A free-format data input scheme written in standard FORTRAN (ANSI 66)

G. F. Butler and J. Pike

Royal Aircraft Establishment, Farnborough, Hampshire GU14 6TD, UK

The 1966 ANSI standard FORTRAN specifies the input of numerical data only in fixed format. Described here is a scheme, written in ANSI 66 FORTRAN, which allows numerical data prepared in accordance with the FORTRAN 77 free-format specification to be read. In addition, the scheme is designed so that ill-formed or ambiguous data are given a reasonable interpretation and the location of the suspect data and the value assumed are identified.

(Received October 1979)

1. Introduction

Programs written in standard FORTRAN IV are limited to data input with fixed format. Often it is more convenient to input data separated by commas or spaces and, in recognition of this, various manufacturers have introduced their own free-format data input routines. Unfortunately, these are not standard and programs and data prepared for input in free-format on one manufacturer's machine cannot, in general, be transferred to those of another manufacturer. In this paper, free-format data input routines, written in standard ANSI 66 FORTRAN (ANSI, 1966) are presented.

The scheme enables data to be read into various arrangements of variables, with each routine starting to read on a new line of data and continuing until either a specified number of numbers has been read or an end of read symbol (/ or $) is encountered. In specifying the routines, the aim has been to give the correct interpretation of all well-formed numbers and, when ambiguous data are encountered, to give a reasonable interpretation and to issue a warning. A failure in the input routines can only occur from a system failure, for example if a number is too large for the computer to handle or an attempt is made to read beyond the end of a file. No 'end of file' detection facility exists in standard FORTRAN, hence this facility of FORTRAN 77 and many non-standard implementations cannot be used in standard FORTRAN 66.

Whilst one of the aims in the specification of the routines has been to maintain compatibility with existing formated data for FORTRAN programs, this has not always been possible. For example, in order to read the two numbers 149 and 736 in FORTRAN 213 format, they would appear in the data as 149736. The input scheme described here would interpret this as the single number 149736. For all cases where formated data are separated by a non-digit character, the routines will read in the numbers as intended. In addition, the routines will interpret correctly free-format data prepared in accordance with the FORTRAN 77 (ANSI, 1978) standard, except for a convention involving null data as noted in Section 2.

2. Details of the data input scheme

The scheme provides routines for reading from one to nine real variables, 1-dimensional, 2-dimensional and 3-dimensional real arrays, and similar routines for integer variables and arrays. The scheme is based on a routine FFORM which reads in a line of data as individual characters and then builds up the appropriate numerical values. The data input routines (using FFROM) are designed to read numbers in any of the following forms:

Sd, S.d, Sd.d, Sd., SdEd, S.dEd, Sd.dEd, S.dEd

where *d* is a single digit or a compact string of digits, *S* is 0 or more blanks optionally followed by + or − and *E* is an *E*, *E* blank, *E*+ or *E*−. Numbers are normally separated by blank or comma (delimiters). Other forms of number or delimiter will be read and accepted, but a warning message is given so that the user can check that the interpretation taken by the read routine is that intended. Some special characters may also be used without giving warning messages as detailed below. The following additional features have been introduced for convenience in using the system.

(1) Non-numeric comments can be inserted in the data, but a warning will be given if there is no delimiter between comments and numbers, e.g.

1.23, ABC, 2.45

will give no warning, but

1.23ABC2.45

will give two warnings.

(2) Alphanumeric comments may be inserted by enclosing them between quotes (either single or double). In this case the quotes are allowed as a delimiter, e.g.

1.23"AB3"2.45

will give no warning.

Although neither single or double quotes are included in the 1966 ANSI FORTRAN character set, they are permitted in input/output records if they are capable of representation by the processor, and are the most natural characters to use for this function. If strict compliance with the 1966 standard is required, it is recommended that subroutine FFORM be altered so that parentheses are used instead and comments are of the form: (AB3).

(3) Null data are denoted by commas optionally enclosing blanks, e.g. 123,, , 345 gives four numbers: 123, two null data and 345. The null data facility can be particularly useful, for it enables the variables being read to retain their existing values.

It should be noted that the new FORTRAN 77 free-format specification differs from the one used here, in that FORTRAN 77 interprets a comma occurring as the first non-blank character on a line as a null datum only on the first line of data input by a read statement. Hence two lines of data:

,123,345
,567

would be interpreted by the input routines described here as five values: 1 null datum, 123, 345, 1 null datum and 567, whereas FORTRAN 77 would interpret these characters as four values: 1 null datum, 123, 345 and 567.

