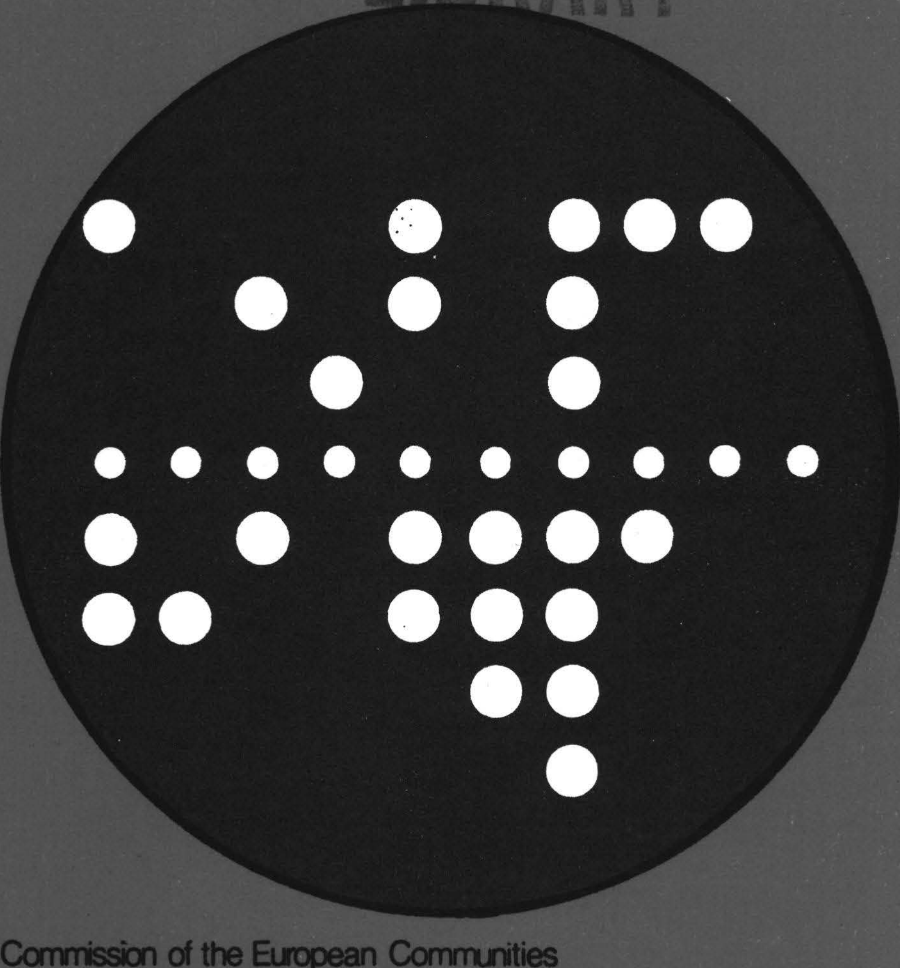
COMPUTING CENTRE NEWSLETTER

October 1978 · No. 25



Commission of the European Communities
JOINT
RESEARCH
CENTRE
Ispra Establishment

CONTENTS

Note of the Editor

The present Newsletter is published monthly except for August and December.

The Newsletter includes:

- Developments, changes, uses of installations
- Announcements, news and abstracts on initiatives and accomplishments.

The Editor thanks in advance those who want to contribute to the Newsletter by sending articles in English or French to one of the following persons of the Editorial Board.

Note de la Rédaction

Le présent Bulletin est publié mensuellement excepté durant les mois d'août et décembre.

Le Bulletin traite des:

- Développements, changements et emploi des des installations
- Avis, nouvelles et résumés concernant les initiatives et les réalisations.

La Rédaction remercie d'avance ceux qui veulent bien contribuer au Bulletin en envoyant des articles en anglais ou français à l'un des membres du Comité de Rédaction.

Computing Centre References

		Room	Tel.
Manager	J. Pire	1816	732
Adjoined	G. Gaggero	1874	787
Computer Room	P. Tomba	1857	797
Adjoined	A. Binda	1857	797
Peripherals	G. Nocera	1825	767
System Group	D. Koenig	1839	742
Adjoined	P.A. Moinil	1841	704
Informatics Support	G. Gaggero	1874	787
o General Information	G. Hudry	1873	787
o Program Information Service	G. Gaggero	1874	787
Adjoined	S. Leo Menardi	1884	721
o Graphics and Support to Users	H.I. de Wolde	1883	1259
Adjoined	A. Pollicini	1886	701
Application Packages	A. Inzaghi	1887	755

Editor : Jean Pire
Layout : Paul De Hoe
Graphical and Printing Workshop, JRC Ispra

A LOOK AT FORTRAN 77 *

A.A. Pollicini

Introduction

The progress in technology is continually providing us with new tools. One such tool is the electronic computing machine. The main characteristic of such a machine is the ability to perform at very high speed some basic operations on electric signals that represent information.

Electronic data processing is based on the binary logic, therefore the atom of information is the binary digit commonly known by the contracted term bit. Once more human intelligence proved its multiform imagination, in building with this unique type of brick a number of different architectures that range, for instance, from 8 bits to 60 bits addressable units and using 3, 4, 6, 7 or 8 bits to represent respectively 8, 16, 64, 128 or 256 external symbols.

