*(ML,FE1,FE2,Z1,Z2,Z3) Floating Point Multiply. Floating point numbers symbolized by "FE1" and "FE2" are multiplied by each other and the product is returned as the value.

If the result exceeds the maximum allowed the value is "Z1", and if less than the minimum the value is "Z2".

*(DV,FE1,FE2,Z1,Z2,Z3) Floating Point Divide. The floating point number represented by "FE1" is divided by that represented by "FE2" and the result returned as the value of this function.

If the result of this operation exceeds the allowable range the value is "Z1", if below the minimum then the value is "Z2".

*(GR,FE1,FE2,T,F) Floating Point Greater Than. If the floating point number represented by "FE1" is greater than that represented by "FE2", then the value of this function is T, otherwise it is F.

*(LT,FE1,FE2,T,F) Floating Point Less Than. If the floating point number represented by "FE1" is less than that represented by "FE2" the value of this expression is T, otherwise it is F.

*(FPMACH, FE1) Floating Point to Machine Representation. The value of this function is the internal machine representation as an octal number of the floating point number symbolized by "FE1".

*(MACHFP,0) Internal Representation to Floating Point. The value of this function is the number in floating point format which is represented internally in the machine by the octal number symbolized by "0".

CHARACTER AND BIT DATA TYPES FOR FORTRAN--A PROPOSAL TO ANSI SUBCOMMITTEE X3J3

Boston Fortran Language Standards Group: Kenneth Ahl, John Barrington, John Hillier, Evelyn Mack, and Walter Whipple (Hartford Data Center, Control Data Corporation, Glastonbury Professional Plaza, 124 Hebron Avenue, Glastonbury, Connecticut 06033, telephone 203/633-0296. November 1971.

I. OBJECTIVES

Scope. A fundamental consideration which must be decided upon when contemplating Binary and Hollerith data types in FORTRAN is the overall approach. Three alternatives present themselves immediately:

1. Handle Hollerith data "under the guise of a name of one of the other types" with a few modifications to legitimize widespread non-standard practices.

2. Extend all the applicable concepts of FORTRAN to new data types (such as CHARACTER and BIT)for Hollerith and Binary data while introducing as few new concepts as possible.

3. Implement a whole new language facility specifically tailored to the manipulation of strings.

Each of these approaches is feasible and each has desirable aspects along with drawbacks.

The first approach minimizes changes to existing compilers and programs. It is undoubtedly the most efficient of the three for machine computation. However, it is inherently machine and compiler dependent as well as cumbersome for the user. If any degree of installation compatibility is to be preserved it can be very wasteful of core storage.

The second approach will require extensive changes to existing compilers and programs. Addressing will require shifting and masking for word oriented machines. Since the current standard FORTRAN makes no provision for the manipulation of aggregates of array elements as vectors or strings (except in I/O statements), the language syntax will have to be extended. However, this approach is highly machine and compiler independent and would provide efficient utilization of core storage. It could be implemented by a logical extension of the same rules that have worked so well for FORTRAN arithmetic.

The third approach will require a completely new compiler section to handle the new syntactical forms. Due to the powerful string operators it would be reasonably efficient. It would be machine and compiler independent but would require a completely different body of knowledge on the part of the programmer.

In the discussion which follows, where the different approaches dictate distinct options, the alternatives are presented in the same order used above.

Summary. Sections II through IV of this proposal deal with the various possibilities for character and binary facilities in the FORTRAN Language. It is intended that this portion constitute a working paper rather than an exposition of the desired modifications. In this way, the reasons for each change may be outlined and considered independently of other features. The fifth section, on the other hand, is intended to convey the opinion of the Boston FORTRAN Language Standards Group on the most desirable way to incorporate the manipulation of character and binary data in FORTRAN. The bibliography at the end is not intended to be exhaustive but to indicate those sources the group found useful.

Disclaimer. The opinions and concepts expressed in this proposal are solely those of the authors and the Boston FORTRAN Language Standards Group. No endorsement or approval by their employers, customers, or professional organizations should be inferred. As an independent group of professionals, no affiliation with the American National Standards Committee, the Association For Computing Machinery, the ACM Special Interest Group on Programming Languages, or any other professional organization exists. While the assitance of Control Data, MITRE, Raytheon, and New England Life is gratefully acknowledged, no approval or knowledge of the group's activities is implied.

II. STORAGE CONSIDERATIONS

Concepts

Knowledge of machine variable storage is important to the FORTRAN user since he must be aware of the layout of variable blocks and because he usually is under a maximum storage constraint most influenced by data storage.

Storage Unit. At the present time the FORTRAN user has a word oriented structure into which he places character or binary information with a packing factor dependent on his machine. Under the guise of a name of one of the other types of variables, short strings are stored and manipulated in a cumbersome, non-standard way. However, retaining the basic storage unit, the word, is appealing since all machines have such an entity. If one character per word were used,

the results would be machine independent at the cost of storage inefficiency on most machines. With clever programming of auxiliary routines, it is possible to perform most of the functions for which other alternatives are recommended. If, however, distinct types for character and binary data are introduced with an appropriate structure for each, then the syntactical forms for operations on numeric types may be extended. With this approach, one defines a character element as a storage unit capable of containing one or more contiguous characters with no defined relationship to a word. Similarly, the bit can be defined as a storage unit capable of containing one or more contiguous characters with no defined relationship to a word. Similarly, the bit can be defined as a storage unit capable of containing one or more contiguous bits with no defined relationship to a word. On any given machine, the storage units of bit, byte (used here to define that amount of storage exactly sufficient to contain one character), and word would have a fixed relationship which could introduce bit-strings and character-strings of variable length as the basic storage units. Such a structure might also be dynamic in nature to allow concatenation to occur in place.

