CANADA. INLAND WATERS DIRECTORATE
REPORT SERIES.
#1

CANADA

C. C. I. W.
LIBRARY

REPORT SERIES No. 1

PLAIN FORTRAN

A guide to compatibility in computer programming

J. J.THERRIEN

INLAND WATERS BRANCH
DEPARTMENT OF ENERGY, MINES AND RESOURCES
OTTAWA, CANADA, 1968

GB
2429
C27
no. 1

CANADA

REPORT SERIES No. 1

PLAIN FORTRAN

A guide to compatibility in computer programming

J. J.THERRIEN

INLAND WATERS BRANCH
DEPARTMENT OF ENERGY, MINES AND RESOURCES
OTTAWA, CANADA, 1968

TABLE OF CONTENTS

Page

PREFACE iv
INTRODUCTION 1
Phase I 2
Phase II 2
Phase III 2
Phase IV 3
1. Names of Variables, Functions and Subprograms 3
2. Types of Variables and Functions 3
3. Types of Constants 3
4. Statement Numbers 3
5. Subscripts 3
6. Expressions 4
7. Character Data (A-Format code) 6
8. Assignment (=) Statement 6
9. GØ TØ Statement 6
10. IF Statement 7
11. DØ Statement 7
    11.1 By-Passing the DØ Loop 8
    11.2 The Value of the Loop Index 8
    11.3 Transfer of Control 8
    11.4 Changing the Value of the Index and Parameters 9
12. CØNTINUE Statement 9
13. STØP Statement 9
14. END Statement 9
15. Input-Output List 9
16. FØRMAT Statement 10
    16.1 Format Codes 10
    16.2 FORTRAN Records 10
    16.3 Brackets Within the Format 10
    16.4 Carriage Control Characters 11
    16.5 Correspondence of List and Format Items 11
    16.6 Allowable Characters with H and A Format Codes 11
    16.7 Allowable Characters with I, F and E Format Codes 11
17. READ Statement 11
    17.1 Read With Format (BCD data) 11
    17.2 Read Without Format (Binary data) 12
18. WRITE Statement 12
    18.1 Write With Format (BCD data) 12
    18.2 Write Without Format (Binary data) 12
19. BACKSPACE Statement 13
20. REWIND Statement 13
21. DIMENSIØN Statement 13
22. CØMMØN Statement 13
23. SUBRØUTINE Statement 14
24. FUNCTIØN Statement 14
25. CALL Statement and Function Reference 14
        Examples: 1, 2, 3 and 4 15
        Restrictions: 25.1 to 25.5 16
26. RETURN Statement 17
27. Library Functions 17
28. Range and Precision of Numbers 17
29. Size of Constants 18
30. Double Precision on the IBM 360 18
31. Features to be Ignored 19

PREFACE

"PLAIN FORTRAN" is a restricted but compatible FORTRAN which is intended for the scientist who does not wish to get involved in the direct comparison of different versions of FORTRAN to determine how compatible they are. It is closer to FORTRAN II than it is to FORTRAN IV and it ensures the compatibility of programs on medium and large scale computers. It has been used successfully to run programs on the CDC 3100, CDC 3300, CDC 6400, UNIVAC 1108, IBM 7040 and IBM 360. The programs produced consistent results on all these machines without changes in the source programs, except in some cases where minor changes had to be made to use double-precision on the IBM 360.

PLAIN FORTRAN will make programs truly compatible. Not only will different compilers accept the source programs without printing error messages but most important of all, the compiled programs will produce consistent results on the different machines, that is, the behaviour of the program will be machine independent.

PLAIN FORTRAN will especially be useful to the research scientist who is just starting to use computers. It will allow him to develop his skills gradually. He normally makes use of generally available computing services over which he has no direct control and therefore must make his programs as machine (or compiler) independent as possible if he is to progress in his research.

PLAIN FORTRAN is described in this publication without reference to existing standards for the FORTRAN programming language. Normally the scientist has access only to the manufacturer's FORTRAN manual. He can use the restrictions in this publication to simplify his work and make his programs compatible without having to examine or assess claims about the compatibility of the manufacturer's FORTRAN. The four phases suggested at the end of the Introduction can be used to specify how "plain" the language should be.

ACKNOWLEDGEMENTS

The author wishes to thank the personnel of the Inland Waters Branch and the Computer Science Division who have through their comments, contributed significantly to this publication.

Readers are invited to forward their comments and criticisms regarding contents and omissions.

PLAIN FORTRAN

A guide to compatibility in computer programming

J.J. THERRIEN

INTRODUCTION

Anyone learning FORTRAN is faced with the problem of separating the basic information from the more sophisticated considerations found in the manufacturer's manuals. Although these manuals contain the standard FORTRAN features, they also contain special features that may or may not be compatible with other manufacturer's special features. The reader usually has no way of determining which features are not standard or essential and which ones are truly compatible. He might only discover that they are not compatible when he is forced to run the programs on other machines. In some cases he might be obtaining wrong results and not discover the fault (incompatibility) for some time, if ever. These features may introduce new concepts, new complications (which are not always explained), and therefore new potential sources of trouble. As a result the newcomer must wade through a lot of material, the usefulness or even the purpose of which, he does not quite understand.

PLAIN FORTRAN is not presented as a series of specifications. This would be ideal and is contemplated for the future. For now, it is presented in the form of restrictions. It is assumed that the reader has some very basic knowledge of FORTRAN and that he has access to a manual. If he adheres at all times to the restrictions mentioned he will be able to run his programs on almost any medium or large scale computer without changes and he will be assured that his programs will yield consistent results.

Any statements or features that are not specifically mentioned should be disregarded completely. Only the essential features that are compatible have been listed; PLAIN FORTRAN does not pretend to include all the features that are compatible. For example the "PAUSE" statement is not mentioned. This statement is practically useless for programs that are run in batches in a large data centre. The "assigned GØ TØ" statement is omitted because it accomplishes nothing that cannot be done with a "computed GØ TØ". It introduces a new type of data and its use is governed by considerations of program execution speeds which are connected with the way these statements are implemented on the machine in question. These considerations are not of prime importance while learning FORTRAN. The EQUIVALENCE statement is also omitted because it is only partly compatible and it introduces many complications.

The reader will find many contradictions between the restrictions listed and the FORTRAN manual he is using. For example, in section 1.1 it is said that names cannot contain more than 6 characters. If the reader happens to have a CDC 3100 manual he will see that up to 8 characters are allowed. This of course is true for that particular version of FORTRAN. For the CDC 6400 the maximum number of characters is 7, and it is 6 for the UNIVAC 1108 and the IBM 360. If he restricts himself to 6 characters he will not have to change his programs to run them on different machines. Such expressions as "cannot be used", "can only be", "are not allowed", etc... should be interpreted in this context.

In the above example one might be prepared to change the size of the names if and when another computer is used. During the compilation, error messages would be printed identifying all the names that have to be shortened.

The problem is that not all incompatibilities produce error messages. For example, in section 11.2 it is mentioned that when using DØ loops the value of the loop index is undefined after the looping has been completed. In some versions of FORTRAN it is defined and it can be determined. If the program is run on another machine and the value is different then incorrect results will be obtained. There will be no error messages printed because the compiler cannot warn the user that this has happened. As far as the user is concerned everything will seem normal. The results might look reasonable although they will be erroneous. It is therefore of the utmost importance to adhere to all restrictions at all times because one can never rely on the compiler for uprooting all the sources of errors.

Experimentation should never be used to get around restrictions. One is sometimes tempted to "try and see" what the compiler will do. For example the value of the DØ loop index after the loop has been completed could be determined by running a test program and printing its value at the end of the loop. One might discover for instance that it is one higher than the last value used in the loop. The problem is that in some versions of FORTRAN its value depends on what statements appear within the loop so that conclusions drawn from the experiment are meaningless even for that particular version of FORTRAN, let alone for other versions. The results obtained might vary in the same compiler, in different compilers for the same version of FORTRAN and for other versions of FORTRAN.

