INTERNAL USE ONLY

INTERNATIONAL COMPUTERS AND TABULATORS LIMITED

Scientific Programming Dept.
ICT 1900 Series

Fortran Note 12
20.9.65.

General Analysis Routines in the FORTRAN IV Compiler

This note describes the dictionary search routine, the 'field extract' routine and the List System data used in the FORTRAN IV Compiler

RECOGNITION OF STATEMENT TYPES

When it is required to discriminate among the various statement types, the complete statement is in some area of store (either STATEMENT or CIMAGE) in character form, with a pointer (POSI) pointing to the first character of the statement name.

For all statements except Assignment statements, Statement Function definitions and some Logical IF statements a routine STS is used to differentiate among the different statement types.

With this routine the first few non-space characters of the statement name are compared against a "Dictionary" to determine the type of statement.

On entry to STS, X2 points to the first character of the Statement name.. On exit, X2 points to the character following the name found and X3 contains a small integer which indicates what dictionary entry was found. If no dictionary entry is found, on exit from STS X2 will be unchanged and X3 will be zero.

e.g. For the statement

DOUBLE PRECISION X

on entry to STS, X2 should be pointing to 'D'. On exit from STS, X2 would be pointing to the character (space or non-space) following 'N'. X3 would be a small integer indicating which dictionary entry was found. Although in theory it is necessary to compare the whole statement name against a dictionary item, in practice only the first few characters of the longer statement names are compared.

For example, the statement

INTEGER X

is recognised as "INTEGER" by searching for a dictionary item INT rather than for the complete name.

Associated with every dictionary item is a number which tells how many non-space characters need to be ignored following recognition to bring the character pointer to the end of the name.

The dictionary therefore consists of two tables - a 'Compare' table, and an associated 'Ignore' table.

These have the following form:

1. Compare Table

   1st word - An Integer which is one less than the number of items in the "Dictionary List". This consists of a series of dictionary items each separated from the next by a space character.

The last item in the list is terminated by a space character. These dictionary items may be any length up to the length for total comparison for the particular statement name.

2. Ignore Table

This is a string of characters (one character per dictionary item) which gives the number of non-space characters of the original statement that are to be ignored following recognition of the dictionary item. The value of the character will be zero if a complete name comparison is made for a particular dictionary item. For example, in the case of INTEGER X the Dictionary item is INT, and the associated Ignore Table character is "4".

The Ignore Table string of characters is stored in the reverse order to the order of items in the Dictionary List, i.e. the character associated with the last item in the Dictionary List is the first item in the Ignore Table and vice versa.

In the FORTRAN Compiler, there are five different dictionaries which are used at different times (i.e. within the Program Description, between program segments, at the beginning of a Program segment, within a MASTER, FUNCTION or SUBROUTINE segment, or within a BLOCK DATA segment).

Before the routine STS is entered to make the dictionary search, the address of the start of the appropriate 'Compare Table' and the address of the start of the associated 'Ignore Table' must be placed in registers LSTP and ISTP of common area LC3.

The value of X3 which is returned upon exit from STS is the number of the entry in the Ignore Table corresponding to the Dictionary Item found. Thus if the item was the last item in the Dictionary List (and therefore the first character in the Ignore Table), the value of X3 would be '1' on exit from STS. If no item is found (i.e. the statement type is not in the current dictionary) the value of X3 on exit from STS would be zero.

X3 on exit from STS can be used as a modifier to branch to the section of the compiler dealing with the type of statement found.

Between the exit from STS and the execution of the appropriate branch instruction, the pointer PØSI is reset to the current value of X2. Thus at the beginning of the analysis of any particular statement (excluding Assignment, Statement Fuction and some Logical IF statements) PØSI is known to point to the first character following the name of the statement.

Assignment statements, Statement Function definitions and Logical IF statements of the form "IF(;) Assignment" are not detected by the STS mechanism.

These statements are distinguished from all the other statements by the fact that they all have an "Equals" character at level zero and do not have a "comma" at level zero anywhere following the equals. Level indicates the number of "("characters which have been found without a corresponding")" character. Thus in the following character string "equals" is at level two.

A + (B + (C = D))

whereas in the following example, "Equals" is at level zero

A = (B + (C + D))

The DO statement is the only statement which has "=" at level zero and also has "," at level zero following the equals.