Storage Layout. Packing and alignment with word boundaries are important considerations because they are a source of deviation between machines. With a word based structure, packing is performed by the user in a machine dependent density. Each storage unit is aligned, by definition, at a word boundary. With binary and character elements, however, packing is automatically dense and machine independent while alignment is either left to the user or automatic. If left to the user, full word quantities may not be properly aligned. With automatic alignment (that is, padding of partial words) some difficulties may be experienced by differences in the amount of padding between machines. In order to facilitate equivalencing, it would seem desirable, in any case, to avoid padding between variables of the same type. The same considerations exist with string typing, although an end-of-string marker might be desirable. Such a marker would prevent the implied concatenation of contiguous strings, however, and it would be advisable to handle string length as another piece of information to be stored with the string origin. Some means of achieveing a word-byte-bit synchronization for seleted variables could improve execution time at the expense of storage.

Storage Assignment. Assignment of memory under the first two alternatives would probably be sequential as the declarative is encountered, while the third alternative would probably assign pointers to a separate dynamic array. Sequential assignment is necessary if FORTRAN syntactical forms for numeric data are to be used or extended for character and binary data. If special language features are to be implemented for strings, then there is no reason to require a sequential assignment or even to require static allocation. Of course a static sequential assignment might make dump reading easier. Some sort of micro (parameter) substitution would be valuable in FORTRAN, particularly for use in code which is dependent upon the relationships of different storage entities. With the use of "Hollerith under the guise of" ... micro substitution would permit some form of machine compatibility with maximum packing. Under the other alternatives, it would not be of particular utility for producing standard code. Of course it would always be useful for varying dimensions if some rudimentary expression evaluation at compile time were permitted.

Standardization Problems. A variety of loop-holes through which non-conforming code may appear to be conforming is possible and many compilers provide diagnostics to help detect the presence of such loop-holes. Under the proviso that "you can do anything if you know better" it might be wise not to prohibit

such loop-holes in the standard. Variables in common, equivalence statements, subroutine parameter lists, and arrays present the opportunity for mixed mode operations without the appropriate mixed mode conversion. This problem is particularly serious with character, bit, and string data because it permits non-conforming code. However, since the problem is not restricted to character and binary data types, it should be considered for the language as a whole rather than be treated here.

Declaratives

Type Statement. The type statements now contained in FORTRAN are adequate for Hollerith under the guise of a name of one of the other types. However, if character and binary data are to be recognized as unique data types, it is necessary to add appropriate declaratives such as CHARACTER and BIT. If a more extensive capability is to be implemented, character string and binary string types should be permitted. The character and character-string types would be composed of single bytes. The present Hollerith usage could be retained for use on machines requiring non-standard bytes for external media. Conformity between installations would be aided by a standard character representation. Binary and binary-string types should allocate one or more contiguous bits.

Byte length may be variable depending on the machine or fixed to some standard. For common word lengths, a six bit byte is most likely to pack into a small number of words. For instance, it fits into one word for machines having 6, 12, 18, 24, 30, 36, 42, 48, 54, and 60 bit words, while it fits into two words for machines with 3, 9, 15, 21, 27, 33 bit words; etc. and into three words for machines whose word length is a power of 2. A seven bit byte, on the other hand requires seven words for even packing, except for 7, 14, 21, 28, 35, 42, 49 bit words which are not common. Present practice has been to insert a padding bit to pack five 7-bit characters into 36 bits. The 8-bit byte suffers from an inability to pack evenly into machines with words of length 12, 15, 18, 30, 36, and 60. On 15-bit machines, 8-bit bytes are especially bad, since 15 bytes pack into 8 words. However, the wide usage of 9-track tapes and 8-bit byte addressing machines as well as the rather large representation in the market by IBM makes the 8-bit byte at least as attractive as a 6-bit byte. An unspecified byte length gets around the packing problem, as hardware manufacturers usually pick the optimum for each machine. Both 7 and 8 bit bytes permit the full ASCII character sets. Most manufacturers are offering 8-bit capability on their newer machines.

Equivalence Statements. No change is contemplated for equivalence statements except that some form of a string specification would seem desirable, perhaps with a new declarative. This declarative would define a word, group of words, or string to be equivalent in every sense to an arbitrary group of characters. The format of this declarative is not immediately apparent.

Common Statements. Common statements present alignment and mixed mode problems that have already been discussed under standardization problems. Dimension statements present alignment problems. Data statements must suit the syntax of the variable being preset. Most present compilers have suitable forms, provided that the appropriate constants are available.

Length Attribute. Some form of variable length attribute has been suggested for the FORTRAN language and the IBM syntax of name*length could be used with the following possible interpretations.