It might be worthwhile at this point to explain some of the terminology. When it is said that results are "defined" this means that they will be the same (within the stated precision) no matter what compiler or what version of FORTRAN is used. Results are "undefined" if there is no guarantee that this will be the case. For the sake of simplicity the word "machine" or "computer" has been used to designate the "version of FORTRAN" that is implemented on computers of the same type. For example, in sections 17.2 and 18.2 it is stated that binary records can only be processed "on the same computer". This actually means that they can be processed with the same version of FORTRAN on machines of the same type. It is difficult to define the exact meaning of "same" without considering specific cases in some detail. In this example, the word "same" should indicate to the newcomer that he can expect serious problems if changes are made to the computing facilities he is using. He should therefore consult an experienced person before committing himself to using these features.

The reader who is learning FORTRAN should not attempt to learn all the statements or features at once. He should start with simple programs and proceed to add new statements to his repertoire as required after he has mastered the more basic ones. The following four phases are suggested.

PHASE I

In this phase the beginner is only concerned with a main program that reads data cards and prints results. The READ with format in section 17.1 is only used for reading cards and the WRITE with format in section 18.1 is only used for printing. The A-format code in section 16 should be ignored.

The following sections and any references to them can be completely ignored:

7.	Character Data
17.2	Read Without Format (Binary Data)
18.2	Write Without Format (Binary Data)
19.	BACKSPACE Statement
20.	REWIND Statement
22.	CØMMØN Statement
23.	SUBRØUTINE Statement
24.	FUNCTIØN Statement
25.	CALL Statement and Function Reference
26.	RETURN Statement
30.	Double Precision on the IBM 360.

PHASE II

Subroutine and function subprograms are now added to the list and the WRITE with format in section 18.1 can be used for punching cards (as well as for printing). Note that the CØMMØN statement and any reference to it or to variables stored in the CØMMØN area should be disregarded completely at this stage.

The following sections are introduced:

23.	SUBRØUTINE Statement
24.	FUNCTIØN Statement
25.	CALL Statement and Function Reference
26.	RETURN Statement
30.	Double Precision on the IBM 360 (only when applicable)

PHASE III

In this phase binary tapes can be used for storing intermediate results and the CØMMØN statement is added to the list.

The following sections are introduced:

17.2	READ Without Format (Binary Data)
18.2	WRITE Without Format (Binary Data)
20.	REWIND Statement
22.	CØMMØN Statement

PHASE IV

Other allowable features are added as required.

NOTE: For many applications there is no need to proceed beyond Phase II. In the first two phases the newcomer should seek the advice of an experienced programmer whenever possible. He should not proceed beyond Phase II without discussing his application with an experienced person.

1. Names of Variables, Functions and Subprograms

1.1 names can be not more than 6 characters long; the length of a name is 1 to 6 characters. For example the name ACCØUNT contains 7 characters and is not valid.

1.2 the first character of any name must be a capital letter of the alphabet, i.e. from A to Z. For example $ACCNT is not valid.

1.3 the other characters if any, must be capital letters of the alphabet (i.e. A to Z) or digits from 0 to 9. For example "A$C" is not a valid name while "A908" is.

2. Types of Variables and Functions

These rules do not apply to subroutine names as they do not have "type" characteristics.

2.1 only two types of variables and functions are allowed, integer (i.e. fixed-point) and real (i.e. floating-point).

2.2 the type is solely determined by the first letter of the name (type declarations are not to be used):

2.2.1 integer type: the first letter of the name is always one of the letters I, J, K, L, M or N.

2.2.2 real type: the first letter of the name is never a letter from I to N, i.e. it is a letter from A to H or from Ø to Z.

2.3 the contents of a variable are undefined until a number is stored in the location in question by the program. All locations must be initialized by the program.

3. Types of Constants

The size of constants will be considered later with the discussion of precision (see sections 28, 29 and 30).

3.1 only two types of constants are allowed, integer and real. For example 1250, 1250.0 and 1.25E+3.

3.2 only E can be used for real constants containing an exponent. For example 1.0D-1 is not allowed except for double precision as in section 30.

4. Statement Numbers

Statement numbers cannot be more than 4 digits long, i.e. the permissible range of statement numbers is from 1 to 9999 inclusive.

5. Subscripts

5.1 a variable can have not more than 3 subscripts. For example ARRAY(2,1,4,I) is not allowed as it has 4 subscripts.

5.2 subscripted variables are dimensioned with a DIMENSIØN statement only when they are not in CØMMØN; if they are in CØMMØN they are only dimensioned in the CØMMØN statement. This restriction does not of course apply if the version of FORTRAN being used does not allow variables to be dimensioned in CØMMØN (see section 22).

5.3 only the following seven forms of subscripts are allowed:

(c), (i), (i+d), (i-d), (c*i), (c*i+d) and (c*i-d)

where "c" and "d" are unsigned integer constants

"i" is a nonsubscripted integer variable

The following are allowed:

ARRAY(I,KIND,2),IND(2*KIND-1), B(10,J+11), INPUT(6), etc...

The following are not allowed:

ARRAY(I,KIND,-2),IND(KIND*2-1), B(10,11+J),INPUT(2*3),ØUT(IND(2)), TAB(-I), etc...

5.4 subscripts must be larger than zero and must not be larger than specified when the array in question was dimensioned.

6. Expressions

6.1 only arithmetic expressions are allowed. The only allowable operators are addition (+), subtraction (-), multiplication (*), division (/) and exponentiation (**). Relational (e.g. EQ, GT,...) and logical (e.g. AND, ØR,...) operators are not allowed. Following are allowable (arithmetic) expressions:

```
A
-2
2.0*A+(ARRAY(I,N,K)/X)
IND+(N+1)*(N+2)/2
```

6.2 mixed expressions are not allowed: an expression must be either all integer or all real. For example the expression (A*I+N) is not allowed. Although they contain two types of data the following are not mixed expressions and are allowed:

ARRAY(I,J,2)*B**2 where ARRAY is a 3-dimensional array
I+2*INIT(A,N) where INIT is an integer function

The first expression is all real because "ARRAY(I,J,2)" yields a real number and so does "B**2". The second expression is all integer because the function reference "INIT(A,N)" yields an integer number.

6.3 brackets (parentheses) should be used extensively (within reason) not only to control the order of the computations but also to make the intended meaning clear to the reader. The order of the computations is from left to right with operations of the same level and the levels are as follows: exponentiation (**) at the highest level, multiplication (*) and division (/) on an equal footing at the second-highest level, and finally addition (+) and subtraction (-) on an equal footing at the lowest level. When brackets are used the operations in the innermost brackets are computed first.

For example, the following expression is not clear to the reader:

```
B*XY**X/Y**Z*YZ**Z/X+Z-2.0*CY
```

It should be written as follows:

```
(B*(XY**X)/(Y**Z))*((YZ**Z)/X)+(Z-
2.0*CY)
```

It might also be written as follows:

```
(Z-2.0*CY)+(B/X)*(XY**X)*(YZ**Z)/
(Y**Z)
```

This last expression is even simpler and clearer.

6.4 if an expression contains a function reference to a subprogram that changes the value of one of its arguments, that argument cannot appear anywhere else in the expression.

For example, if IFUNC is a function that changes the value of its first argument then the following is not allowed:

```
I*J+IFUNC(I,1,N)
```

The value of I in "I*J" will depend on whether "IFUNC(I,1,N)" has been evaluated before or after "I*J" and the order may depend on the particular version of FORTRAN being used. For example:

```
(I*J)+IFUNC(I,1,N)
```

might give results that are different from:

```
I*J+(IFUNC(I,1,N))
```

6.5 if an expression contains a function reference to a subprogram that changes the value of a variable in CØMMØN then that variable cannot appear in the expression.

For example, if CØMP is a function that changes the value of B which is CØMMØN then the following is not allowed:

```
A*B+CØMP(X,Y)
```

The same reasoning as in 6.4 applies. There is no guarantee that "CØMP(X,Y)" will be evaluated before "A*B" and vice versa. The results are not defined.