The way in which computing machines are actually used is to submit to them a coded description of an algorithm that is expected to solve a specific problem. Because there is a great distance between our natural languages and machine codes, we currently use an intermediate means, called programming language to provide such descriptions.

In practice a computer manufacturer provides users with a processor constituted by the definition of the language plus the compiler that translates the language on and for his machine. It is evident that each manufacturer exploits all features of his own machine architecture in the most effective way aiming at the enhancement of the performances of his machine. This is the reason for all software incompatibility and of the difficulties in moving pieces of software.

The direct approach to software portability should be to design a language for a virtual machine making abstraction from the architecture and then to provide a different interpreter for its code on each computer. This is indeed done in some problem oriented systems. But the widely used general purpose languages existed before the need for portable software, therefore an a posteriori remedy was necessary: a standard definition of the language.

---

* *Reprint of the lecture: Programming Language Standards given during the ISPRA-COURSE on PROGRAM LIBRARY AND INFORMATION SERVICE TECHNIQUES. 17 – 20.10.1978*

The languages themselves may provide features to support some abstractions. A feature very helpful towards portability is abstract data type, then the standardization of languages with strong typing does not involve heavy constraints and may be promoted by the author, an implementor or a user's group.

On the contrary, for languages designed during the 60's and supported by many manufacturers in their own fashion, the task of standardization had to be carried out by official bodies at national and international level.

A programming language standard is a text that defines a set of accepted features and gives a syntactic description for them, as far as possible in a clear and unequivocal way.

Once approved, such a text should become a constraint both for program developers and computer manufacturers, in the sense that a program is said to be **standard conforming if all language features it uses are included in the standard**, while a **processor** is said to be **standard conforming if it includes all the features of the standard** as a subset.

This attitude often results in a trap for the naive user of a standard conforming processor.

The scope of the present analysis will be restricted to the FORTRAN language, because a great effort was focused on its standardization, mainly by the works of the American National Standards Institute.

ANSI FORTRAN STANDARDS: the path from 1966 to 1978

The American Standards Association (successively renamed United States of America Standards Institute mid 1966 and American National Standards Institute in 1969) approved in March 1966 two FORTRAN IV standards. ANS FORTRAN [1] and ANS BASIC FORTRAN [2] as a subset of the full language.

Two clarifications were issued in 1969 [3] and 1971 [4]. Outside U.S.A. an ECMA Standard on FORTRAN [5] was issued in 1965 that ranges between the two ANS standards.

In 1972 an ISO recommendation for FORTRAN [6] presented three levels of the language that followed, with few secondary changes, the previous standards giving them an international character.

It is admitted that an ANS standard looses validity after a five year period unless a revision procedure has started before the period is expired.

The revision took a long period, during which several working documents were published.

The X3J3 Committee issued a draft proposed ANS FORTRAN [7] in 1976 for public review and comment. After examination of the public comments received, the Committee edited in 1977 [8] a new text defining the full language and a subset to replace respectively ANS FORTRAN and ANS Basic FORTRAN. The new language has been given the informal designation FORTRAN 77.

During the ballot period, few final amendments were adopted and reported in a further working document [9].

Finally, the Board of Standards Review approved the new standard on April 3, 1978 and the official edition was announced [10].

An overview of the differences

Since only major differences wil! be dealt with, a functional grouping of arguments is adopted rather than the structural classification of the standard.

Most of the arguments are innovations introduced by the new standard.

There are two kinds of innovations:
1) features that are already present in some existing processors as extensions;
2) features that are a novelty for the language at all.

No distinction is made in the descriptions that follow, concerning the kind to which each feature belongs, but some concluding remarks will be made, which issue from this fact. On the contrary, the restrictions are very limited if we consider the many arguments the actual standard covers.

Indeed, the X3J3 Committee was specially sensible to the need to preserve compatibility with the previous standard (but this does not mean with the extensions of the existing compilers). Since each restriction is going to cause existing applications to come into conflict with compilers that will conform to the new standard, the fact is explicitly considered in one of the appendices that integrate the text of the standard.

Indeed, Appendix A lists those changes that are known to cause conflict and states that they were retained "... only ... to correct an error in the previous standard or to add to the power of the FORTRAN language in a significant manner."

I Elements of the language

a) The character set
   - There are two new special characters: apostrophe (') and colon (:)
   - A partial definition of the collating sequence is introduced as follows:

- the blank character is less than any digit and letter.
- letters are in alphabetical order and A is less than Z.
- digits are in increasing order and 0 is less than 9.
- digits and letters must not be intermixed, but the precedence of the two classes of characters is free.
- the position of special characters, other than blank, is left free.

b) Lines
- There are three types of comment lines:
  - lines with the character C in column 1 (as in 1966 text).
  - lines with the character * in column 1 (new).
  - lines with blank characters in columns 1 to 72 (**conflict** with 1966 text. Formally, such a blank line resulted the initial line of a statement).
- Continuation lines must contain blank characters in columns 1 to 5 and any character other than blank or zero in column 6 (**conflict** with 1966 text. Columns 1 to 5 of continuation lines could contain any character with the only limitation that column 1 must not contain the letter C that meant comment line).