Name type	Unit for length
Binary	(Number of bits in a storage unit; default, one bit
	(Maximum number of bits in the strings;
	or (default one bit.
Character	(Number of bits in the character; default, eight.
	(Maximum number of characters in the string;
	or (default one character.
Numeric	(Number of bytes in the data type.
	or (Number of significant figures for the data type.

Note that in the case of numeric data, the first unit implies storage in a byte oriented array, rather than a word oriented array. A real number could be used for length, with a length in bits following a period to imply bit storage of the value instead of byte storage. In the cases of numeric data, alignment might be forced or required to suit arithmetic processes on a given computer. Such numeric declarations would be useful for machine optimization. The second unit for numeric data types would probably guarantee a minimum up to the maximum established by the implementation.

Lambird has proposed that in the case of character data, the length be variable up to the limit imposed by the initial declaration. He further proposed functions and subroutines to redefine and retrieve the length.

III - ARITHMETIC

A. Character Constants

Character constants have been considered to be of three possible forms:

1. 'cc...c'
2. 'cc...c'C (as 'dd...d'B is for bit strings)
3. nHcc...c

Advantages of the first is that it is already employed in most implementations. In format statements, it is not necessary to count the number of characters (as is necessary in 3.), as long as the total number of characters is less than the record size. The form is compatible with the bit-string constant suggested below. Example 'JOHN'

Bit-string constants can be represented in a compatible format, for example: 'dd...d'B or '10110111'B.

In addition to the use of B to denote binary data, O could be used to indicate octal or H used to indicate hexadecimal. In all cases blank would be allowed and would be equivalent to the digit zero.

For special applications, it might be desirable to right adjust the data in a field. This could be indicated by a field width following the type designator.

```
'ABC'C10 =  '       ABC'C
'77'O12  =  '0077'O       = '000000111111'B
```

Statements where used: DATA, FORMAT, character expressions.

B. Variables and Arrays

1. Hollerith Variables

The Hollerith variable is the character data under the guise of a name of one of the other types presented in the present standard.

2. Character Variables

These are single characters each occupying one character-storage unit. A character variable is a datum containing { g / one / a variable number } of characters dependent upon the proposal form selected.

They are defined by a CHARACTER type statement.

Example: CHARACTER CH,LETTER,DIGITS

The value of a CHARACTER variable may be defined by:

a) DATA statement

```
DATA CH/'J'/,DIGITS/'2'/
```

b) Assignment statement

```
LETTER = 'K'
DIGITS = CH
```

c) Input list as specified in section IV

d) Function or Subroutine reference: CALL SUBR (CH,DIGITS)

e) Common

3. CHARACTER Arrays

Character arrays can be represented as strings or arrays of strings. A string in this connotation is a connected sequence of characters. Each character is defined to occupy one character-storage unit (byte) Character arrays are specified by an array declarator in a DIMENSION or CHARACTER statement.

a) For a CHARACTER declarator:

```
CHARACTER  A,B,X
DIMENSION  A(27), B(27,10),X
```

X is a single character variable, A has a length of 27 characters, and B is an array of 10 elements each of length 27 characters.

b) For a string declarator:

```
CHARACTER *27 A,B(10)
CHARACTER A*27,B*27(10),X
```

The *27 indicates the length attribute, that is, the number of character variables in an element..

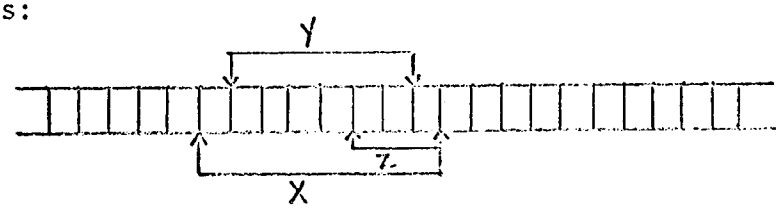
C. Multiple Element Syntax

The following discussion treats only the character data type, since multiple element syntax is not applicable to Hollerith data and is implicit in the definition of string.

Character data must be able to be referenced more than one character at a time. Proposals for accomplishing this are:

a) Equivalencing arrays
b) Element lists (also called implied DO loops or partial arrays)
c) Subarray definition

Let X be a character of length 8 with elements Y and Z of length 6 and 3 as follows:



a) Equivalencing arrays:

```
CHARACTER X(8), Y(6),Z(3)
EQUIVALENCE (X(2),Y(1)),(X(6),X(1))
```

The entire array could be referenced by use of its name without subscripts. Alternatively, the array name could be followed by empty parentheses, null subscripts, or asterisk subscripts to mean the entire array.

b) Element List

The element list, or partial array, can be proposed in three forms, each of which would have defaults designated for missing parameters. The defaulted parameters could be merely null or replaced by an asterisk.

1. Use of a colon to separate the starting character and the length.

Employing the above sketch
X(2:6) is the same as Y
X(6:3) is the same as Z

2. Use of double period to specify start and final

```
X(2..7) is the same as Y
X(6..8) is the same as Z
```

3. Implied DO loops in form similar to present usage

```
(X(I),I=2,7) is the same as Y
(X(1),I=6,8) is the same as Z
```

If C were dimensioned (10,10) as type Character many forms for referencing the entire array would be possible, such as:

```
C          C(1:10,1:10)     C(1..10,1..10)   ((C(I,J),I=1,10)J=1,10)
C()
C(,)       C(:,:)           C(..,..)
C(*,*)
C(*)
```

Element lists would be a generally useful feature in FORTRAN in places other than I-O lists and DATA statements.