7. Character Data

Character data are data obtained by reading characters with an A-format code. The use of character data should be avoided as much as possible. It is not essential for most scientific computing. If it is not used, then this section and all references to A-format codes in section 16 can be ignored. The manipulation of character data will only be machine independent if the following rules are adhered to at all times.

7.1 character data can only be generated by a READ statement with an A-format code. There are no character constants, e.g. literal constants such as 'ABC',3HABC and 3RABC are not allowed anywhere. Note that H is an allowable format code and can only be used in a FORMAT statement (section 16). See section 16.6 for a list of the allowable characters for data.

7.2 character data should only be stored in integer variables (subscripted or not). Real variables should never be used for this purpose.

7.3 only the A-format codes A1, A2, A3 and A4 are allowed. A maximum of 4 characters can be stored in each location. If less than 4 characters are stored they are left-justified by padding on the right with blanks. If "c", "d", "e", and "f" represent one character each and if "b" represents a single blank then reading:

"c"	with A1 stores the same as reading "cbbb" with A4
"cd"	with A2 stores the same as reading "cdbb" with A4
"cde"	with A3 stores the same as reading "cdeb" with A4
"cdef"	with A4 stores "cdef" in the location in question.

7.4 The following rules for manipulating character data also apply to data that are used by subprograms through the argument list or the CØMMØN area. Character data can only be used as follows:

7.4.1 in a simple assignment statement that does not contain operators (i.e. computations) and that does not involve type conversions (e.g. integer to real conversion). For example, the following are allowed:

```
ICH(1,I)=IN
```

where ICH is an array.

```
ICHAR=ICHF(I,2)
```

where ICHF is a function returning character data or an array containing character data.

The following are not allowed:

```
  ABC = ICHAR
  IND = ICHAR1-ICHAR2
ICHAR = 2*ICHAR1+10
```

7.4.2 in an IF statement that compares only character data and only tests for equality, i.e. it cannot test for one quantity being larger or smaller than the other. The brackets must and can only contain the subtraction of two quantities which are both character data. These quantities can only be integer variables (subscripted or not) that contain character data or function references that yield character data. Since the IF statement is only allowed to test for equality, the first and last statement numbers specified must always be the same. The following are allowed:

```
IF(ICHAR1-ICHAR2)110,120,110
IF(IFUNC(I+1,J)-ICHAR)1020,
90, 1020
```

The following are not allowed:

```
IF(ICHAR)110,120,110
IF(ICHAR1-ICHAR2)110,120,130
IF(ICHAR1-10)110,120,110
IF(ICHAR-ABC)110,120,110
IF(ICHAR1-ICHAR2)110,110,120
IF(ICHAR1+ICHAR2)110,120,110
IF(ICHAR1-ICHAR2-ICHAR3)110,
120,110
```

Because of the way character data are stored the comparison of character data can produce integer overflows in some cases, i.e. the result of the comparison (subtraction) is a number that exceeds the capacity of the machine. The IF statement will always produce the correct results (i.e. the numbers are unequal when the overflow occurs) but in some computer installations, this situation is treated as an error. Error messages are printed and sometimes the job is aborted. It is possible to prevent these overflows in a machine independent manner by testing for the sign of the values before comparing them. They are only compared if they have the same sign; then no overflow can occur. If they do not have the same sign, they are not equal and no comparison is performed. The rules for the IF statement can be broken only for testing the sign of the values. The fact that one value is larger or smaller than the other is meaningless in PLAIN FORTRAN; the same characters might produce opposite results on different machines.

In most installations these overflows are tolerated to allow the comparison of character data and they are not treated as errors. The comparison of character data should only be used if it is absolutely essential.

7.4.3 in the "list" of a READ or WRITE statement. When used with a FØRMAT statement, only the allowable (see section 16.1) A-format codes can be used, i.e. A1, A2, A3 or A4. The READ statement with format stores the data as shown in section 7.3. In the WRITE statement with format, the A code specifies the number (1 to 4) of characters that are to be written out starting at the leftmost character stored. If "c", "d", "e" and "f" each represent one of the four characters stored then writing "cdef" with:

A1 will produce "c"
A2 will produce "cd"
A3 will produce "cde"
A4 will produce "cdef"

Character data can be used with READ or WRITE statements without format; see those statements for general restrictions.

7.4.4 as an argument in a CALL statement or in a function reference provided that the subprogram in question obeys the above rules for manipulating character data.

8. Assignment (=) Statement

It is of the form:

v = e

where v is an integer or a real variable (subscripted or not) and it cannot be a function or an array name.

e is an arithmetic expression (see section 6), either integer or real.

Conversion occurs after the expression "e" has been evaluated if v and e are not of the same type (i.e. they are not both real or both integer). The assignment statement is the only statement where conversion is allowed. Multiple replacement statements, e.g. "A=B=C=2.0", are not allowed. Examples of allowable statements are:

```
   IND = IND+1
     X = (2.0*Y)-(Z* P)
ICØUNT = WNUM+2.0*(A+B)
  WNUM = ICØUNT+1
```

Section 10.3 contains comments about errors inherent in real (floating-point) numbers.

9. GØ TØ Statement

Only two types of GØ TØ statements are allowed (e.g. the "assigned" GØ TØ is not allowed):

9.1 unconditional:

GØ TØ n
where n is a statement number

Example:
GØ TØ 1100

9.2 computed:

GØ TØ $(n_1, n_2, \ldots, n_m)$,i
where n_1 to n_m are statement numbers
i is a nonsubscripted integer variable which must have a value greater than zero and not greater than the number (m) of statements listed, i.e. 1, 2, 3,..., m.

Example:
GØ TØ (10,110,130),IND
where IND = 1, 2 or 3

If IND is 1, control will be transferred to statement 10; if it is 2, to statement 110, and if it is 3, to statement 130. If it is less than 1, or larger than 3, the results are undefined.

The following restrictions are implied above:

9.2.1 the comma (,) that separates the right-hand bracket from i cannot be omitted. The following is not allowed:

GØ TØ (1100,10)I

9.2.2 i cannot be an expression. The following is not allowed:

GØ TØ (90,100),I-2

even if I can only have values of 3 or 4.

9.2.3 i cannot be a subscripted variable, a function name or an array name. The following is not allowed:

GØ TØ (90,100),INIT(2,I)

9.2.4 i must be integer; it cannot be real. The following is not allowed:

GØ TØ (90,100),X

9.2.5 i must have a value which is larger than zero and is not larger than the number (m) of statements specified. The following is not allowed:

```
I = 4
GØ TØ (10,20),I
```

10. IF Statement

Only the "three-branch" arithmetic IF statement is allowed. Logical or two-branch IF statements are not allowed.

IF(e)n_1,n_2,n_3
where e is an arithmetic expression (see section 6) which is either integer or real.
n_1,n_2,n_3 are statement numbers

Example:

```
IF(I+2-N) 110,10,1000
IF(A*2.0+B) 110,10,1000
```

The program will branch to statement 110 if the expression in the brackets has a negative value, to statement 10 if it is exactly zero and to statement 1000 if it has a positive value. The following restrictions are implied above:

10.1 the expression "e" must be an arithmetic expression (see section 6) and it cannot therefore be a logical or a relational expression. The following is not allowed:

```
IF(I .GT. 16) ...
```

10.2 all three statement numbers n_1,n_2 and n_3 and the two separating commas must be present at all times and cannot be replaced by an actual statement. The following are not allowed:

```
IF(...) 10,120
IF(...) GØ TØ 120
```

10.3 the comparison of real (floating-point) numbers for exact equality should be avoided because precision comes into play and the results are therefore machine dependent. A floating-point number is usually stored internally in the machine as a binary or hexadecimal fraction of a given precision (i.e. number of bits) and an exponent. In general it is impossible to exactly represent a decimal number in this fashion. There is always a small (plus or minus) error. For a given precision, the size and sign of this error depends on both the size of the decimal number itself and on the method used for converting that number to the internal representation. If computations were involved, the error also depends on the errors of the numbers involved and on the kind of operations performed. Since real numbers always have an error, one should always think of them as having a range and use the IF statement accordingly.