Remark: What was previously defined as END line is considered as END statement by the new text. This implies that some additional functions are tied to the keyword END (see paragraph VIII - Program units), but does not represent a conflict.

II Data types and specification statements

a) Type CHARACTER

This is the most significant improvement of the typing features of the language. This innovation also involves changes in other aspects of the language. Such related changes will be reported in the following paragraphs.

- a character constant is constituted as a (non empty) string of **any character representable by the processor** and must be enclosed in apostrophes.

  The blank character is significant. Any significant apostrophe within the string must be represented as two consecutive apostrophes.
- the Hollerith data type expressed as

  $$nHc_1c_2 \dots c_n$$

  is no longer valid (**conflict with 1966 text**).

  The committee, being conscious that Hollerith constants could be retained by some manufacturers as an extension, gives in Appendix C some recommendations for the uniformity of such an extension.

- The type CHARACTER allows the representation of strings of characters of fixed declared length.

  The following are examples of type CHARACTER statements:

  CHARACTER * 3 MONTH (12)
  MONTH is an array of 12 elements each consisting of three characters.

  CHARACTER * 6 MON,TUE*7,WED*9,THU*8,FRI,SAT*8,SUN
  the listed variables are specified for string of different lengths. The specific length declarators, where present, override the general length declarator, thus:
  the strings MON,FRI,SUN are 6 characters long.
  the string TUE is 7 characters long.
  the strings THU and SAT are 8 characters long.
  the string WED is 9 characters long.

  CHARACTER LETTER (26)
  LETTER is an array of elements of 1 character, indeed in absence of any length declaration the default length is one.

  CHARACTER TITLE (*)
  TITLE as appears in the statement may be used as dummy argument in a subprogram. Then the length of the associated actual argument will apply.

- The allocation of strings requires **one character storage unit** for each character in the string.

  This specification allows compilers the freedom for architecture-oriented allocation and effective core management. But, all other data types are allocated in **numeric storage units** that is storage words. This fact entails a physical separation in core between character strings and data of different types. As a consequence, the EQUIVALENCE and COMMON statements may be used for character entities only with homogeneous lists of variable names.

- The operations on strings are:
  - concatenation allowed by the operator // (two consecutive slashes) used in dyadic expressions of infixed notation.
  - substring reference allowed by the use of a variable name or array element name of type character followed by two integer expressions enclosed in parentheses and separated by a colon.
    The values of the two expressions give the position in the string of respectively the first and the last characters of the substring.

- inquiry for the length of a given string allowed by the intrinsic function LEN (S).
- inquiry for the occúrrence of a given substring within a given string allowed by the intrinsic function INDEX (S, SUBS).
- inquiry for the position of a given character in the collating sequence allowed by the intrinsic function ICHAR (C).
- inquiry for the character corresponding to a given position within the collating sequence allowed by the intrinsic function CHAR (I).

Remark: String manipulation will no longer constitute a dramatic portability dilemma for the Fortran programmer. In effect, the representation of characters according to the previous standard and in such a way that programs may be transferred on different computers, implied the use of one storage word per character that was a considerable waste of core.
On the other hand, the use of manufacturer provided features as well as programming tricks to get a more effective installation dependent code, implied high conversion cost to run on different installations.
It must be noted that software belonging to this second category will incur the same troubles also with the new standard.

b) IMPLICIT statement

- the implicit integer or real type attribution tied to the formation rule of symbolic names, can be confirmed or overridden by an IMPLICIT statement that has the form:

  IMPLICIT type (range$_1$ [, range$_2$] ...)

  where type may be one of the allowed data types. (In the case of character entities type may include the length declarator and assumes the form CHARACTER [*len]). Each range may be either a single letter or the extremes of a series of contiguous letters in the form $C_1 - C_n$. All ranges must appear in alphabetic order.
- the specifications of an IMPLICIT statement apply within the scope of a program unit, but more than one such statement may appear in the same unit provided that no intersection occurs among different ranges.
- this statement must precede all other specification statements, except PARAMETER statements, within a program unit.
- any type statement may override the specification of an IMPLICIT statement for any particular symbolic name.
- IMPLICIT statements do not affect any intrinsic function name.

c) PARAMETER statement

- the PARAMETER statement gives a constant a symbolic name and has the form:

  PARAMETER (name1=expres1[,name2=expres2] ...)

- each name must match in type with the associated expression and must be unique within the program unit.
- names that are not of implied type integer or real must be previously defined in type.
- the primaries of the expressions must be constants or parameters previously defined.
- such symbolic names can be used in any subsequent statement of the program unit as primary in an expression or DATA statement, but cannot be used in format specifications nor as part of other constants.

d) INTRINSIC statement

- this statement has the form:

  INTRINSIC fun1 [,fun2] ...