Form 3 above is rather clumsy. Forms 2 and 3 cannot be applied to other than the leftmost subscript. The advantages of forms 1 and 2 are illustrated by the following examples of partial arrays.

```
1)  DIMENSION A(10,10)          The 3 characters starting
    . . . A(5,5:3) . . .        at 5,5

2)  DIMENSION A(10,10)          The characters from 5,5 to 7,5 .
    . . . A(5..7,5). . .

3)  DIMENSION A(10,10)          The characters from 5,5 to 7,5 .
    . . . A(5,5:7,5)
```

This form would be allowed in: Data statements, I/O lists, arguments, and assignment statements.

The use of : in the above is only for example. The separator could be any character including comma.

c) Subarray definition:

The subarray definition is similar to the equivalencing arrays paragraph, but employing special syntax for subarray definition:

```
CHARACTER  X(8)
SUBARRAY   Y(6) :X(2), Z(3) :X(6)
```

with the same referencing as in paragraph a).

D. Expressions Involving Characters

1. Comparison

In the current standard, Hollerith data may not be compared or concatenated. These operations can be done for either Hollerith data, or the new CHARACTER data type by defining either intrinsic functions, COMPR or CONCAT, or

operators, (A .EQ.B) or (A-B). The .EQ. appears to be generally acceptable. The A-B will produce problems if mixed mode expressions are allowed. (See following subsection.) The relational operators (.LT., .GT.,etc) could be used in a collating sequence sense.

2. Concatenation and string operations

A serious problem occurs with String expressions: the generation of temporary results. If the length of a temporary cannot be predicted, then the compiler cannot efficiently allocate storage. Such temporaries could be generated as a result of a function attempting to return a string result, or by the concatenation of several strings.

There are several solutions to this problem, none of them completely satisfactory:

1) Do not allow functions, intrinsic or otherwise, to return strings as a result. This includes a prohibition on concatenation.

2) Provide some means of statically allocating storage for intermediate and function results, such as allowing function names to appear in Dimension statements to assign sufficient storage for the return of a string result.

3) Require FORTRAN to dynamically allocate core at execution time when necessary.

The first is restrictive, the second both imposes restrictions and wastes storage in each temporary, while the third is neither efficient nor FORTRAN like. Also, selection of the first requires that all string operations be done by Subroutine where results can be returned to user defined areas.

E. Assignment Statements

A=B where A and/or B may be arrays should be permitted.

The interpretation would be:

If the length of A is the same as the length of B, each element of B will be moved to the corresponding element of A. If A is shorter, the remainder of B should be ignored. If A is longer, it might be filled by null entries (zero, blanks, . False.)

F. Mixed Mode Expressions and Statements

There are several ways of treating Character variables when used in arithmetic context. One suggestion is that the arithmetic value of a string be the ASCII value of the first character. Another possibility is to convert strings to and from integer mode as required, this would be defined only for strings containing characters with the format of integer constants. It is also possible to not define such operations.

Bit strings represent a different problem. One solution is to define no conversion between bit strings (arrays) and other data types and declare that the bit pattern shall simply be transferred between variables.

Consider mixed mode statements of the form:

I = C and C = I

where I is an integer variable and C is a CHARACTER variable. Two interpretataions are possible:

1) ASCII collating sequence using intrinsic or implied functions

I = ASCII (C)
C = IICSA (I)

either defined for the first character of C only, or possibly for all characters resulting in an array of integers, one element per character.

2. The FORMAT-statement type of I conversion:

I = BCDBIN (C)
C = BINBCD (I)

defined as right-adjusted, blank-filled numeric to BCD for as many characters as are converted if C is long enough, (a suggestion).

The first of these two alternatives might be preferred for the implicit mixed-mode conversion because the BINBCD function violates the proposed rule that a function may not return a character string as a result.

Another objection to the second form is that it is defined for a very limited subset of the total character set, the numerals. CHARACTER data is generally used to permit format-free processing of input and output data. For this, the entire character set and a conversion to a standard collating sequence are highly desirable. When character operations have determined the field to be numeric, then some form of Encode/Decode or reread may be used to perform a numeric, formatted conversion.

G. String Functions & Subroutines

Even though there may be problems passing arguments to subprograms on a word oriented machine, it would be a serious limitation to restrict the use of CHARACTER/BIT variables, constants, or expressions in calls to subprograms. (It may be necessary to pass a pointer to the starting character.)

From the earlier discussion of string expressions, it appears that it would be unwise to try to define character functions which can return a value of more than a single character, unless some means of declaring a function dimension were added to FORTRAN. However, since the need for such a facility exists for all types of functions, not just character and bit functions, it will not be considered further here.

In passing arguments (as well as in I/O lists), the extra information (pointer to first character) may make it desirable to pass Character arguments in a manner different than that of other data types. This is not a problem since the Actual and Dummy arguments can be required to be of the same type. However, this feature will make it difficult, if not impossible, to continue to handle character data under the guise of some other data type.

