, Huntington Beach, CA |
| RATFOR | B. Kernighan | Bell Labs., Murray Hill, NJ |
| SAFTE | J. M. Kamrad | Sperry Univac, St. Paul |
| SFOR | D. O'Neill | Bell Labs., Holmdel, NJ |
| SFORTN | L. L. Pooler | Hughes Aircraft, Oceanside, CA |
| SFOR-LSI | T. Mizoguchi | Mitsubishi, Tokyo |
| SFP | R. Rich | Johns Hopkins University, Silver Spring, MD |
| SFTRAN | J. Flynn | Jet Propulsion Lab., Pasadena |
| SF-CHAT | F. Foldvary | Lawrence Laboratory, Livermore, CA |
| SF-CONCORDIA | T. Radhakrishnan | Concordia University, Montreal |
| SF-HENKE | W. L. Henke | MIT, Cambridge, MA |
| SF-RIO | R. N. Melo | Pontificia Univ. Catolico, Rio de Janeiro |
| SF-VISNAVICH | F. Towster | University of Southwestern Louisiana, Lafayette |
| SGOL | C. T. Zahn | Stanford LAC, Stanford |
| SHELTRAN | G. A. Croes | Shell International Petroleum, London |
| SIXTRAN | W. H. Burkhardt | University of Stuttgart |
| SPA | S. A. Steele | RCA, Moorestown, NJ |
| SPARKS | E. Horowitz | University of Southern California, Los Angeles |
| SPDS | D. Gormley | IBM, Westlake Village, CA |
| SPFORT | B. Press | TRW, Redondo Beach, CA |
| SPIFFY | L. Carpenter | Boeing Computer Services, Renton, WA |
| SPL | A. Wilson | On-line Systems Inc., Pittsburgh |
| SPL&C | A. Holgado | University of Michigan, Ann Arbc |
| STAPLE | S. L. Stewart | National Bureau of Standards, Washington |
| STRAN | W. L. Johnson | Ford Motor Co., Dearborn |
| STRUFO | G. Maioli | C. N. E. N., Bologna |
| STRUFORT | O. Murro | C. S. A. T. A., Bari |
| SUGFOR | A. Beckmann | Uppsala University |
| S-FORTRAN | G. de Balbine | Caine, Farber & Gordon Inc., Pasadena |
| S-WATFIV/WM | N. E. Gibbs | College of William and Mary, Williamsburg, VA |
| TRANSFOR | L. Carpenter | Boeing Computer Services, Renton, WA |
| TR6B-TR7X | L. Mossberg | Volvo, Trollhättan, Sweden |
| WATFIV-S | P. Dirksen | University of Waterloo, Ontario |
| XFORT | A. Gyarfas | Hungarian Academy of Science, Budapest |
| ZCMP | Z. C. Motteler | Michigan Tech. University, Houghton |

The list contains 67 items.

The geographical distribution is:

USA 51 (incl. California 21),
Canada 5,
Britain 2,
Italy 2,
Japan 2,
Sweden 2,
Brazil 1,
Germany 1,
Hungary 1.

# APPENDIX B

## OVERVIEW OF SOFTWARE TOOLS MENTIONED

| Section | Function |
|---|---|
| 2.6.4.2 | Syntax checkers |
| 2.6.4.3 | Statement identifier |
| | Frequency counters |
| | Structure analyser |
| | Path analyser |
| | Program linkage analyser |
| | (plus several undefined checkers) |
| 2.6.4.4.1 | Special purpose preprocessors |
| 2.6.4.4.2 | Fortran-to-Fortran optimizer |
| | Preprocessor to dimension arrays |
| | " " allocate common blocks |
| | " for recursive Fortran |
| | " to all abbreviations of keywords |
| 2.6.4.4.3 | Macroprocessors |
| 2.6.4.4.4 | Structured Fortran preprocessors |
| 2.6.4.5 | Debugging aids |
| 2.6.4.6 | Source concordance programs |
| | Source tidying programs |
| | Flowcharting programs |
| 2.6.4.7 | Utilities |
| | Character-handling routines |
| 2.6.4.8 | Fortran II to IV translators |
| | Fortran dialect-to-dialect translators |
| | Fortran dialect-to-standard translators |
| | Fortran-to-other language translators |
| | Other language-to-Fortran translators |
| | Other language translators |
| 2.6.4.9 | Compiler testers |