  Each fun is the name of an intrinsic function. If an intrinsic function name is used as actual argument in a program unit, it must appear in an INTRINSIC statement in that program unit.
- the names of intrinsic functions for type conversion and for choosing the largest or smallest value, must not be used as actual argument.

e) EXTERNAL statement

- the appearence of an intrinsic function name in an EXTERNAL statement inhibits the reference to the intrinsic function in the program unit and a homonymous external procedure must be provided. This is partially in **conflict** with 1966 text in the sense that all names referred to as **basic external functions** by the previous standard are now included in the table of **intrinsic functions**.
- a function name must not appear in both INTRINSIC and EXTERNAL statement.

f) Generic names for intrinsic functions

- if the operations performed on a set of arguments by an intrinsic function are significant even if the set may belong to different data types, a generic name may be used to invoke such a function. The suitable function will be selected according to the type of the actual arguments.
- if intrinsic function names are used as actual argument, they must be specific names.

- the appearance of a generic name in type statement does not remove the generic property.

III DATA initialization

a) DATA statement
   - in addition to variable and array element names (already allowed by the previous standard), also array names, substring names and implied-DO lists may receive initial values in a DATA statement.
   - in the constant list either constants either symbolic names for constant (see PARAMETER statement) may be used both for the values and the repetition counters.
   - integer constant expressions are allowed both for subscript and substring expressions.
   - in the case of character and logical entities, the name and the corresponding constant must be of the same type. For the other data types, conversion is allowed as for arithmetic expressions (see paragraph V).

IV Arrays

a) Dimensions of an array
   - an array may have up to 7 dimensions.
   - for each dimension, a lower bound and an upper bound are allowed. In this case the array declarator assumes the form:

     name (dl_1 : du_1 [,dl_2 : du_2] ...)

     the lower bound may be omitted and in this case the value 1 is assumed.
   - if the array name is not a dummy argument, the dimension bounds are integer constant expressions.
   - if the array name is a dummy argument, the dimension bounds may be integer expressions and, in addition, the upper bound of the last dimension may be an asterisk to mean an assumed-size array.
   - only one restriction is made on the values of the dimension bounds, that is the upper bound must not be less than the lower bound. Negative and null values are accepted.

b) Reference to an array element
   - to reference an array element, the array name must be followed by a subscript of the form:

     (s_1 [, s_2] ...)

   - each entity separated by comma is called a subscript expression and must be an integer expression. Array element references and function references are allowed within subscript expressions.

- the number of subscript expressions must always be equal to the number of dimensions of the corresponding array declarator.

  This is in **conflict** with 1966 text that allowed:

  ```
  DIMENSION A (10,10),B(50)
  EQUIVALENCE (A(51),B(1))
  ```

- the value of any subscript expression must not be greater than the upper bound for corresponding dimension. This is a welcome **conflict** with 1966 text that allowed any subscript for which the successor function returned a value that did not exceed the array size.

  For instance:

DIMENSION A(2,3,5)	size of A = 30
10 B=A(4,4,1)	the 10th element
20 C=A(1,4,5)	the 31th element

  in which, after evaluation of the successor function

  $$f=S1+D1(S2-1)+D1 \cdot D2(S3-1)$$

  statement 10 resulted legal, while statement 20 was illegal.

V EXPRESSIONS

a) generalization of expression use
   - it is in the spirit of the new standard to allow expressions where formerly variables were required.

b) mixed mode expressions
   - in arithmetic expressions with more than one operand, if there are operands of different type, conversions from type to type are allowed following the criterion that the absorbing power of type is in the order:

     COMPLEX, DOUBLE PRECISION, REAL, INTEGER.

     However, combinations of COMPLEX and DOUBLE PRECISION are not allowed.

The following table shows the situation for the arithmetic operators + – * /, while a more complicated rule applies to the exponentiation operator.

x_1 \ x_2	I_2	R_2	D_2	C_2
I_1	$I = I_1$ op I_2	R=REAL(I_1)opR_2	D=DBLE(I_1)opD_2	C=(CMPLX(REAL(I_1)) op C_2)
R_1	$R=R_1$ opREAL(I_2)	$R = R_1$ opR_2	D=DBLE(R_1)opD_2	C=CMPLX(R_1)opC_2
D_1	$D=D_1$ opDBLE(I_2)	$D=D_1$ opDBLE(R_2)	$D = D_1$ op D_2	Prohibited
C_1	C=(C_1 op CMPLX(REAL(I_2)))	$C=C_1$ opCMPLX(R_2)	Prohibited	$C = C_1$ op C_2

*Type of the result of (x_1 op x_2) applicable when op is + ; – ; * ; /*

c) logical expressions
- two logical operators were added:
  .EQV. that gives value TRUE for matching values of the operands and FALSE otherwise
  .NEQV. that gives value TRUE for discordant values of the operands and FALSE otherwise
- the precedence order of the logical operators is: .NOT.,.AND.,.OR.,(.EQV. and .NEQV.)

VI Control statements

a) Block-IF and related statements
This is the only issue of the modern concepts on control structuring that was retained by the committee.
- the syntax for this control structure is:

```
 IF (e) THEN
   [statements]
[ELSE IF (e) THEN]
[  [statements]  ]
[....            ]
[ELSE        ]
[  [statements]]
 END IF
```

- any number of ELSE IF statements may appear after the block-IF statement and before the ELSE statement (if present) or the END IF statement.