For an IF statement, to find real numbers equal, they must be identical internally in the machine. For example, the following statement is rather meaningless in practice and should be avoided:

```
IF(A-5.0) 10,20,10
```

Even if the value of A was obtained by reading "5.0" from a card directly into A, there is no guarantee that the above statement will transfer control to statement 20. The method used for converting the source program constant "5.0" in the IF statement to the internal representation might be slightly different from the method used at execution time for "formatting" the "5.0" from the card. Although this situation might be rare, there have been actual cases in the past where this has happened. In such a case the IF statement would find the numbers unequal. If A was computed from other values the chances are that the IF statement will never transfer control to statement 20, even if A was obtained from the statement "A=10.0-5.0" for instance.

The above considerations do not apply to integer (fixed-point) numbers because they are always represented exactly in the machine and no error is involved. When converting from real to integer or vice versa, the above considerations for floating-point numbers should be taken into account.

11. DØ Statement

The DØ statement can only have two forms:

DØ n m = i,j,k or
DØ n m = i,j (same as k=1 above)

where n is a statement number specifying the last statement in the range of the loop.
m is the index of the loop. It must be a nonsubscripted integer variable.
i,j and k are the loop parameters. They must be either unsigned integer constants, or unsigned (i.e. not preceded by a

minus sign) and nonsubscripted integer variables.

When i and j are integer variables they cannot have negative values. The following is not valid:

```
M = -4
N = -1
DØ 110 IND = M,N
```

The value of k must under all circumstances be larger than zero, i.e. it can be 1, 2, 3, etc... The following is not valid:

```
M = -4
DØ 110 I = J,K,M
```

11.1 By-Passing the DØ Loop

If the value of j is smaller than the value of i the results are undefined. In some versions of FORTRAN the loop will be executed once (i.e. the same as if j had been equal to i), while in others, the loop will be by-passed completely. In either case the final value of m is undefined (see section 11.2).

When it is possible for j to be smaller than i, an IF statement should always be used to make sure that the program will always behave in the same way no matter what version of FORTRAN is used.

Example 1: by-passing the loop

```
        IF(M-N) 100,100,150
    100 DØ 140 I=M,N
        ...
    140 CØNTINUE
        ...
    150  (set I if necessary)
```

Example 2: executing the loop once

```
        IF(M-N) 110,110,100
    100 N=M
    110 DØ 140 I=M,N
        ...
    140 CØNTINUE
```

11.2 The Value of the Loop Index (m)

The value of m is undefined after the looping has been completed. Its value depends on which version of FORTRAN is being used and in some versions its value changes according to circumstances.

The value of m is defined throughout the loop; i.e. it is available to any statement that is part of the loop. The value of m is defined outside the loop only if control was transferred to a point outside the loop by an IF or a GØ TØ statement that was part of the loop.

Example:

```
       DØ 20 I=1,10
       K1=I
       ...
       IF(I-N) 20,20,30
    20 CØNTINUE
    30 K2=I
```

The value of K1 will always be correct because the statement "K1=I" is within the loop. If N is smaller than 10, K2 will always have the correct value because the IF statement will transfer control to statement 30 (which is outside the loop) before the loop is completed; the value of I is defined in statement 30 because the loop was not terminated through statement 20.

But if N is larger than or equal to 10, the IF statement will always transfer control to statement 20 which is within the loop. Eventually, i.e. after the 10th time through the loop, control will drop from statement 20 to statement 30. In that case the value of K2 is undefined; the value of I is undefined in statement 30 because the loop was completed through statement 20.

The value of I should have been set at the end of the loop to ensure correct results under all circumstances and on any machine. The above example should have been written as follows:

```
       DØ 20 I=1,10
       K1=I
       ...
       IF(I-N) 20,20,30
    20 CØNTINUE
       I=10
    30 K2=I
```

The statement "I=10" defines the value of I to be used in statement 30. The value of K2 will always be correct no matter what version of FORTRAN is used and no matter what the value of N is.

11.3 Transfer of Control

The range of the DØ loop starts with the statement immediately following the DØ statement and includes all statements up to and including the statement ending the loop, i.e. statement n in "DØ n m=...". The DØ statement itself is not part of the range. The usual rules apply for the nesting of DØ loops.

11.3.1 control can be transferred from any statement within the range to any other statement within or outside the range. See the example in 11.2.

11.3.2 control cannot be transferred from a statement outside the range to a statement within the range of the loop. The following is not allowed:

```
    GØ TØ 110
100 DØ 120 I=1,10
    ...
110 A=B+C
    ...
120 CØNTINUE
```

Example 1 in section 11.1 is valid because the DØ statement itself is not part of the loop.

11.4 Changing the Value of the Index and Parameters

The value of the loop index (m) and of the control parameters (i,j,k) cannot be changed by a statement within the range of the loop. The following are not allowed:

```
    DØ 100 I=M,N              DØ 1030 K=1,N
    ...                       ...
    N=N+1                     DØ 1020 K=1,10
    I=I-1                     ...
    ...                  1020 CØNTINUE
100 CØNTINUE                  ...
    ...                  1030 CØNTINUE
    DØ 140 I=M,N,K            DØ 530 I=M,N
    ...                       DØ 520 N=1,10
    K=2                       ...
    ...                   520 CØNTINUE
140 CØNTINUE              530 CØNTINUE
```

12. CØNTINUE Statement

There are no compatibility problems with the CØNTINUE statement. It is usually used at the end of a DØ loop to prevent an IF or a GØ TØ statement from being the last statement in the loop.

13. STOP Statement

The simple form "STØP" can be used on most machines. It does not actually stop the machine but simply terminates the job in question by returning control to the operating system. The form "STØP n" where n is an integer constant should never be used.

14. END Statement

The END statement must never be used as an executable statement, and therefore can never have a statement number. Return from a subprogram should always be performed by a RETURN statement (not by the END statement) and a computer run should always be terminated by a STØP statement (not by the END statement of the main program).

15. Input-Output List

The input-output (I/O) list is used with READ and WRITE statements. It consists of a number of items separated by commas.

15.1 the I/O list can only contain variables (subscripted or not) and array names. It cannot contain constants (except integer constants used for subscripts and for indexing as in section 15.2 below). Neither can it contain expressions (except for subscripts) nor function references.

15.2 the items in the list can be indexed as follows:

(...,...,....,m=i,j)

where m is the index
i,j are the index loop parameters
m,i and j are the same as for a DØ statement

The value of m is undefined after the indexing has been completed. The indexing can be nested up to 3 levels.

Example:

```
((X(I),(A(I,J,K),B(J),J=M,N),I=1,
10),C(K,1),K=2,4)
```

15.3 in an input list, a variable that appears as an input variable cannot be used for a subscript or for indexing in the same list. The following are not allowed:

```
READ(10,KARD) I,IND(I+2)
READ(MAG) I,(IND(J),J=1,I)
READ(MAG)(I,IND(I),I=M,N)
```

The results of these operations are undefined. There is no guarantee in the first two examples that the I used as a subscript and for indexing will have the value just read in the same statement. In some versions it will and in other versions it might not. Some manuals do not explain what actually will happen in such situations.

In the third example the results are again undefined. This is similar to changing the value of the index of a DØ loop.

16. FØRMAT Statement

16.1 Format Codes

The only allowable format codes are I,F,E,X,H and A:

aIw
aFw.d
aEw.d
wX
wH
aA1,aA2,aA3 and aA4

where a, w and d are unsigned integer constants

a is optional and denotes the number of times the same code is to be repeated. If it is used it must be larger than 1, i.e. 2,3,...

w denotes the total width of the field and it is always larger than zero, i.e. 1,2,3... Note that the format code "aAw" has been written as aA1, aA2, aA3 and aA4 above because w can only have values of 1, 2, 3 or 4 with that code (see section 7).

d denotes the number of places to the right of the decimal. It can be zero or larger than zero but must always be smaller than w.

The use of literal data is not allowed; always use the H code instead. For example, 'ABCD' is not allowed and 4HABCD should be used instead. When the format codes are used for output with the WRITE statement, e.g. for printing, the following apply:

16.1.1 I-format code: there is no need to allow a space for the sign if the number is positive, e.g. "123" can be printed with the format code I3. The shortest field that can be used for output is therefore I1 if the number is positive. If the number is negative always allow an extra space for the sign, e.g. print "-123" with I4.