IV. INPUT/OUTPUT CONSIDERATIONS

A. Formats

Formatted I/O will require changes in the FORMAT statement to incorporate CHARACTER and BIT data types. CHARACTER data can continue to use the 'A' descriptor, or a new type (e.g. "C") could be defined. BIT data will probably require a new data type (e.g. "B").

The new descriptors would have a repeat count associated with them, and possibly a width. Unless we consider that the data is stored in strings, the width specification is probably not too meaningful. We could specify that the string length defines 'g' for the string, CHARACTER and BIT data would assume 'g' equal to one in the absence of a STRING declaration. This equivalent 'g' would then interact with the width in the same manner as currently occurs with the "A" descriptor.

It may be desirable to continue to use "A" for CHARACTER data type, but may cause conflicts if it is desired to continue temporarily the use of Hollerith under the guise of another type. This is especially true since it may be desirable to use "C" for COMPLEX data. A possible resolution is that since the width specification is not too meaningful for CHARACTER data, the descriptor "A" would imply CHARACTER and "Aw" would imply Hollerith under the guise of another type. Alternatively, another letter such as "K" could be used to designate character data.

No consensus has been reached on the exact format of BIT conversion, whether the representation should be binary, octal, hex, or even variable.

B. Input-Output lists

There appears to be no reason to restrict or modify the I/O list for CHARACTER of BIT data types.

This would include arrays, array elements, variables, and implied DO loops.

It would be desirable to also allow the forms of expressions and constants on output for all data types.

C. Unformatted Input-Output

Unformatted I/O of CHARACTER and BIT data should be allowed. There may be some problems due to non-adherence to word boundaries; but since no machine compatibility of binary data is assumed or implied in the standard, the problem can be solved by the use of additional control words, etc.

D. Encode/Decode

ENCODE/DECODE have proven to be a reasonable technique for in-core translation to/from Hollerith data. They are easy to learn, moderately efficient, and usually easy to implement.

Since there is rarely any reason to have CHARACTER data unless it is either input or intended for eventual output, the same basic method of conversion would appear desirable.

ENCODE/DECODE should ideally impose no limit on the string length which can be used. There should be a definition of multiple record operations. (The methods used in current implementations are not necessarily the best.)

Because of the inherent inefficiency in ENCODE/DECODE, it would be desirable to have other methods of doing certain conversions, especially INTEGER to/from CHARACTER/BIT.

E. Reread

Experience indicates that the REREAD capability is desirable. A format such as reading from unit zero (or REREAD) will reformat the last record read on any unit. It is not normally difficult to implement and is useful for reading records which could not be read via the original format.

It should be noted that reread coupled with rewrite reduces, but does not eliminate, the necessity for ENCODE/DECODE. Reread reads the last record from a core buffer, rewrite writes into this core buffer. Rewrite could be implemented in the same manner as reread (e.g. by a WRITE on unit zero). It is less desirable than ENCODE/DECODE for the following reasons. An operation to generate the record (rewrite) and another to retrieve it (Reread) would be required, the record size would be limited to an internal work area size, and there would be no easy way to handle multiple record Read/Write statements.

V RECOMMENDED FEATURES

A. Introductory Remarks

This proposal adopts the second of the three alternatives listed in Section I, namely to extend the language by adding new data types with a few changes to the syntax. The new types are:

CHARACTER	for character data, and
BIT	for binary data.

Character and Bit strings are implemented as arrays, where the first dimension could be considered to be the string length.

The proposed Character and Bit additions include the facilities now available by using "Hollerith under the guise of a name of one of the other types", but are independent of the data types now in the language.

A major advantage of the proposed Character and Bit data types is that most of the syntax necessary for defining and using these types is identical with that of the existing types. As a consequence, additions to syntax for the existing types can enhance the new types and vice versa. For example, the multiple-element syntax proposed in the following section as an addition for string and substring referencing can be easily extended to all data types, thereby simplifying the source code required to handle any array.

The Fortran modifications required to implement Character and Binary data types, along with desirable extensions to increase their utility, are presented in Section B. Section C contains examples of the use of the

B. Proposed Fortran Modifications

DECLARATIVE. Two new type declaratives, CHARACTER and BIT, are required with the same syntax rules as any other type declarative. A variable, array element, or array name of type Character or Bit may appear anywhere a variable of another type may appear, with the same restrictions regarding the mixing of variables of different types. One bit of storage shall be allocated for a single element of type Bit and one byte for a single element of type Character. Successive allocations of data elements of the same type shall be continuous, but a word may actually contain any number of characters or bits as long as the packing factor is transparent to the user. For Character data, at least the FORTRAN Character set shall be implemented. Additional characters, representable on the object machine, could be added to the set of character variables. The relationship between data elements occupying a bit, byte, and word is undefined, thus the results of mixing these types in Equivalence statements and subroutine parameters is also undefined. In statements which allocate storage, the compiler will insert whatever padding is required to achieve alignment of byte and word boundaries, between different data types.