- all blocks of statements may be empty.
- the transfer of control to the statements in a block-IF is only possible by entering the IF statement.

b) DO-loop
- the DO-variable may be of type integer, real or double precision as well as the initial, terminal and incrementation parameters are allowed to be arithmetic expressions of type integer, real or double precision. The expressions are evaluated and converted according to the type of the DO-variable at entering the DO statement, therefore assignments to variables in these expressions, during loop execution does not affect the control.
- the incrementation parameter may have a positive or a negative but not zero value. Initial and terminal parameters may have any value. The iteration count is established before iteration, and if zero the loop is not executed as in the classic WHILE FALSE situation.
- the DO-variable remains defined at exiting from the loop, whenever it happens by jumping outside or at completion of the iterations. (Conflict with 1966 text).
- the extended range of a DO-loop is no longer admitted and this conflicts with 1966 text.
  At this subject it is worthwhile noting that those preprocessors implementing internal procedures will incur troubles as shown in the following example. (Fig. 1).

c) Computed GO TO
- the transfer of control depends on the value of an integer expression instead of an integer variable.
- if the control expression is out of range, the execution goes in sequence.

VII Input/Output statements

In addition to the already existing file positioning statements BACKSPACE, ENDFILE, REWIND, three new auxiliary input/output statements, OPEN, CLOSE and INQUIRE were added. Their purpose is to allow for new flexibility in program-periphery communications.
External files as currently identified by unit in FORTRAN IV, are now said to be preconnected and can still be accessed as before, without any need of the new statements (to guarantee compatibility). Otherwise connection between an external file and a unit can be controlled by opening and closing it explicitly as well as its actual properties may be inquired at any moment.

ACCOUNTED WORK UNITS TABLE FOR ALL JOBS OF THE GENERAL SERVICES - Monthly and Cumulative Statistics

	January	February	March	April	May	June	July	August	September	October	November	December
Year 1977	44	74	78	32	26	36	27	25	27	31	40	34
accumulation	44	118	196	228	254	290	317	342	369	400	440	474
Year 1978	51	43	55	50	49	74	36	31	33			
accumulation	51	94	149	199	248	322	359	391	424			

ACCOUNTED WORK UNITS TABLE FOR THE JOBS OF ALL THE OBJECTIVES AND GENERAL SERVICES - Monthly and Cumulative Statistics

	January	February	March	April	May	June	July	August	September	October	November	December
Year 1977	135	218	312	193	180	269	244	196	277	275	284	179
accumulation	135	353	665	858	1038	1307	1551	1747	2024	2300	2584	2763
Year 1978	211	213	283	232	202	317	230	270	240			
accumulation	211	424	707	939	1141	1,458	1688	1958	2.198			

ACCOUNTED WORK UNITS TABLE FOR THE JOBS OF THE EXTERNAL USERS - Monthly and Cumulative Statistics

	January	February	March	April	May	June	July	August	September	October	November	December
Year 1977	13	14	18	16	13	22	19	18	27	25	21	20
accumulation	13	27	45	61	74	96	115	133	160	185	206	226
Year 1978	12	10	11	46	23	11	9	5	12			
accumulation	12	22	33	79	102	113	123	128	140			

EQUIVALENT TIME TABLE FOR ALL JOBS OF ALL USERS - Monthly and Cumulative Statistics

	January	February	March	April	May	June	July	August	September	October	November	December
Year 1977	158	241	314	242	202	294	266	217	299	299	318	235
accumulation	158	399	713	955	1157	1451	1717	1934	2233	2532	2850	3085
Year 1978	276	261	356	298	262	335	245	297	267			
accumulation	276	537	893	1191	1453	1.788	2.033	2.330	2.597			

Utilisation of computer centre by the objectives and appropriation accounts for the month of September 1978

		IBM 370/165 equivalent time in hours
1.20.2	General Services - Administration - Ispra	28.78
1.20.3	General Services - Technical - Ispra	4.67
1.30.4	L.M.A.	0.15
1.90.0	ESSOR	24.22
1.92.0	Support to the Commission	10.04
2.10.1	Reactor Safety	116.12
2.10.2	Plutonium Fuel and Actinide Research	11.53
2.10.3	Nuclear Materials	0.95
2.20.1	Solar Energy	–
2.20.2	Hydrogen	–
2.20.4	Design Studies on Thermonuclear Fusion	3.14
2.30.0	Environment and Resources	14.97
2.40.0	METRE	1.27
2.50.1	Informatics	29.87
2.50.2	Training	0.06
2.50.3	Safeguards	2.57
309	Programming Support	–
	TOTAL	239.79
1.94.0	Services to External Users	12.38
	TOTAL	252.17