16.1.2 F-format code: always allow room for a sign (even for positive numbers), for the decimal point (even if d is zero) and for at least one digit to the left of the decimal point (even if the magnitude of the number is less than one). In other words in Fw.d, the smallest value of d is zero and its largest value is (w-3).

16.1.3 E-format code: for the fraction allow room for the sign, for one digit to the left of the decimal point and for the decimal point, and for the exponent allow room for the "E", for the sign of the exponent and two spaces for a 2-digit exponent. In other words in Ew.d, the smallest value of d is 1 (no data are printed if d is zero) and its largest value is (w-7).

16.2 FØRTRAN Records

The use of the slash "/" is allowed to delineate the start of a new FØRTRAN record only when printing.

The end of the FØRMAT statement can also be used to start a new record. If the list requires more format items than are given in the format, a new record will be started at the beginning of the FØRMAT statement. This feature is never used when the FØRMAT statement contains brackets, as in section 16.3, i.e. repetition factors.

The FØRTRAN record should never be longer than 132 characters when used with BCD information on tape or when printing (132 includes the carriage control character). For cards, the record length is never longer than 80 characters.

16.3 Brackets within the FØRMAT

A group of format items can be repeated a number of times if they are enclosed in brackets:

```
FØRMAT(...,a(item,...,item),...)
```

where "a" is an unsigned integer constant specifying the number of times the items in the bracket are to be repeated.

For example:
3(3A4,I3) is the same as
3A4,I3,3A4,I3,3A4,I3

16.3.1 When brackets are used, "a" must always be specified and must be larger than one, i.e. 2,3,... The following is not allowed:

...,(3A4,I3),...

16.3.2 The brackets cannot be nested,

i.e. they cannot contain other brackets. The following is not allowed:

...,3(3A4,2(I2,I1),I3),...

16.4 Carriage Control Characters

When the FØRMAT statement is used for printing, the first character is used for carriage control and is not printed. Although not all versions of FØRTRAN implement carriage control the same way, the end result should be as shown here. The only allowable carriage control characters are:

blank single spacing (advance one line before printing)
zero(0) double spacing (advance two lines before printing)
one(1) advance to the first line of the next page before printing.

16.5 Correspondence of List and Format Items

Except for the X and H format codes which do not have corresponding list items, there must be a one-to-one correspondence between the items in the list and the items in the format, i.e. no type conversion is allowed.

16.5.1 the list item must be an integer variable for the I and A (see section 7.2) format codes.

16.5.2 the list item must be a real (floating-point) variable for the F-and E-format codes.

16.6 Allowable Characters with H-and A-Format Codes

The following 43 characters are compatible and can be used with the A-and H-format codes:

A to Z	capital letters, i.e. upper case only
0 to 9	decimal digits
blank	
-	minus sign
.	decimal point
,	comma
/	slash
*	asterisk
$	dollar sign

The above characters can be used as data as well as for printing. The first 2 columns of data cards must never contain "//" or "/*"; these identify control cards on the IBM 360. In general, the last five characters listed above, i.e. special characters, should be avoided at the beginning of a data card.

The following 4 characters can be used only for printing and should be avoided as much as possible:

+	plus sign
=	equal sign
(	left parenthesis
)	right parenthesis

For the IBM 360, these four characters must be punched in EBCDIC in the H field of the source-program card because the "BCD" option of the compiler does not change these fields to EBCDIC. They will have to be changed back to BCD when the program is used on other machines. These four characters should never be used as data because the IBM 360 uses different codes for these characters.

16.7 Allowable Character with I-, F-and E-Format Codes

Because of restrictions mentioned in 16.6 regarding the plus sign, "+", no input data should contain that sign. This is especially true of the exponent with E-type numbers. For example instead of 1.0E+10 use 1.0E10 for data. This restriction only applies to data. It does not apply to the coding of constants in the source program.

Blanks are only allowed as leading characters. Blanks cannot be used within or on the right-hand side of a figure (a blank is allowed on the left-side of an exponent). For example "1 2" cannot be read as I3. If an I- or F-type field contains all blanks it is interpreted as a value of zero.

17. READ Statement

Provisions should always be made for the program to recognize the end of the file by examining the data just read, i.e. the last record in the file should contain data that identifies it as the last data record (End-of-file should never be used).

17.1 Read with Format (BCD data)

READ (u,f) list

where u is the unit number of the device (i.e. card reader, tape unit, disc or drum). It must be an unsigned integer variable. It is bad practice to use a constant because unit numbers

are not compatible and must be changed for different machines or different data centres.
f is a FØRMAT statement number. It cannot be a variable.
"list" is the list described in section 15.

The only really compatible medium are cards, provided restrictions as to allowable characters given in section 16 are adhered to at all times. Tapes are not usually compatible; there are difficulties with physical record lengths on different machines. Tapes (or records on discs or drums) can be processed by the same machine if they were written with a WRITE statement with format and if the number of characters (i.e. positions read) is not larger than the number of characters written by the corresponding WRITE statement. For cards, the maximum number of characters is 80 per record, and for other devices, the number should not exceed 132 characters per record. See section 16.2 for the processing of several records with the same "list".

In this statement, the "list" must contain at least one item, it cannot be omitted.

17.2 Read without Format (Binary Data)

READ (u) list

where u and "list" are the same as in 17.1.

This statement cannot be used to read cards. It can only be used to read binary tape (disc or drum) records that were produced on the same machine with a WRITE statement without format. The number of items read by this statement must never be larger than the number written in the corresponding WRITE statement and there must be a one-to-one correspondence: integer (fixed-point) items must correspond to integer items and real (floating-point) items must correspond to real items. This is similar to the correspondence of items listed in the CØMMØN area in different subprograms except that the list of the READ statement can be shorter than the list of the corresponding WRITE statement.

As far as the FØRTRAN program is concerned, each execution of this statement processes one and only one, binary record. The physical arrangement of the data on the tape is of no significance at this stage as it depends on the version of FØRTRAN used. It is sufficient to say that in general, it is more economical to process a few, long binary records than it is to process a large number of small (short list) records.

The execution of this statement always starts at the beginning of a FØRTRAN binary record. If the list of the previous READ statement was too short then the remainder of the previous record is lost for the moment. If necessary it can later be processed by going back to that record with a BACKSPACE or REWIND statement and reading it again with a sufficiently long list. The list may be omitted with this statement. In that case the record in question is by-passed completely, i.e. the file is positioned at the next record.

This statement is much more efficient than the READ with format because it does not involve data conversions, i.e. conversion from the character representation of the input data to the internal machine representation. This statement should only be used to read back intermediate results that were stored on the same computer.

18. WRITE Statement

18.1 Write with Format (BCD data)

WRITE(u,f) list

where u,f and "list" are the same as in section 17.1

The "list" may be omitted in this statement.

This statement can be used to punch cards and to write BCD records on tape (disc or drum). See section 17.1 above for restrictions for these devices. This statement can also be used to print data. The maximum allowable record length is 132 characters including the carriage control character.

18.2 Write without Format (Binary Data)

WRITE (u) list

where u and "list" are the same as in section 17.1.

The "list" must contain at least one item; it cannot be omitted in this statement.

Each execution of this statement produces one binary FØRTRAN record. See section 17.2 for restrictions. This statement should only be used for storing intermediate results to be read back on the same computer.

19. BACKSPACE Statement

BACKSPACE u

where u is the same as in section 17.1.

This statement can only be used with records on tape (disc or drum). Each execution makes the program go back one FØRTRAN (BCD or binary) record; after the first record has been reached this statement has no effect.

Excessive execution time sometimes results when using this statement on some machines. This statement should only be used when absolutely necessary. The REWIND statement should be used where applicable.

20. REWIND Statement

REWIND u

where u is the same as in section 17.1.

This statement is used to reposition a file at the first FØRTRAN record (BCD or binary) on tape (disc or drum).