CONSTANTS. Character and Bit constants will be delimited by single quotes. A double quote in a Character constant will be interpreted as an embedded single quote. For Bit data, blanks will be interpreted as zeroes. Constants are followed by a character to designate the type of the constant (C or blank for Character data, B for binary, O for octal, and H for hexadecimal). Octal and hexadecimal constants are merely shorthand notations for Bit constants. The length of any constant is the number of character or bit positions it occupies. A Character constant is measured in bytes, a Bit constant in bits. A Character constant may optionally be followed by a justification and length specification which permits specifying only the subfield which contains non-blank characters. Left justification is the default, with L or R specifiable. The complete specification is:

$$\text{'text ...'} \left[\begin{matrix}\text{blank}\\ \underline{\text{C}}\\ \underline{\text{B}}\\ \text{O}\\ \text{H}\end{matrix}\right] \left[\begin{matrix}\text{blank}\\ \underline{\text{L}}\\ \\ \text{R}\end{matrix}\right] \left[\begin{matrix}\text{blank}\\ \text{n}\\ \\ \end{matrix}\right]$$

MODE CONVERSIONS. Character and Bit data may appear in mixed mode expressions with data of other types. Conversion of Character data shall occur via the Integer type for each character. A standard character code is required for such conversions and it is suggested that the USASCII code be specified for character-integer conversion. Bit data is handled in the same manner except that each binary digit is converted to its same value as an integer. Integers other than zero and one are undefined when converted to Bit. Conversion to other data types for the purpose of evaluating mixed mode expressions is done by converting first to Integer, then to the required type.

CHARACTER AND BINARY EXPRESSIONS. Character expressions are limited to a constant, variable, array element, array or function reference. No Character operations are defined. Bit expressions are composed of binary operands and binary operators which are the same as the logical operators .AND., .OR.,.XOR. and .NOT., where a True logical state is the same as a binary 1 for the resulting operation.

CHARACTER AND BIT ASSIGNMENT STATEMENTS. Assignment statements follow the existing rules of Fortran, except that an expression may cause more than one assignment operation if the length of the evaluated expression exceeds one and the receiving variable is an array name. In other words, it is proposed that a statement of the form

ARRAY1 = ARRAY2

mean that the elements of ARRAY2 be moved to ARRAY1. If ARRAY1 and ARRAY2 have unequal dimensions, either the excess elements of ARRAY2 would be ignored or ARRAY1 would be filled out with null data, depending on whether the declared size of ARRAY2 is larger or smaller than the declared size of ARRAY1. A null datum could be defined as blank for a Character array, False for a Logical array, and zero for the other types.

COMPARISON OF CHARACTER AND BIT EXPRESSIONS. The .EQ. and .NE. operators are defined for the Character and Bit data types. The other relational operators are undefined. It is proposed that two arrays may be compared for equality:

ARRAY1 .EQ. ARRAY2
ARRAY1 .NE. ARRAY2

where the assumption would be made that the shorter array is followed by sufficient null data to make the comparison meaningful.

SYNTAX FOR SUBARRAYS. In the assignment and comparison operations, as well as in I/O lists and subprogram arguments, it would be convenient to be able to reference only part of an array. Further, this feature would make some of the more obvious choices for implicit Character functions (such as a concatenation function and a substring function) unnecessary, since these operations could be coded in-line in one or two statements.

The syntax proposed for subarray reference is one of the two following forms:

a) Array (subscriptsof beginning ... subscripts of end)

Ex: CHARACTER ARRAY (4,6,8)
ARRAY (1,1,6 ... 3,1,6) refers to the three elements (1,1,6), (2,1,6) and (3,1,6).

b) Array (length : subscript of beginning)

Ex: CHARACTER ARRAY (4,6,8)
ARRAY (3: 1,1,6) refers to the same three elements.

The choice of syntax (b) would have the interesting feature of providing the ability to omit subscripts from left to right with well-defined results, (with the omission of the length indicating a length equal to the product of the omitted dimensions).

Ex. CHARACTER ARRAY (4,6,8)

Reference	Length	Begins
ARRAY (3 : 1,1,6)	3	1,1,6
ARRAY (: 1,1,6)	1	1,1,6
ARRAY (: 1,6)	4	1,1,6

ARRAY (: 6)	24	1,1,6
ARRAY (23 : 6)	23	1,1,6
ARRAY ( )	192	1,1,1

FORMAT STATEMENTS. Present Fortran format rules apply; however, for character data types the designator nA implies left justification in a field of n characters. Since each element is a list item, truncation cannot occur. Bit data uses the designator nBw for bit strings. w is the number of bits to be converted to (from) each output (input) character (e.g. 1 or blank indicates a binary number base, 3 indicates octal, and 4 indicates hexadecimal).

FORMATED TYPE CONVERSION. A means of in-core conversion between data types according to a format statement is desirable. This feature is already implemented in many compilers as ENCODE/DECODE. Encode has a syntax similar to the formatted Write (Decode similar to the formatted Read), but has an array name instead of a unit number. The array used to contain the formatted form of the variable list is defined as valid for type Character only. The number of characters per internal record should be specifiable to allow multiple record capability. The same standard format scanning and editing capabilities now used for formatted Write/Read would be used for Encode/Decode.

```
Ex.   CHARACTER  ARRAY (200)
      DIMENSION MATRIX (20,10)
      DECODE (20,100,ARRAY)  MATRIX
100   FORMAT (20I1)
```

LENGTH AND OTHER ATTRIBUTES. It is convenient in many areas of Fortran, especially in Character handling, to be able to recover attributes of variables. In particular it is useful to be able to set a length attribute and then use this defined length later. This aviods the need to pass the length as an argument to a subprogram.