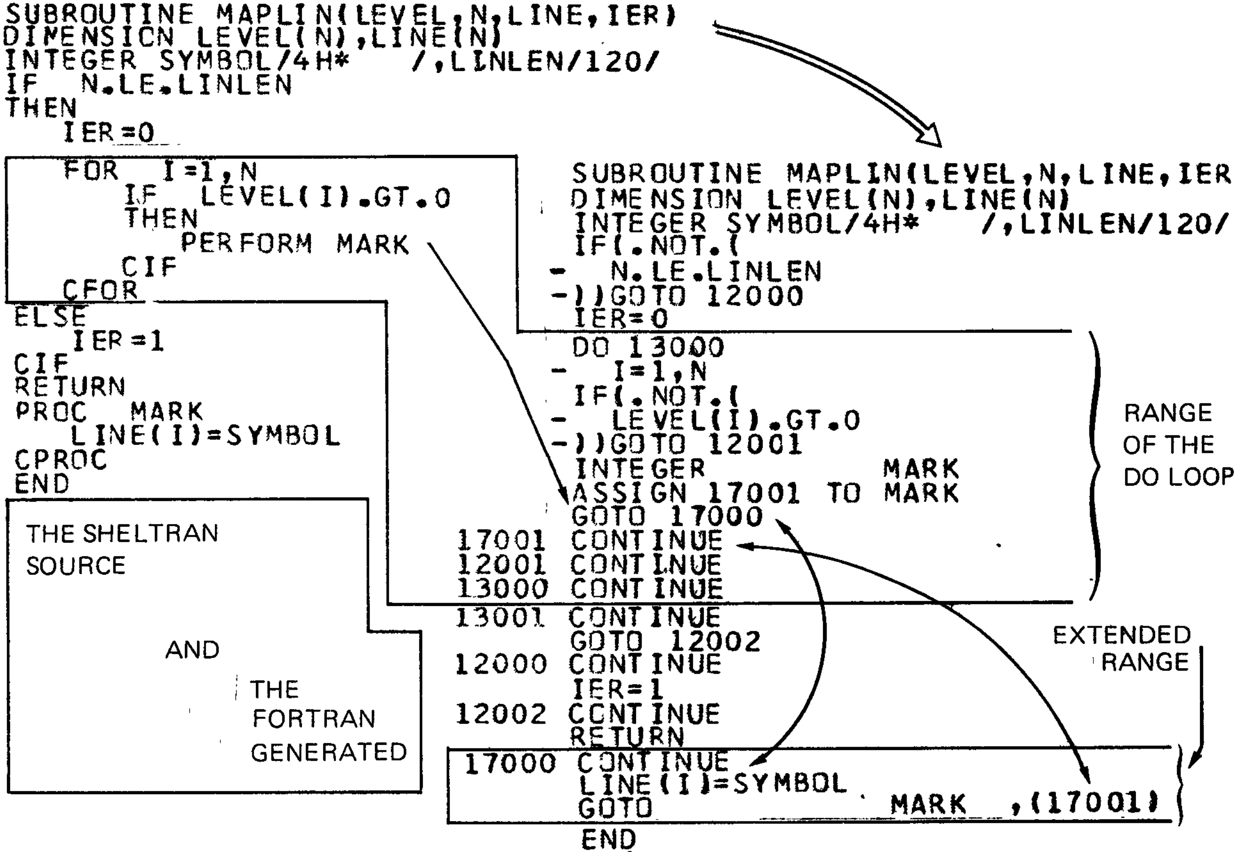

SUBROUTINE MAPLIN(LEVEL,N,LINE,IER)
DIMENSION LEVEL(N),LINE(N)
INTEGER SYMBOL/4H* /,LINLEN/120/
IF N.LE.LINLEN
THEN
IER=0
FOR I=1,N
IF LEVEL(I).GT.0
THEN
PERFORM MARK
CIF
CFOR
ELSE
IER=1
CIF
RETURN
PROC MARK
LINE(I)=SYMBOL
CPROC
END
THE SHELTRAN SOURCE
AND
THE FORTRAN GENERATED
SUBROUTINE MAPLIN(LEVEL,N,LINE,IER)
DIMENSION LEVEL(N),LINE(N)
INTEGER SYMBOL/4H* /,LINLEN/120/
IF(.NOT.(
- N.LE.LINLEN
-))GOTO 12000
IER=0
DO 13000
- I=1,N
IF(.NOT.(
- LEVEL(I).GT.0
-))GOTO 12001
INTEGER MARK
ASSIGN 17001 TO MARK
GOTO 17000
17001 CONTINUE
12001 CONTINUE
13000 CONTINUE
13001 CONTINUE
GOTO 12002
12000 CONTINUE
IER=1
12002 CONTINUE
RETURN
17000 CONTINUE
LINE(I)=SYMBOL
GOTO MARK ,(17001)
END
RANGE OF THE DO LOOP
EXTENDED RANGE

FIG. 1

a) OPEN statement

- it has the form:

  OPEN (olist)

  in which olist is a list of specifiers from the following sequence:

[UNIT=] u	to specify one external unit. It is mandatory.
IOSTAT=ios	ios is an integer variable that returns a value at completion of the execution of the statement. (Zero means successful operation).
ERR=label	to monitor error conditions.
FILE=fname	name of the file to be connected to the external unit.
STATUS=sta	to specify whether the file is NEW, OLD or SCRATCH.
ACCESS=acc	acc may be SEQUENTIAL or DIRECT.
FORM=fm	fm may be FORMATTED or UNFORMATTED.
RECL=rl	rl is an integer expression whose value specifies the size of all records; rl applies only to direct access and must be positive.
BLANK=blnk	blnk may be NULL to ignore blanks in numeric fields or ZERO to accept them as zeros.