(4) Multiple data are denoted by *i* * *number* where *i* is a positive integer, e.g. 3*3.14159 means three values of

CCC-0010-4620/81/0024-0256 $01.00

3 14159, 3*, means three null data When * is used in this way no warning is generated

(5) The character ends the read call forthwith and further characters beyond the slash on the same line are ignored Thus can be used for the protection of further data or as a method of retaining the existing values for all further variables of the read call

For all read calls except the 2-dimensional and 3-dimensional array reads $ has the same effect as For 2-dimensional array reads, $ stops the current 1-dimensional sub-array and $$ stops the read For 3-dimensional array reads, $ stops the current 1-dimensional sub-array, $$ stops the current 2-dimensional sub-array and $$$ ends the read Note that $ and / will have no effect if included between quotes

Use of read calls and warnings

The warnings and other features are best illustrated by an example To read 3 numbers into an array PTS at locations PTS(4) to PTS(6) from input device 5 would use

CALL READA (5, IDENT, IERR, PTS, 4, 6)

IDENT is an integer which identifies the read call or if set to zero suppresses the warnings IERR is an integer which records the number of data read during the read call and is set negative if any of the conditions which would cause a warning are met IERR is useful in batch running, IDENT is used to locate suspect data If IDENT = 10 and the characters read from device 5 are

X =3 14 Y= 3E1 Z= 2

then a warning would be given because the number 3 14 starts with an = symbol (only space or , are permitted) The form of the warning would be

***** DATA READ WARNINGS *****

FOR IDENT = 10 AFTER 1 LINE AND 3 CHAR
=3 STARTS NUMBER

X =3 14 Y= 3F1 Z= 2
+

The program would continue running with PTS(4) to PTS(6) set to the values 3 14, 3 0 and 2 0 respectively and IERR set to −3

3. Implementation

The free-format data input scheme presented here has been implemented and tested on a number of computer systems Up to now the routines have been used successfully on the following computers DEC KL10B (RAE Farnborough) IBM 370/168 (UKAEA Harwell), ICL 1906S (RAE Farnborough) ICL (RAE Bedford) and G E. Mk III time sharing service The only modification found to be necessary was on the ICL 4130, where the double quotation character is not allowed in FORTRAN programs (the inclusion of alphanumeric comments can still be achieved, of course, by the use of single quotes)

Experience has shown that these routines require about 3K words of memory and are between 10 and 15 times slower than using the normal FORTRAN formated input Typically, on the DEC KL10, 1000 numbers can be input in less than 10 s

4. Availability

Copies of the program, annotated listings, user's guide, full documentation and a demonstration program can be obtained from the authors

References

ANSI (1966) *Standard FORTRAN*, ANSI X3 9—1966 American National Standards Institute, New York.
ANSI (1978) *Programming Language FORTRAN*, ANSI X3 9—1978 American National Standards Institute, New York

---

Book review

ILP Intermediate Language for Pictures by P J W ten Hagen, T Hagen, P Klint, H Noot, N J Smt and A H Veen, 1981, 110 pages (*Mathematical Centre*, Dfl 14 79)

Almost everyone who has written any graphics software must at some time have devised a picture definition format, and due to the growth in availability of display systems and the frenetic activity of national and international standards organisations in the graphics area, the business of designing what have now (for obscure reasons) come to be known as graphical 'metafile' formats may be considered to have reached epidemic proportions It is refreshing, therefore, to come across a picture language definition which is based on sound principles of language design, and which addresses with meticulous care the many detailed questions of semantic interpretation which are often left vague or ambiguous in other specifications No-one concerned with graphic system design will fail to find many interesting ideas within this slim volume

The ILP picture language is at a higher level than most recently proposed metafile formats, being highly structured with provision for definition and instancing of subpictures An interpreter for the language would therefore require a lot more memory than is usually present in the dedicated microprocessors controlling typical currently available graphical devices A figure of 60K bytes is quoted in the book for an interpreter implemented on a PDP-11, but the interpreter for this language performs many tasks (such as transformation and clipping) which are done at a higher level in conventional systems

A strong feature of ILP is the rigid separation between *attributes* and *picture primitives*, and the attributes themselves are further subdivided into mutually independent and non-interfering classes. Even so, the distribution of space in the book lends weight to the view that in designing a picture language the syntax is easy—it's the semantics that cause all the trouble The complete syntax of ILP is defined in a generously formated 10 page Appendix, whereas almost the whole of Chapter 3 (over 50 pages) is devoted to a detailed discussion of semantic issues Although the overall design aims of the language are described, I felt that more discussion of the choice of individual features of the syntax (which often seem rather arbitrary at first) would have aided comprehension, at least on a first reading

Reference is made to both a compiler and an interpreter of ILP developed at the Mathematical Centre, and also to an embedding of ILP in ALGOL68, but the format produced by the compiler is not described However, the importance of the book lies not in promoting the widespread adoption of ILP, but in illustrating the problems that must be faced in producing a rigorous specification of a powerful and elegant picture description language The experience of the authors should give encouragement to all who wish to see simplicity, economy and elegance prevail over the unstructured morass of conflicting and incompatible graphics systems still being produced

The book should find a place in the library of any institution running an advanced course on computer graphics, but its main benefits will be to graphics system designers (to show them how others have approached the problems they have to grapple with) and to those involved with graphics standards (to persuade them not to set their sights too low)

A C Kilgour (Glasgow)