It may even be desirable to have available an entire symbol table entry including the name, type, and dimensions of the variable. There are several problems, one of which is the large amount of core which might be required for symbol table information. It is therefore proposed that a declaration of the form SYMBOL TABLE be available on which the programmer could list the symbols to be included; omission of a SYMBOL TABLE statement would cause the table to be generated for all variables.

There are some unresolved problems such as, what is the symbol table for an element of an array? What is the symbol table for a temporary? Is this approach sufficiently efficient?

Among the proposed functions and subroutines are:

LENGTH (symbol)	returns length
CALL SETLEN (symbol,len)	sets length
CALL NAME (symbol,name)	retrieves name of symbol
IDIMS(symbol,0)	retrieves number of dimensions
IDIMS(symbol,n)	retrieves the nth dimension
ITYPE(symbol)	retrieves symbol type (e.g. 0= integer, 1= real, etc.)

The examples in Section C assume the existence of such a Symbol Table for all actual parameters in the calling program.

```
C     ILLUSTRATING CHARACTER-STRING I/O, SUBSTRINGS, AND MIXED MODE.
C
C
C      MAIN PROG - READ SCHEDULE CARDS, ASSIGN TIMES, AND PRINT.
C
      CHARACTER ORGTRM (1), COMT (1), TIME (5), LENGTH (2), FROM (2),
     X    TO (2), TRAINO (3), COMMNT (44), BOARD (2)
      CHARACTERS  HOURS (2), MINS (2)
      EQUIVALENCE (HOURS (1), TIME (1)), (MINS (1), HOURS (4))
C           TIME = HHOMM.
C
      IRDR = 2
      IPTR = 5
   30 CONTINUE
      READ (IRDR, 10) ORGTRM,COMT,TIME,LENGTH,FROM,TO,TRAINO,COMMNT,
     X    BOARD
   10 FORMAT (1A,1X,1A,1X,5A,1X,2A,1X,2A,2X,2A,2X,3A,1X,44A,2A)
C     FILL IN HOURS AND MINUTES IF NOT ALREADY FILLED IN.
      CALL ASSIGN (HOURS, MINS)
      PRINT (IPTR, 20) TIME, TRAINO, COMMNT, ORGTRM, FROM, TO, LENGTH
   20 FORMAT (1X, 5A,2X,3A,1X,44A,1X,1A,1X, 2A,*-*,2A, 2X, 2A,1X,*MINS*)
      IF(COMT .NE. *E*) GO TO 30
   40 CONTINUE
      CALL EXIT
      END

C     ILLUSTRATING CONCATENATION
C
C
C           MAIN PROGRAM - BOB AND CAROL AND TED AND ALICE
C
      CHARACTER TABLE (10, 10), MOVIE (150), AND
      INTEGER  SIZES (10), COUNT, TOTAL
C     CONCATENATING 4 NAMES FROM TABLE.
      DATA   COUNT /4/
      DATA   TABLE (*,1) /*BOB*       /, SIZES (1) /3/
      DATA   TABLE (*,2) /*CAROL*     /, SIZES (2) /5/
      DATA   TABLE (*,3) /*TED*       /, SIZES (3) /3/
      DATA   TABLE (*,4) /*ALICE*     /, SIZES (4) /5/
      DATA   AND /* AND */
C
      TOTAL1 = 1
      DO 10, I = 1, COUNT
      TOTAL = CONCAT (MOVIE, TOTAL1, TABLE (1, I), SIZES (I))
      TOTAL1 = CONCAT (MOVIE, TOTAL, AND, 5)
   10 CONTINUE
C
      PRINT 1, MOVIE (1:TOTAL)
    1 FORMAT (1H1, 150A)
      CALL EXIT
      END
C
C
C
C          CONCATENATION FUNCTION
C
C     INPUT - A = CHARACTER STRING TO WHICH TO APPEND.
C             IA = WHERE TO START APPENDING IN A.
C             B = CHARACTERS TO BE APPENDED.
```

```
C                IB = NO. OF CHARACTERS TO APPEND.
C               VALUE OF FUNCTION CONCAT IS NEW SIZE OF A.
C
      INTEGER FUNCTION CONCAT (A, IA, B, IB)
      CHARACTER A (1), B (1)
      CONCAT = IB + IA - 1
      A (IA...CONCAT) = B (1...IB)
      RETURN
      END

C     ILLUSTRATIONS OF ROUTINES SUGGESTED BY LAMBIRD
      FUNCTION ALF(STRING)
C...  RESULT IS TRUE IF THE CHARACTERS OF STRING ARE ALPHABETIC OR BLANK
C...  AND FALSE OTHERWISE.
      CHARACTER STRING(1)
      LOGICAL ALF
      ALF=.TRUE.
      DO 1 I=1,LENGTH(STRING)
      IF(STRING(I).GT.≠Z≠)GO TO 2