b) CLOSE statement

- it has the form:

  CLOSE (cllist)

  in which cllist is a list of specifiers from the following sequence:

[UNIT=] u	to recall the external unit. It is mandatory.
IOSTAT=ios	as in OPEN
ERR=label	as in OPEN
STATUS=sta	to specify whether the final disposition has to be KEEP or DELETE.

c) INQUIRE statement

- this statement may be used in two ways: either to inquire about properties of a named file, or of the connection to an external unit.
- the form "by file" is:

  INQUIRE (iflist)

  where iflist must contain the specifier FILE=fname and may include other specifiers from an extended list.

- the form "by unit" is:

  INQUIRE (iulist)

  where iulist must contain the UNIT specifier and may include some specifiers of the list.
- at completion of the execution the variables coupled to those specifiers having an actual meaning, become defined.

d) Data transfer statements
- the transfer of data is performed by the statements:

  READ (cilist) [iolist]

  WRITE (cilist) [iolist]

  The control information list cilist must contain one or more of the following specifiers:

[UNIT=] u	to refer to a unit. It is mandatory and u may be: - an integer expression to identify an external unit. - a symbolic name of a character entity that identifies an internal file.
[FMT=] f	to provide formatted control. The f identifier may be: - the label of a FORMAT statement or an integer variable to which the label value has been assigned. - a character array name that contains the format specifications or a character expression that gives them directly. - an asterisk to mean list-directed input/output.
REC=rn	to point to the requested record in direct access method. rn is an integer expression that must have a positive value.
IOSTAT=ios	to return in ios an integer value depending on whether completion was successful (zero) or not.
ERR=label	to monitor I/O errors.
END=label	to monitor end file condition.

  The input/output list iolist is a list of items to or from which data are to be transferred. Within output lists any type of constant or expression are permitted.
- the short forms:

  READ f [,iolist]

  PRINT f [,iolist] are allowed.

- the possibility of specifying an internal file as unit allows for data transfer and conversion from storage to storage without physical transmission of information. This facility is limited to sequential access and formatted control. In addition, auxiliary input/output statements must not specify internal files.
- the list-directed input/output supplies formatted control without any need of FORMAT specifications. Data are converted in accordance to the type of the items in the input/output list. The values are delimited by separators (comma or slash or blank). Blanks cannot be used as zero and embedded blanks are significant only within character constants. Complex are enclosed in parentheses.

e) FORMAT specifications
  - apostrophes editing may be used in alternative to Hollerith editing.
  - some format descriptors were introduced for tabulation editing with left and right alignement (P, PL, PR).
  - there are new format descriptors (SP, SS, S) for sign control.
  - embedded and trailing blanks in numeric fields are removed or converted to zeros according to the specification BN or BZ.
  - a colon edit descriptor may terminate format control if the items of input/output list are exhausted.
  - the exponent field in E and D descriptors must be preceded by the sign (**conflict** with 1966 text that allowed blank to be interpreted as a plus sign).

VIII Program units

a) Program segmentation
  - a name may be assigned to the main program, using the optional statement

    PROGRAM name

    that must be the first one of the main.
  - also BLOCK DATA subprogram may have a symbolic name.
  - every program unit must terminate with an END statement. If executed in a subprogram it implies a RETURN and a STOP in the main program.

b) Multiple entries
  - one or more ENTRY statements may appear in a FUNCTION or SUBROUTINE subprogram.
  - such entries are referenced by name or by a CALL statement depending on whether present in a FUNCTION or SUBROUTINE subprogram.

c) Alternate returns

This is the most regrettable extension because it allows for more intricate control paths.

- the RETURN statement may contain the selection of a variable point within the calling unit, to which the control is to be transferred.

  RETURN [e]

  where e is an integer expression.
- in this case, one or more dummy arguments in the SUBROUTINE or ENTRY statement are asterisks, to which are associated actual arguments of the form *label.

d) Definition status of entities

- the execution of a RETURN statement or END statement within a subprogram causes the entities referenced by the subprogram to become undefined except entities:
  - in blank common
  - in a named common block which also appears either in the main program or in another program unit that is referencing the subprogram directly or indirectly.
  - initially defined and that never changed their status.
  - appearing in a SAVE statement.
- the SAVE statement has the form:

  SAVE [a[,a] ...]

  each a may be a common block name enclosed in slashes, an array name or a variable name.

  It is to be noted that entities in a named common block and saved by a program unit, might become undefined in another program unit referencing the same common block and lacking in the corresponding SAVE statement.

COMPATIBILITY AND INCOMPATIBILITY WITH EXISTING COMPILERS

The 1966 standards reflected the intent of recognizing a subset belonging to several implementations rather than of giving uniformity to divergent extensions implemented by different processors.

