To be submitted for publication

UCRL-19463
Preprint

MASTER

AN EXTENDED SET OF FORTRAN INPUT/OUTPUT ROUTINES

E. Close

February 16, 1971

AEC Contract No. W-7405-eng-48

LAWRENCE RADIATION LABORATORY
UNIVERSITY of CALIFORNIA BERKELEY

UCRL-19463

DISCLAIMER

UCRL-19463

TOWARDS A MORE GENERAL SET OF FORTRAN I/O SUBROUTINES

1. INTRODUCTION

Over the past number of years there has evolved in ALGOL a style of input-output that has no direct equivalent in FORTRAN. In particular, the FORTRAN READ/WRITE routines read/write one or more cards/lines for each call and, as usually used, there is a closely associated format statement for each such operation. In ALGOL, however, it is possible to read and write item-by-item [1, 2] and the formating of these operations can be preset in a rather convenient fashion [3, 4].

This difference in input/output is, in general, not because of the difference in the languages; but, instead, seems to be one of style and standardization of the earlier FORTRAN approach. In certain applications, it is advantageous to have the item-by-item control that these routines provide. Below is presented a set of basic FORTRAN subroutines that have been derived from [1, 2, 3, 4]. They have, in so far as possible, the same names, calling sequences, and effect as their ALGOL equivalents.

The routines naturally divide themselves into classes. The first (Table 1) is a basic set of input-output routines that provide a small self-contained system containing most of the features of the I/O package. Next (Table 2), is an additional set of subroutines that allow the user to set input-output parameters and, thus, gain greater control over the data transmission. A third set (Table 3) are derived routines that follow, to a certain degree, the Berkeley style [3, 4] of input-output.

A few elementary character handling routines are furnished as a separate class of subroutines (Table 4).

In so far as is possible, the routines have the same names, calling sequences, and effect as their ALGOL equivalents [1, 2, 3, 4].

The user-oriented subroutines (Tables 1, 2, 3, 4) are all based on a lower level set of subroutines (Table 5) that actually carry out much of the work. These, too, have been isolated separately and written in FORTRAN. It is these routines that are most system dependent and for the purpose of understanding, the input-output package can be considered to be black boxes. Once it is understood what the overall picture is and how the user-oriented routines work, then these lower level subroutines naturally fall into place and are rather easy to understand since they actually implement the package.

While, at first glance, it may appear that such a set of input-output routines is expensive in coding and time, it should be remembered that it is the overall style and structure that is being presented and that the ideas and structure are simple. Thus, a properly rewritten FORTRAN I/O package could easily, I believe, contain the usual FORTRAN routines and also an expanded set of routines such as are presented here. For the present, however, this is presented as an, essentially, stand alone package of FORTRAN subroutines.

Sections 2 through 7 that follow describe, in general, the ideas associated with the various classes of routines. Appendix A has a more detailed description of some selected lower level routines that will help understand key points; thus, easing the task of modifying these routines. Appendix B gives some examples to illustrate the size of this subroutine package and some pertinent comments on their use. For the person who wants to simply use the routines, Tables 1 — 4, Table 6, and Appendix B should suffice.

2. BASIC USER-ORIENTED SUBROUTINES

The basic user-oriented subroutines are tabulated in Table 1 along with a comment that should help in understanding their proper use.

The first two are INMODE, OUTMODE that select the mode of input, output. The following convention has been decided upon. There shall be two input and two output modes. One of these is a standard mode, selected by calling by calling INMODE or OUTMODE with the hollerith constant 1HS. The other is a FORTRAN input-out mode in which standard FORTRAN read/write routines are used for all input-output and this is selected by calling inmode or outmode with the hollerith constant 1HF. The original selection at compile time is standard input and standard output via a data statement.

There is complete compatibility between the standard and FORTRAN mode; however, a certain amount of care must be taken when switching modes since the same I/O routine called in two different modes will, in general, produce two different output actions. More will be said about this later.