      IF(STRING(I).EQ.≠ ≠)GO TO 1
      IF(STRING(I).LT.≠A≠)GO TO 2
    1 CONTINUE
      RETURN
    2 CONTINUE
      ALF=.FALSE.
      RETURN
      END


      FUNCTION LENGTH (STRING, LENG)
C...  MAINTAINS A TABLE OF STRING LENGTHS IN COMMON.  USES LOCF INTRIN-
C...  SIC FUNCTION TO RETURN ADDRESS OF STRING FOR TABLE ARGUMENT AND
C...  OLDLNG TO RETURN DECLARED LENGTH OF STRING.
      CHARACTER STRING(1)
      INTEGER OLDLNG
      BIT IADD(18)
      COMMON MTABL,LTABL(2,1)
      IADD=LOCF(STRING)
      DO 1 I=1,MTABL
      IF(LTABL(1,I).EQ.IADD)GO TO 2
    1 CONTINUE
      LENGTH=OLDLNG(STRING)
      RETURN
    2 CONTINUE
      LENGTH=LTABL(2,I)
      RETURN
      ENTRY NULNG
C...  HYPOTHESIZES FUNCTION CALL ON LEFT OF ASSIGNMENT STATEMENT PASSING
C...  A PARAMETER AND A VALUE ASSOCIATED WITH THE FUNCTION NAME.
      IADD=LOCF(STRING)
      DO 4 I=1,MTABL
      IF(LTABL(1,I).EQ.IADD)GO TO 5
    4 CONTINUE
```

```
      MTABL=MTABL+1
      LTABL(1,MTABL)=IADD
      LTABL(2,MTABL)=OLDLNG(STRING)
      RETURN
    5 CONTINUE
      IF(NULNG.LT.0) LENG=OLDLNG(STRING)
      LTABL(2,I) = LENG
      RETURN
      END


      FUNCTION OLDLNG (STRING)
      INTEGER OLDLNG
      OLDLNG = IDIMS (SYMBOL, 1)
      RETURN
      END
C     ILLUSTRATIONS OF ROUTINES SUGGESTED BY LAMBIRD
      FUNCTION INDEX(STRING,PATT)
C...  SEARCHES STRING TO FIND PATT.  RETURNS STARTING POSITION IN STRING
C...  IF FOUND, OTHERWISE ZERO.
      CHARACTER STRING(1),PATT(1)
      INDEX=0
      IL=LENGTH(STRING)
      JL=LENGTH(PATT)
      IF((IL.EQ.0).OR.(JL.EQ.0))RETURN
      IF(JL.GT.IL)RETURN
      DO 1 I=1,IL-JL+1
      DO 2 J=1,JL
      IF(STRING(IL+JL-1).NE.PATT(JL))GO TO 1
    2 CONTINUE
      GO TO 3
    1 CONTINUE
      RETURN
    3 CONTINUE
      INDEX=I
      RETURN
      END
      FUNCTION NUM(STRING)
C...  TESTS TO SEE IF STRING IS ALL NUMERIC.
      CHARACTER STRING(1)
      LOGICAL NUM
      NUM=.TRUE.
      DO 1 I=1,LENGTH(STRING)
      IF((STRING(I).LT.≠0≠).OR.(STRING(I).GT.≠9≠))GO TO 2
    1 CONTINUE
      RETURN
    2 CONTINUE
      NUM=.FALSE.
      RETURN
      END
```

```
      SUBROUTINE MATOUT (ARRAY,N)
C     PRINT CORRELATION MATRIX
C     WHERE RANK MAY VARY FROM 2 TO 50
      CHARACTER FMT(16)/*(2H X,I2, F8.3)*/
     X,TABLE(30)/* 1 2 3 4 5 6 7 8 9101112131415*/
      WRITE(6,*) *1CORRELATION MATRIX*
      LIM=N
    1 M= MIN(LIM,15)
      FMT(10...11) = TABLE(2*M-)...2*M)
      WRITE (6,100) (I, I=1,M)
  100 FORMAT (*0*,7X,(*X*,I2,5X))
      DO 5 I=1,N
      WRITE (6,FMT) I, (ARRAY(I,J),J=1,M)
    5 CONTINUE
      LIM = LIM -M
      IF (LIM .GT. 0) GO TO 1
      RETURN
      END
```

VI. BIBLIOGRAPHY

1. Stuart, Fredric "FORTRAN Programming", John Wiley & Sons, Inc. 1970. An exposition of elementary FORTRAN Programming particularly valuable for its comparisons of compilers.

2. "USA standard X3.9-1966," American National Standards Institute, March 1966. The present standard upon which this proposal is based. A reformatted concordance version has also been made available to the group.

3. Frank Engle, Jr., numerous memos containing comments on 4 - 6 below.

4. Robert J. Lambird, P.E. "String Enhancements to ANSI Standard FORTRAN". A comprehensive proposal to incorporate strings into FORTRAN which relies heavily upon functions, a field designator "FLD", and a concatenation operator as well as a unique syntax for strings.

5. SHARE FORTRAN PROJECT "in-core formatting and character specifications", November 1966.

6. WATFIV programming manual, Appendix B, "Character variables with WATFIV". Essentially the same as 5 above.