The "status" of a statement (number of "equals" at level zero and "commas" at level zero following the "equals") is determined at the time the statement is formed in STATEMENT and is available for any statement in a register STAT (of Common area LC3).

For Assignment Statements, Statement Function definitions and Logical IF statements of the form "IF(1) Assignment", the value of STAT is one. For all other statements (except DO) the value of STAT is zero. For DO, the value of STAT is two or three, but this fact is not used in the compiler at present.

Statement Analysis (GETFIELD)

For all statements except FØRMAT, the basic routine used to ascertain the structure of the statement is GETFIELD.

On entry to this routine (at GFE1), PØSI is pointing to the first character of a "field" which is to be found. On exit, PØSI is pointing to the first character following the field terminator.

At this point, the field is in a store area starting at register NAME. The length of the field is held as a counter in the register LENG. The field terminator is in the register TERM (and in X5). The type of field found is given by the value of X4 as follows.

X4 < 0 No field.

In this case, POSI was not originally pointing to a letter or the start of a number or a logical constant. On exit, TERM holds the character originally pointed at unless that character started a character string with the form of a logical or relational operator (e.g. .AND.) in which case TERM holds the internal single character representation for that operator. POSI points to the first non-space character following the original character (or logical or relational operator).

X4 = 0 Alphanumeric Field

In this case POSI was originally pointing at a letter. On exit, the store area starting at NAME will contain the character string starting at the letter and terminating with the first non-sapce, non-alphanumeric character (which will appear in TERM) after that letter. Space characters are ignored in the character string. Thus the field 'AL PH A' is the same as the field 'ALPHA'.

X4 = 1 Floating Point Number

In this case POSI was pointing at the first character of a character string representing an unsigned floating point number. On exit NAME --> NAME + 1 (and X6, X7) contains the normalised internal representation of that number. LENG is 2 (in the counter position).

X4 = 2 Small Integer

In this case POSI was pointing to a digit starting a digit string representing a small integer (< 4096). The terminator of this string is not a point (.) unless this starts a logical or relational operator in which case the terminator is the complete operator (in internal single character form).

On exit NAME (and X7) contains the internal binary representation of the integer and LENG is 1 (in the counter position).

X4 = 3 large integer.

This is the same as for X4 = 2 except that the value of the integer is in the range 4096 to 8388607.

X4 = 4 Double Precision Number

In this case POSI was pointing to the first character of a character string representing an unsigned double precision number.

On exit NAME —> NAME + 3 contains the standardized internal representation of that number. LENG is 4 (in the counter position).

X4 = 5 Logical Constant

In this case POSI was pointing to the first character of a character string representing one of the constants.TRUE. or .FALSE. On exit NAME contains the internal representation of that constant (1 for.TRUE., 0 for FALSE). LENG is 1 (in the counter position).

Except for thecase of logical or relational operators or the exponentiation group (**), as terminators, the value of TERM is the internal representation of the non-space character following the end of a field. At the end of a Statement, the terminator is given the value 63 ('E of S' character).

The character strings which act as terminators are given the following values by GETFIELD.

.NOT.	1
.AND.	2
.OR.	3
.LT.	4
.LE.	5
.EQ.	6
.NE.	7
.GT.	8
**	10

Other than by the interpretation of these character strings the characters 1 - 9 cannot appear as terminators in their own right (as they would always be part of either an alphanumeric or numeric field). 10 is the internal value for ":" which may act as a field terminator but which is not basically in the FORTRAN character set.

Statement Analysis (LISTSCAN)

In the analysis of a statement, many fields which are produced by GETFIELD are names of constants or variable or functions which may have been generated earlier in the program or to which reference is expected to be made at a later stage. All such items are stored in the compiler (along with details about the item) in the "Main List" of the Compiler. All basic communication with the Main List is through a series of routines collectively known as "LISTSCAN".

This series of routines is used to determine whether a given item is in the Main List and to make new list items. The entry points to the system are LSE1, LSE5 and LSEA.

For each entry, the name of the item is in an area starting at NAME. The length (in words) of the item name is in the counter position of the register LENG. Normally these parameters will be generated by GETFIELD

The form of a list item is as shown below

Word 1 Control word used by LISTSCAN

Word 2 Data word used by Compiler

Word 3 etc.Name of Item as presented in NAME etc.