21. DIMENSIØN Statement

This statement is used to specify the dimensions (i.e. the maximum value of each subscript) of subscripted variables that are not in CØMMØN (also see section 22). It is always placed at the beginning of a main program or of subprograms and precedes the first executable statement.

No variables can have more than 3 subscripts (see section 5). The list takes the form of items separated by commas and each item is an array name with the maximum value (larger than one) of subscripts in brackets. The items can only have three forms (adjustable dimensions are not allowed):

a(i)	for one-dimensional arrays
a(i,j)	for two-dimensional arrays
a(i,j,k)	for three-dimensional arrays

where a is the array name, real or integer. i,j and k are unsigned integer constants (they cannot be variables) larger than one, i.e. 2,3,...

Example:
DIMENSIØN A(2,4),IND(5),KIND(500,2),...

22. CØMMØN Statement

This statement is used to make data available to both the main program and subprograms or just between subprograms by making variables share the same storage locations. Arrays that are in CØMMØN must be dimensioned in this statement (this is done in the same way as in the DIMENSIØN statement) unless the version of FØRTRAN being used does not allow it; in that case they must be dimensioned in a DIMENSIØN statement. Most versions of FØRTRAN now allow dimensions in the CØMMØN statement.

The items are listed in the following order: first, all the real variables, then all the integer variables (this facilitates the use of double-precision on some machines).

There must be a one-to-one correspondence between the items listed in the CØMMØN area wherever it appears in the main program and/or in the subprograms. The number of items must be the same, the corresponding items must be of the same type (i.e. real or integer) and if they are arrays they must have exactly the same dimensions. It is the order that is important; the names used need not be the same although it is good practice to have them the same whenever possible.

If the main program or one of the subroutines does not use all of the variables in CØMMØN then dummy variables (i.e. names that are not used for other purposes) are inserted in CØMMØN to make the lists match as specified above. For example in one subprogram we might have:

CØMMØN A(10,2),BØX,X,Y(50),IND,NUM(4,6)

while in another we might have:

CØMMØN A(10,2),BØX,XYZ,DUM(50),IND,
IDUM(4,6)

where DUM and IDUM might be dummy variables. There is a one-to-one correspondence between the items:

A(10,2)	A(10,2)	(real)
BØX	BØX	(real)
X	XYZ	(real)
Y(50)	DUM(50)	(real)
IND	IND	(integer)
NUM(4,6)	IDUM(4,6)	(integer)

The CØMMØN statement should never be used in a program or subprogram that does not use some data in the CØMMØN area, i.e. the CØMMØN statement never contains only dummy variables.

The use of "labelled" CØMMØN is not allowed. There can be only one (unlabelled) CØMMØN area. The CØMMØN statement never contains slashes.

The following are not allowed:

```
COMMØN /DATA/ A,B,C,D
COMMØN / / A,B,C,D
```

23. SUBRØUTINE Statement

The dummy arguments listed in this statement will be replaced by the actual arguments listed in a CALL statement. The arguments in the list are separated by commas (slashes are not allowed).

The dummy arguments can only be nonsubscripted real or integer variables, or array names. They cannot be function or subroutine names and they cannot be in CØMMØN. When a dummy argument is an array name it must also appear in a DIMENSIØN statement in the subprogram. The dimensions specified (adjustable dimensions are not allowed) must be exactly the same as those of the actual argument which will appear in the CALL statement (or the function reference in the case of a function subprogram). There is one exception to this rule. When the dummy argument in question is the name of an array that is one-dimensional (i.e. one subscript) it can have any dimension (larger than one) in the subprogram. It should be noted, of course, that the value of the subscript in the subprogram must never exceed the dimension of the corresponding actual argument in the calling program. It should also be noted that if this array name appears unsubscripted in an input or output list the number of items processed will equal the dimension specified in the subprogram, not that of the actual argument in the calling program.

It is possible for a subroutine subprogram to have no arguments; in this case the brackets are omitted and the CØMMØN area is used to share data with the calling program. As noted in section 24, a function subprogram always has at least one argument.

24. FUNCTIØN Statement

The dummy arguments listed in this statement will be replaced by the actual arguments listed in a function reference. The restrictions for the dummy arguments in this statement are exactly the same as for the SUBRØUTINE statement in section 23.

The rules for naming functions are explained in sections 1 and 2. The type (integer or real) of the function is determined solely by the first letter of its name. A function subprogram must always have at least one argument.

25. CALL Statement and Function Reference

The actual arguments specified in a CALL statement or in a function reference must have a one-to-one correspondence with the dummy arguments listed in the SUBRØUTINE or FUNCTIØN statement respectively. The number and order of the arguments must be the same, they must be of the same type (i.e. corresponding arguments are either both real or both integer), and array names always correspond to array names.

The following function subprogram will be used as an example in explaining the way in which arguments are used by subprograms:

```
   FUNCTIØN CØMP (XD,YD)
   CØMMØN AD,BD,CD
10 XD=XD+1.0
20 BD=XD+YD
30 CØMP=YD+CD
   RETURN
   END
```

It is possible for this subprogram to change the value of its first argument (XD) in statement 10, and to change the value of the second variable (BD) in CØMMØN in statement 20. Although the subprogram shown above happens to be a function subprogram the same considerations also apply to subroutine subprograms.

In general it is possible for a subprogram to change the value of a dummy argument or of a variable in CØMMØN if the variable:

- appears on the left side of an assignment statement (e.g. statements 10 and 20)
- appears as the loop index in a DØ statement or as the loop index of an indexed (section 15.2) input/output list. See section 25.6 for restrictions applying to dummy arguments
- appears as in input variable in the list of a READ statement
- is changed by another subprogram that is referenced by the subprogram in question.

In general a variable (i.e. location) in CØMMØN is "not used" by a subprogram if:

- it does not appear anywhere else in the subprogram, i.e. it only appears in the CØMMØN statement. For example AD above is a dummy variable.
- it is "not used" by another subprogram that is referenced by the subprogram in question.

The following calling program will be used to describe the use of arguments:

```
    CØMMØN A,B,C
    A=1.0
    B=1.0
    C=1.0
    X=1.0
    Y=1.0
100 A=CØMP(X,Y)
```

Four examples will be worked out with the above program by changing the arguments of the function reference in statement 100 above: CØMP(X,Y), CØMP (C,Y), CØMP(X,X) and CØMP(X,B). To understand the mechanism by which the actual arguments are used in the subprogram, the function reference in statement 100 will be replaced by a series of equivalent statements. For each example three sets of statements will be given. The columns entitled "Version I" and "Version II" show how two different versions of FØRTRAN might handle the arguments while the column entitled "User" shows how the user usually "thinks" they are handled. The results are compared in each case. The important thing to remember is that the results must be independent of the version used <u>and</u> the results must be the same as what the user expects them to be. The examples are followed by restrictions which must be observed at all times to obtain consistently correct results with different versions of FØRTRAN.

<u>EXAMPLE 1: A=CØMP(X,Y)</u>

Version I	Version II	User
1 XD=X	--	--
2 YD=Y	2 YD=Y	--
10 XD=XD+1.0	10 X=X+1.0	10 X=X+1.0
20 B=XD+YD	20 B=X+YD	20 B=X+Y
30 CØMP=YD+C	30 CØMP=YD+C	30 CØMP=Y+C
31 X=XD	--	--
100 A=CØMP	100 A=CØMP	100 A=CØMP

Results	Results	Results
X=2.0	X=2.0	X=2.0
A=2.0	A=2.0	A=2.0
B=3.0	B=3.0	B=3.0

The <u>same</u> results are obtained in all three cases because this is a <u>valid</u> example. Note in <u>Version I</u> that the subprogram does not work directly with the actual arguments X and Y. It first stores (statements 1 and 2) the values of the arguments in temporary work areas (XD,YD) and then uses the work areas to perform the computations. Before returning to the calling program, it stores (statement 31) the new value of XD, in the location of the actual argument X. The value of YD was not changed; its value is not stored in Y for this reason.