The next three routines, FNDUNIT, DRPUNIT, CNTUNIT, are used, respectively, to find, drop, and connect the unit that appears as their integer argument. A more complete discussion of how the input-output channels are arranged can be found in Section 6 where some of the ideas connected with the lower level routines are discussed. The following short summary will prove sufficient to use the I/O package.

All the input/output done using the user level routines works through one input/output channel that is designated the current input/output channel. Initially, the input channel is selected as 60 and the output channel as 61 via a data statement. However, another choice can be made by a call to the subroutine CNTUNIT. For example, I = CNTUNIT (2, 2HIN) will set I = 2, the name of the unit connected, and will connect unit 2 as the current input unit. The previously connected input unit, 60, is stored. In general, this routine will suffice for the user. However, since storage space is finite, the number of units that can be stored is set to six. Thus, the subroutine FNDUNIT can be used to find a unit. For example, I = FNDUNIT(Ø) will establish whether more storage space exists for storing units since I = -1 implies that there is no unit with name zero; that is, no empty place to put another unit. And, similarly, DRPUNIT can be used to purge a unit from the storage area if more space is needed.

It should be noted that a unit can always be connected, even if there is no place to put the currently active one that it is replacing. If there is no storage available, the currently active unit is dropped and the new one connected. It can be reconnected; it will, however, be treated as a new unit and thus the channel characteristics will be reset. This dropping of a unit does not necessarily imply that the one line of information is lost. The exact effect of this unit switching depends on the implementation of the lower level routines.

The two basic input-output routines are IN and OUT. All other routines that transmit data, such as INREAL, OUTREAL, are based on calls to IN/OUT. This has been deliberately done so that the exact code that constitutes their body can be written as is most suitable. The input routine IN(N, UNIT, FMT, A) transmits from the input channel -UNIT- the N items A(1)..., A(N) according to the format -FMT-. The output routine OUT(N, UNIT, FMT, A) behaves similarly when writing on the output channel -UNIT-.

These two routines were principally designed to work in the standard input-output mode. As implemented in the lower level FORTRAN subroutines, the subroutine IN does a simple formated READ(UNIT, FMT) (A(i), i = 1, N) in the FORTRAN mode and completely ignores the format in the standard mode, fmt = 1HS. In the character mode, fmt = 1HA, n characters are packed into A left justified. The output routine OUT does a simple formated WRITE(UNIT, FMT) (A(i), I = 1, N) in the FORTRAN mode and in the standard mode it also does a formated write. The result is, essentially, the same formated write except in the FORTRAN mode the next call to OUT will start on a new line in the usual FORTRAN fashion; whereas, in the standard mode, the write will start in the next column after the last printed character of the preceding output transmission.

I might note in passing that the simple expediency of being able to use a FORTRAN WRITE statement or a READ statement in a mode similar to the here defined standard mode would eliminate the need for IN and OUT. And, as will be seen shortly, keeping track of print and read positions in a user-addressable manner would also be quite useful. I point these out because it is not easy to directly modify some of the existing systems input/output routines, even by people who know the system. The five subroutines CARDS, LINES, SPACES, PAGE, S complete this basic list. CALL CARDS (N) skips N-1 cards on the currently active input unit, the n-th card is then the current data card for standard input and is in the one line holding buffer. CALL LINES (N), produces N line feed carriage returns on the currently active output unit, while CALL SPACES (N) prints N blanks on the currently active output unit. This illustrates two things: one is that such routines should not need the CALL associated with FORTRAN programs and the other is that whenever a unit is not specified in the argument list, the routine operates on the currently active unit. This unit is the last unit set up by some definite action; for example, the compiler via a data statement assigns 60, 61 as input/output units. Likewise, any call to a routine with a unit argument will make that unit the currently active unit.

Subroutine PAGE performs a page eject followed by a carriage return. The top and left margins are set to the current system values.

Subroutine S outputs the string STRING. In ALGOL, a string is well defined. Thus, a nice convenient way of outputing text is to say OUTSTRING (TEXT). This can be done here by defining STRING to be any valid hollerith constant that is itself a valid FORTRAN format.