Entry LSE1 is used to investigate the existence of a List Item with a particular name (presented in NAME etc.), If X5 is not zero, the investigation starts at the beginning of the Main List. On exit from LISTCAN, X5 is zero, and X3 contains the address of the first word of the item found. If this item is not the item required (correct name but wrong characterstics), a re-entry at LSE1 (with X5 = 0) will continue the search from the point reached by the previous search.

There are two exit points after entry at LSE1:

1. First word after the call if no list of the correct name is found.
2. Second word after the call if a list item of the correct name is found.

LSE5 is used to insert a List item of the correct name after LSE1 has been used previously. The item produced will have a clear data word. X3 will contain the address of the control word for that item.

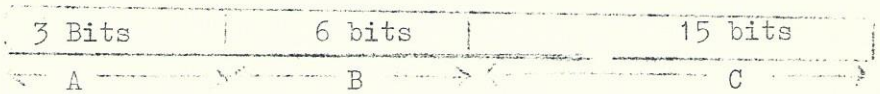
LSEA is the same as LSE5 except that it is used when the item to be inserted will eventually generate a Cue in the segment leader. (eg. subroutine items).

For most List Items the single Data Word is sufficient to hold all the pertinent information about the item. However, for certain classes of items, for example Array names, one word is not sufficient to hold all the required data. In these cases a List Item "Fray" is made (starting at a location given by NAVL) which is long enough to contain the required information. The modifier part of the Data Word of the original List Item is then set to NAVL, and NAVL incremented by the length of the Fray.

Frays are not made within the routines of LISTSCAN but by the sections of the compiler which recognize the item data requirements (such as the sections processing the DIMENSION statement).

The List Item Data Word

The Data Word of a List Item, if not zero, is divided into three parts as shown in the following diagram



'A' indicates the class of item eg. whether it is a variable or an array or a constant or a label etc.

'B' is a qualifier for 'A' giving the particular characterstics of the type for the item. eg. for a Variable - whether it is of mode INTEGER or REAL etc. and whether it is a COMMON variable or a DATA or a standard variable etc.

'C' is an address which is interpreted according to the values of A and B. eg. for an array, it must be the address of the Fray which contains the rest of the data about the array.

When a List item is initially made by LISTSCAN (using entry LSE5 or LSEA) the data word is left clear. Any information which is to be put into the data word is done from the sections of the compiler which are defining the item.

The data word of a List Item may change progressively throughout a compilation as more is discovered about the item. For example consider the quantity X in the following series of statements

```
INTEGER   X
DIMENSION X (10)
COMMON    X
```

Before the statement INTEGER is encountered there is no List Item for 'X'. In INTEGER, a list item will be made for X with 'Type' marked as 'Undefined' and with a qualifier marked 'INTEGER'.

In DIMENSION, the 'Type' will be changed to 'Array, unassigned' and a Fray will be made for the Item. The address part of the Data Word will be the address of the Fray.

In COMMON, the Type will be changed to 'Array, assigned' and the qualifier will be marked as 'COMMON item' as well as 'INTEGER'.

At this stage, the Item X is completely defined and can not be changed during the remainder of the compilation.

The details of the construction of the Data Word for any list item is given on the following pages. In these details, 'List Item Address' refers to the address of the Control Word, used by LISTSCAN (ie. Word 1), of that item.

List Item Data Word

← A →	← B →	← C →	List Item Details

(0) Item Unknown as to type

A	B	C	List Item Details
000	← 3 →	000	List Item Address
000	← 3 →	101	Address of Dummy Arg(LW)

Unassigned as to Type
Dummy Argument Unassigned

(1) Variables

A	B	C	List Item Details	
001	← 3 →	000	Address of Variable(LV)	Normal Variable
001	← 3 →	001	Address of Variable(LC)	DATA variable
001	← 3 →	100	Address of LC referencing Var.	Variable Equivalent to non COMMON Array.
001	← 3 →	101	Address of Dummy Var(LW)	Dummy Variable
001	← 3 →	110	Address of Fray	COMMON Variable

(2) Assigned Arrays

A	B	C	List Item Details	
010	← 3 →	000	Address of Fray	Standard Array
010	← 3 →	001	"	DATA Array
010	← 3 →	101	Address of Array Header Add. (LW)	Dummy Array
010	← 3 →	110	Address of Fray	COMMON Array

(3) Unassigned Arrays

A	B	C	List Item Details	
011	← 3 →	000	Address of Fray	Unassigned Array
011	← 3 →	101	Addr.of Array Header Add. (LW)	Dummy Array