In <u>Version II</u> the actual argument X is worked on ~~directly~~ because XD is likely to change value as it appears on the left-hand side of the assignment statement 10. The <u>User</u> usually assumes that the subprogram is working directly with the actual arguments X and Y at all times. This is what he should have (and usually has) in mind when he designs the subprogram, provided he obeys the restrictions listed below.

Note that the variables in CØMMØN (i.e. B and C in the calling program corresponding to BD and CD in the subprogram) are always worked on directly. This applies to any version of FØRTRAN.

<u>EXAMPLE 2: A=CØMP(C,Y)</u>

Version I	Version II	User
1 XD=C	--	--
2 YD=Y	2 YD=Y	--
10 XD=XD+1.0	10 C=C+1.0	10 C=C+1.0
20 B=XD+YD	20 B=C+YD	20 B=C+Y
30 CØMP=YD+C	30 CØMP=YD+C	30 CØMP=Y+C
31 C=XD	--	--
100 A=CØMP	100 A=CØMP	100 A=CØMP

Results	Results	Results
C=2.0	C=2.0	C=2.0
A=2.0	A=3.0	A=3.0
B=3.0	B=3.0	B=3.0

The results differ (A is 2.0 in Version I) because this is <u>not</u> a valid example. See section 25.4 for general restrictions.

<u>EXAMPLE 3: A=CØMP(X,X)</u>

Version I	Version II	User
1 XD=X	--	--
2 YD=X	2 YD=X	--
10 XD=XD+1.0	10 X=X+1.0	10 X=X+1.0
20 B=XD+YD	20 B=X+YD	20 B=X+X
30 CØMP=YD+C	30 CØMP=YD+C	30 CØMP=X+C
31 X=XD	--	--
100 A=CØMP	100 A=CØMP	100 A=CØMP

Results	Results	Results
X=2.0	X=2.0	X=2.0
A=2.0	A=2.0	A=3.0
B=3.0	B=3.0	B=4.0

Note that the results that the user expects are <u>not</u> those (A and B are different) he would get from the two versions of FØRTRAN shown. See section 25.4 for general restrictions.

<u>EXAMPLE 4: A=CØMP(X,B)</u>

Version I	Version II	User
1 XD=X	--	--
2 YD=B	2 YD=B	--
10 XD=XD+1.0	10 X=X+1.0	10 X=X+1.0
20 B=XD+YD	20 B=X+YD	20 B=X+B
30 CØMP=YD+C	30 CØMP=YD+C	30 CØMP=B+C
31 X=XD	--	--
100 A=CØMP	100 A=CØMP	100 A=CØMP

Results	Results	Results
X=2.0	X=2.0	X=2.0
A=2.0	A=2.0	A=4.0
B=3.0	B=3.0	B=3.0

The results that the user expects are not those (A is different) that he would get from the two versions of FØRTRAN shown. See section 25.5 for general restrictions.

25.1 If the dummy argument is not an array name (i.e. it is a nonsubscripted variable) and it is not possible for the subprogram to change its value (e.g. YD in CØMP) then the corresponding actual argument can be an expression (including function references), a variable (subscripted or not) or a constant. It cannot be an array name (see section 25.3). The following are allowed:

```
A=CØMP(X,2.0+Y)
A=CØMP(Y,ARRAY(I,J,2))
```

where CØMP is as described above.

The following is not allowed:

```
A=CØMP(2.0,Y)
```

because CØMP changes the value of its first argument (XD).

25.2 If the dummy argument is not an array name and it is possible for the subprogram to change its value, the actual argument can only be a variable (subscripted or not). See the examples in 25.1.

25.3 If the dummy argument is an array name, the corresponding actual argument must always be an array name and vice versa. It cannot be a subscripted variable for instance. When an argument is an array name, it is always worked on directly by the subroutine, i.e. the values of the argument are not stored in temporary storage. The method used is similar to the "User" column in the above examples; the results are always what one would expect by simply replacing the dummy array name by the name of the actual argument. The restrictions listed in sections 25.4 to 25.6 do not therefore apply to arguments that are array names, whether or not the actual argument is in CØMMØN.

25.4 If it is possible for the subprogram to change the value of the dummy argument then the actual argument cannot appear twice in the same CALL statement or function reference, nor can it be a variable in CØMMØN that is used by the subprogram in question. In the above example (CØMP) the following is allowed:

```
A=CØMP(A,Y)
```

Although A is in CØMMØN it (i.e. AD) is not used by CØMP. It would give results similar to CØMP(X,Y) in Example 1. The following is not allowed:

```
A=CØMP(C,Y)
```

This is Example 2. The subprogram CØMP changes the value of its first argument (XD) and C is in CØMMØN and (i.e. CD) is used by CØMP in statement 30. Also not allowed is:

```
A=CØMP(X,X)
```

This is Example 3. The subprogram CØMP changes the value of its first argument (XD) and the corresponding actual argument X appears more than once in the function reference.

25.5 If the subprogram changes the value of a variable in CØMMØN then that variable cannot be used as an argument. The following is not allowed:

```
A=CØMP(X,B)
```

This is Example 4. The value of B (i.e. BD in the subprogram) is changed by statement 20 of the CØMP subprogram.

25.6 When it is possible for a subprogram to change the value of a dummy argument (e.g. I2 in NUM2 below) then the value of the corresponding actual argument upon returning to the calling program is only defined if the dummy argument in question appears in the subprogram on the left of an assignment statement (e.g. statement 30 in NUM2 below) or as an input variable in the list of a READ statement. This point is especially important when a subprogram references other subprograms (which in turn might reference others). The following calling program and two subprograms will be used as an example:

```
      ...
   10 I=1
   50 K=NUM1(I)
      ...
      END
      FUNCTIØN NUM1(I1)
   40 NUM1=NUM2(I1)
      RETURN
      END
      FUNCTIØN NUM2(I2)
   20 NUM2=I2
   30 I2=I2+1
      RETURN
      END
```

To understand how the arguments are used, the above statements will be replaced by equivalent statements (similarly as in examples 1,2, 3 and 4 above):

Version I		User
10 I=1	...(Main)...	10 I=1
11 I1=I	...(NUM1)...	--
12 I2=I1	...(NUM2)...	--
20 NUM2=I2	...(NUM2)...	20 NUM2=I
30 I2=I2+1	...(NUM2)...	30 I=I+1
31 I1=I2	...(NUM2)	--
40 NUM1=NUM2	...(NUM1)...	40 NUM1=NUM2
50 K=NUM1	...(Main)...	50 K=NUM1

Results	Results
K=1	K=1
I=1	I=2

Note that Version I leaves I unchanged while the User expects it to be incremented by 1. In NUM2 the value of the actual argument I1 is reset in statement 31 because the corresponding dummy argument I2 appears on the left-hand side of assignment statement 30. This is not the case in NUM1 for the actual argument I. It is not reset before returning to the calling program because NUM1 has no way of knowing that NUM2 has changed the value of I1. It assumes that I1 remained unchanged and that therefore I need not be changed.

If in NUM1 the dummy argument I1 had appeared on the left side of an assignment statement or as an input variable, then the value of the actual argument I would have been changed by the equivalent statement "I=I1" following statement 40.

Subprogram NUM1 should be rewritten as follows:

```
      FUNCTIØN NUM1(I1)
      K1=I1
   40 NUM1=NUM2(K1)
   42 I1=K1
      RETURN
      END
```

Note that the dummy argument I1 now appears on the left of assignment statement 42 so that the actual argument will be changed upon returning to the calling program.

The above rule is also important when a dummy argument is used as the loop index of a DØ statement. This is only allowed if this same dummy variable also appears in the subprogram on the left of an assignment statement or as an input variable; otherwise the value of the corresponding actual argument is undefined in the calling program.

A dummy argument must never be used as the loop index of an indexed (section 15.2) input/output list.

26. RETURN Statement

This statement can only be used in a subroutine or a function subprogram. It must never be used in a main program to terminate the job; the STØP statement should be used. It only has one form, "RETURN". Multiple returns, e.g. "RETURN 2", are not allowed.

A subprogram can contain several RETURN statements. A RETURN statement should always be executed to return control to the calling program; the END statement should never be used for that purpose.