On the contrary, during the revision period, X3J3 acted almost as a FORTRAN development committee and featured the definition of a truly new

FEATURE	CDC 6000 V3 7600 V1 EXT.FORT.	C II FORTRAN IV ETENDU	HONEYWELL 600/6000 FORTRAN	IBM 360,370 FORTRAN IV (G and H)	SIEMENS 4004 FORTRAN IV	UNIVAC 1100 FORTRAN V
Type CHARACTER	–	–	YES	–	–	–
IMPLICIT	–	YES	YES	YES	YES	YES
PARAMETER	–	–	YES	–	–	YES
INTRINSIC	COMPLETELY NEW FEATURE					
Generic name	–	–	YES	–	YES	YES
Mixed mode	YES	YES	YES	YES	YES	YES
EQV.,.NEQV.	COMPLETELY NEW FEATURE					
IF-THEN-ELSE	COMPLETELY NEW FEATURE					
Comp. GOTO	–	YES	–	YES	YES	–
OPEN, CLOSE INQUIRE	COMPLETELY NEW FEATURE					
ERR, END options	–	not general	YES	not general	not general	ERR only
Intern. File	ENCODE DECODE	ENCODE DECODE	ENCODE DECODE	–	–	ENCODE DECODE
Direct Access	–	READ/WRITE DISK & DRUM	CALL RANSIZ	DEFINE FILE	DEFINE FILE	DEFINE FILE
Short form READ,PRINT	YES	–	YES	YES	YES	YES
List-dir. I/O	–	INPUT OUTPUT	nFORMAT(v)	–	–	–
New FORMAT descriptors	partly	partly	partly	partly	partly	partly
PROGRAM st.	YES	–	–	–	YES	–
ENTRY st.	without parameter	YES	YES	YES	YES	YES
Alternate Returns	RETURNS (list)	YES	YES	YES	YES	YES
SAVE	COMPLETELY NEW FEATURE					

language, considering compatibility only with respect to the 1966 standard. Indeed, FORTRAN 77 includes facilities already provided by some processors, but in many cases the syntactic form is quite different, therefore most of the existing FORTRAN programs will fall into incompatibility with the new standard, even if the techniques they apply are now retained by the new standard.

Less compatibility problems may instead be expected from those features that are real innovations introduced by the standard, since they will be obviously implemented in a standard conforming way by forthcoming processors.

To warn users against some of the incompatibilities that will arise, the next table gives a map of major innovations versus six widely used processors. This table is based on a comparative study carried out at Ispra on the referred FORTRAN compilers [11].

ANNOUNCEMENTS OF COMPILERS IMPLEMENTING FORTRAN 77

Information about compilers for FORTRAN 77 are beginning to circulate.

- A full language FORTRAN 77 ANSI conforming implemented by Tandem computers.
- A FORTRAN 77 implemented on Honeywell 6000 series by Lahey Computer Systems.
- A portable and complete FORTRAN 77 on UNIX systems implemented by Bell Laboratories.
- An extension of subset FORTRAN 77, auspiciously named "FORTRAN 80" developed by INTEL.

were reported last July by FOR-WORD [12] and next publication of further announcements is foreseen.

REFERENCES

[1] American National Standards Institute
ANS FORTRAN (ANS X3.9–1966)

[2] American National Standards Institute
ANS Basic FORTRAN (ANS X3.10–1966)

[3] Clarification of FORTRAN Standards - Initial Progress
Comm ACM 12,5 (May 1969) pp. 289–294

[4] Clarification of FORTRAN Standards - Second Report
Comm ACM 14,10 (Oct. 1971) pp. 628–642

[5] European Computer Manufacturers Association
ECMA Standard on FORTRAN (ECMA-9) 1965

[6] International Organization for Standardization Programming Language FORTRAN ISO recommendation No. 1539 (1972)

[7] Draft proposed ANS FORTRAN (X3J3/76)
Sigplan Notices 11,3 (Mar. 1976) special issue

[8] Draft proposed ANS FORTRAN (X3J3/90)
Working document (20.6.1977)

[9] Amendments to ANS FORTRAN (X3J3/97)
Working document (6.10.1977)

[10] American National Standards Institute
ANS Programming Language FORTRAN X3.9-1978.
(Revision and Consolidation of X3.9-1966 and X3.10-1966)

[11] Prinzivalli G., Studio di problemi inerenti alla portabilità di programmi in linguaggio FORTRAN
Tesi di Laurea in Matematica (1976), Università di Torino.

[12] Meissner L.P. (editor), FOR-WORD. FORTRAN Newsletter
Vol. 4, No. 2 (July 1978) p.7

The Newsletter is available at:

Mrs. A. Cambon
Support to Computing
Bldg. 36 - Tel.730

Des exemplaires du Bulletin sont disponibles chez:

Mme A. Cambon
Support to Computing
Bât. 36 - Tel. 730

Les personnes intéressées et désireuses de recevoir régulièrement "Computing Centre Newsletter" sont priées de remplir le bulletin suivant et de l'envoyer à

Mme A. Cambon
Support to Computing
Bât. 36, Tel. 730

Nom ....

Adresse ....

....

Tel. ....

The persons interested in receiving regularly the "Computing Centre Newsletter" are requested to fill out the following form and to send it to:

Mrs. A. Cambon
Support to Computing
Building 36, Tel. 730

Nom ....

Address ....

....

Tel. ....