(4a) Functions and Subroutines

A	B	C	List Item Details	
100	← 3 →	000	Address of List Item	Normal Function, Subroutine
100	← 3 →	001	Addr.of Start of routine	Dummy Function, Subroutine
100	← 3 →	010	Address of List Item	Special Intrinsic Function

(4b) Block Names and Other Cued Items

A	B	C	List Item Details	
100		100	Length of Block	Normal COMMON Block name
100		101	Length of Block	BLOCK DATA COMMON Block name
100		110	Length of Block	Special Lower Common Preset
100		111	Peripheral Value	Peripheral Cue

(5) Statement Function Items.

101	<—3—>	X	0	Address (UP) of Statement Fctn	Statement Function
101	<—3—>	X	1	Address of Argument (LW)	S.F. Argument

(6) Constants

110	<—3—>	X	Address of Constant (LC)	Constant

(7) Statement Labels

111	00	<—4—>	Address of Label in Program	Program Label
111	01	X	Address of 'Stepping Stone' (UP)	Label encountered within Statement *
111	10	X	Address of FORMAT Constant (LC)	FORMAT Label
111	11	X	Address of FORMAT Constant (LC)	Label encountered within READ, WRITE **

* This item is made when a label appears within any executable statement other than READ or WRITE if it has not appeared at the beginning of a statement. It is changed to the 'Program Label' item when the label appears at the beginning of a statement (other than FORMAT).

** This item is made when a label appears within a READ or WRITE statement if it has not previously appeared at the beginning of a FORMAT statement. It is changed to a FORMAT label item when the label appears at the beginning of a FORMAT Statement.

For Data Words of Type 0, 1, 2, 3, 4a, 5, and 6, the first three bits of 'B' give the mode of the item as follows:-

1 - Integer

2 - Real

3 - Double Precision

4 - Complex

5 - Logical

The items which require Frays are COMMON variables and Arrays.

A Fray for a COMMON Variable is three words long and has the following form:-

Word 1 Address of a Constant which references the item *

Word 2 Address of Item within COMMON block

Word 3 C/m to COMMON block name **

* For a COMMON Variable in all but a BLOCK DATA segment a constant is required which contains the actual address of the variable. The address of this constant is in the first word of the Fray.
If the COMMON variable is in a BLOCK DATA segment, no constant is required. In this case word 1 contains # 40000000.

** Word 3 must contain the LISTCAN Control word used to find the COMMON Block name Item. It is of the form c/m where 'c' is the length (in words) of the block name, and 'm' is the address of the COMMON block List Item.

Frays for n-dimensional arrays are 4 + n words long and have the following forms.

1. Unassigned Arrays - These are arrays which have yet to be assigned actual space in the object program.

   Word 1 [top 6 bits - Number of Dimensions
   Bottom 18 bits - Address of Array Header in Object Program.

   Word 2 Product of Dimensions (e.g. I.J.K for a I,J,K array)

   Word 3 Sum of Partial Products of Dimension (for an I,J,K array this is 1 + I + I.J)

   Word 4 Zero

   Rest of Fray (n Words) Partial Products (For an I, J, K array, this is 1, I, I.J in the 3 words)

2. Assigned Arrays. These are arrays which have been assigned space in the object program.

   Word 1 top 6 bits - Number of Dimensions
   bottom 18 bits-Address of Array Header in Object Program

   Word 2 Address of 1st element of the Array

   Word 3 1) Zero - standard Array (not COMMON or DATA)
   2) #40000000 - DATA, non-COMMON array
   3) c/m to COMMON block name if a COMMON Array

Word 4 Zero

Word 5 top 6 bits Number of Dimensions

bottom 18 bits Array Base Address *

Rest of Fray (n - 1 words) Partial Products (same as for unassigned array fray e.g. I,I.J for an I,J,K array)

* For an array 'a' defined by a statement of the form 'DIMENSION' $a[d_n]$ the address of an array element $a[s_n]$ is given by the relation

$$a[s_n] = b + q\left(s_1 + \sum_{i=2}^{n} \left(s_i \prod_{j=1}^{i-1} d_j\right)\right)$$

Where 'q' is the number of locations taken by one element (1, 2 or 4 registers) and 'b' is the "Base Address".

V.K. Taylor