27. Library Functions

Following is a list of the most commonly used library functions that are available on the CDC 3100, UNIVAC 1108, IBM 7040 and IBM 360:

EXP,ALØG,ALØG10,ATAN,SIN,CØS,TAN,SQRT, ABS and IABS.

Because some versions of FØRTRAN allow mixed expressions, it is necessary to specify IABS for integer results, and ABS for floating-point results. A more comprehensive list of compatible library functions will be available at a later date.

28. Range and Precision of numbers

The allowable range and/or the precision of numbers is machine dependent. Tables 1 and 2 show what figures can be accommodated by the CDC 3100, UNIVAC 1108, IBM 7040 and IBM 360.

For integer numbers, the largest number shown for the UNIVAC 1108 is for the case where conversion to floating-point might be involved; this is likely to be the case in most programs. The actual number of digits is 10.3 if this restriction does not apply. For the floating-point numbers, the precision of the fraction shown for the IBM 360 is the worst precision. There is a loss of precision with some numbers because normalization is performed in hexadecimal instead of binary. The best precision that can be obtained is 7.2 decimal digits for single precision and 16.8 decimal digits for double precision.

The use of double precision on the IBM 360 is mentioned in section 30. For other machines one should consult the relevant literature and discuss the implications with an

	Magnitude of Largest Integer	Number of Decimal Digits
CDC 3100	8,388,608	6.8
UNIVAC 1108	134,217,727	8.1
IBM 7040	34,359,738,367	10.3
IBM 360	2,147,483,647	9.2

TABLE 1 Integer Numbers

	Precision of the Fraction		Range of the Exponent	
	Single	Double	Single	Double
CDC 3100	10.6	--	-308 to +308	--
UNIVAC 1108	8.1	18	-38 to +38	-308 to +308
IBM 7040	8.1	--	-38 to +38	--
IBM 360	6.3	15.9	-78 to +76	-78 to +76

TABLE 2 Floating-point Numbers

experienced person. The availability of double precision does not of course depend on the machine itself but depends on the version of FØRTRAN that is available in the particular installation.

29. Size of Constants

The following suggested sizes are in relation with the tables shown in section 28.

29.1 Integer Constants

An integer constant can have from 1 to 7 digits (the sign not included) and its absolute value must not be larger than 8,388,608.

29.2 Real Constants

The exponent of a real constant has a maximum of 2 digits and the permissible range is from -38 to +38 including zero.

If single precision is used on the IBM 360, the fractional part of constants can only have from 1 to 7 digits (not including the sign and decimal point) for a precision of 6.3 digits in the worst case. See section 30 for the use of double precision on the IBM 360.

On other machines 1 to 9 digits can be used and the precision is 8.1 digits in the worst case.

30. Double Precision on the IBM 360

The rules in section 2 for the names of variables and functions, and the rules in section 22 for listing real variables ahead of integer variables in the CØMMØN area make it a simple matter to change all variables and function subprograms to double precision. Insert the following statement ahead of the main program and ahead of each subprogram:

```
IMPLICIT REAL*8 (A-H,Ø-Z)
```

This statement will specify double precision for all variables and function names starting with the letters from A to H and from Ø to Z, i.e. all real variables and real function names will have double precision.

Real constants are made double-precision constants in the following way:

- constants with an exponent: use D instead of E, for example 7.0E+2 becomes 7.0D+2 in double precision.
- constants without an exponent: always use 8 or 9 digits (not including the sign and decimal point), for example write "-1.0000000" instead of "-1.0" to make it a double-precision constant. If not more than 9 digits are used, these constants will be acceptable in single precision on other machines.

Library Functions must all be changed for double precision. Following is a list of the double-precision version of the functions listed in section 27:

DEXP, DLØG, DLØG10, DATAN, DSIN, DCØS, DTAN, DSQRT and DABS (IABS is integer and is not therefore affected).

The above names can be used directly where applicable, although the following scheme will make it easier to change the program back to single precision for other machines. Use the single-precision names in the body of the program and subprograms and equate the double-precision name to the single-precision name by means of statement functions. They are inserted just ahead of the first executable statement of the main program or subprogram as required.

For example:

```
ABS(X)=DABS(X)
...
B=ABS(DØG)
...
...
FIR=4.0*(A-ABS(D-2.0*W))
```

The first statement is not an executable statement; it defines a statement function which in fact will replace ABS by DABS wherever ABS appears in the program. The other statements will in fact be compiled as follows:

```
B=DABS(DØG) and
FIR=4.0*(A-DABS(D-2.0*W))
```

To change the library function names back to single precision, simply remove the statement functions, e.g. "ABS(X)=DABS(X)" in the above example.

The F-format code is not affected by the use of double precision but the E-format code is affected. The D-format code must be used instead of E. Note that on output the D will appear instead of E as part of the output. If the data are punched on cards, for example, they must be read back in with a D code. This might create difficulties if these data are to be read on other machines.

Some machines, e.g. the CDC 3100, do not have double precision, and on some machines the implementation of double precision is different. On the UNIVAC 1108 all double-precision constants must have an exponent, e.g. the only way to make "1.0" a double-precision constant on that machine is to write it with a D exponent, e.g. "1.0D+0".

Note that the features mentioned in this section should only be used on the IBM 360 and that the IMPLICIT statement can only be used exactly as shown above (i.e. all other type declarations should be ignored). None of the above features should be used for single precision.

31. Features to be Ignored

It has been mentioned earlier that all statements or features of FØRTRAN that have not been mentioned specifically above are to be disregarded and are not to be used. Following is a list of some of the statements or features that are not to be used. The list is intended to help the reader in identifying the most popular features that are not to be used. The list is not complete; features not appearing in this list are not necessarily allowed.

Following are some of the features that are not to be used:

1. Logical and relational expressions
2. Logical IF, two-branch IF
3. More than 3 subscripts and adjustable dimensions
4. EQUIVALENCE
5. CØMMØN / label / ...
6. Assigned GØ TØ, ASSIGN
7. PAUSE, PAUSE n, PAUSE 'message'
8. STØP n
9. END and ERR features for the READ
10. Format codes G,L,Z,T,R,P, and literal constants (e.g. 'ABC')
11. END FILE a
12. Direct access statements: DEFINE FILE, READ with apostrophe, FIND
13. Type declarations IMPLICIT (except as in section 30), REAL, INTEGER, LØGICAL, CØMPLEX, DØUBLE PRECISIØN, CHARACTER
14. Slashes with dummy arguments, i.e. by name
15. Statement functions (except as in section 30)
16. ENTRY statement in subprograms
17. RETURN n, i.e. multiple returns from subprograms
18. EXTERNAL
19. DATA
20. (READ b, list), PUNCH, PRINT, READ TAPE, WRITE TAPE, etc...
21. BLØCK DATA
22. ENCØDE, DECØDE, BUFFER, IN, BUFFER ØUT
23. Mixed mode arithmetic
24. Character Data beyond 4 characters
25. NAMELIST
26. Variable format in READ or WRITE
27. ABNØRMAL
28. PARAMETER

The above list would be much longer if all possible extra features on the different machines were listed. With each version of FØRTRAN there are a number of functions and subprograms which have been written in assembly language and which are supplied by the manufacturer. A lot of these subprograms are not compatible and should not be used. It is difficult to make a comprehensive list of these but following are some of these subprograms that should not be used: FLD,IØCHK,IØCHKF,UNITST,UNITSTF, EØFCK,EØFCKF,AND,ØR,XØR,BØØL,CØMPL,CBRT,LENGTHF, SLITE,SLITEF,SLITET,SLITETF,SSWTCH,SSWTCHF,DVCHK, DVCHKF,EXFLT,EXFLTF,ØVERFL,ØVERFLF,ERF,ERFC, GAMMA,ALGAMA, etc... A complete list of allowable, i.e. compatible, library functions and subroutines will be available shortly. For the time being one should only use the functions listed in section 27.

Environment Canada Library, Burlington
3 9055 1017 2782 3

Date Due

22 MAY 2009			

BRODART, INC. Cat. No. 23 233 Printed in U.S.A.