This, then, completes the basic set of routines. The simple subroutine below illustrates their use:

```
SUBROUTINE TEST
      CALL OUT MODE (1HS)
      CALL PAGE
      CALL OUT (1,2,7H(3F6.2),10.21)
      CALL S(19 H(*THIS IS A TEST*))
      RETURN
END
```

A call to TEST would produce on unit 2 starting on a new page

10.21 THIS IS A TEST

On the other hand, if we call OUTMODE with 1HF, the results are 10.21.

THIS IS A TEST.

3. ADDITIONAL SUBROUTINES FOR SETTING I/O PARAMETERS

The subroutines discussed here and tabulated in Table 2 all deal in some way with the current input/output unit depending on whether they are input or output action. The current unit is defined to be the last referenced unit. The compiler, via a data statement, initially sets the current input unit to 60 and the current output unit to 61.

The two subroutines H LIM and V LIM are margin setting routines. Initially, the left margin is set to 1 and the right margin to 132, the top margin to 5 and the bottom margin to 60. This gives a line length of 132 characters with 56 lines per page. The first character is printed in print position 1, usually a carriage control column in FORTRAN, and the first line of print starts on line 5. The actual spacing on the output printer depends on the printer overflow characteristics. These margins can easily be reset. For example, CALL H LIM (5, 110) causes the left margin to be column 5 and the right column (10).

The two routines READP, PRINTP return as values, the current value of the read position/print position pointer. These values are the next column that will be read/printed. The only reason they have an argument is because CDC FORTRAN [6] requires that function subroutines have one or more arguments.

The two routines IN TAB, OUT TAB are somewhat similar to the tab operation on a typewriter. CALL IN TAB(N) will cause the read pointer to be set to N and the next character read will be from column N. Similarly, for OUT TAB. Obviously, one can easily set a number of tabulation positions by presetting an array in the calling program.

The subroutine IOPARAM can be used to set almost all the input/output variables that the I/O package uses. The first argument is the number of values that are in the arrays MODES, NAMES, VALUES. The variables are contained in the common block with name IO. Their compiled values and definitions are given in Tables 6 and 7.

The next four subroutines PRINTER, PUNCH, INUNIT, OUTUNIT are used to set the currently active input/output units to the requested value. The punch has been selected as 14 and the printed as 61.

The four format routines IFORMAT, RFORMAT, BFORMAT, OFORMAT are used to preset the formats that are used for the outputting of integer, real, logical (Boolean), and octal values when routines are used that have no format specification. In the next section, we shall discuss derived routines and it will be seen that many of these have no format specification. When this is the case, the appropriate preset format is used. These print formats are initially set by the compiler via data statements to standard values which are I23, E23.14, L10, $\bar{0}$23. They can be reset at any time by calling the appropriate format routine. Once set to some value, they retain that value until reset by another call to the format routine. The routines that furnish the format along with the items to be transmitted do not disturb these preset formats.

The field width of the output quantity is specified by the variable FIELD. The quantities will be right justified with zero fill in this field width. The number of decimal digits in a real number are specified by the variable DEC. The logical variable FIXED selects between fixed and floating point representation. Thus, CALL RFORMAT (.FALSE., 23, 14) will set the above standard format for the output of real numbers.

The subroutine EFILE is used to write an end-of-file on the named unit. It is necessary to use this routine when in the standard (partial line) output mode since its use ensures that the one line output buffer will be emptied.

4. DERIVED INPUT-OUTPUT ROUTINES

The routines discussed here are all based on the input-output routines IN, OUT. They obtain their preset values from the common block IO and are essentially independent of one another and of all the other user-oriented subroutines that have been discussed in Sections 2 and 3. These routines include routines similar to those used by CDC ALGOL [2] and those presented by DeVogelaere [3], and also some of the logical variants. Their action is approximately the same as their ALGOL equivalents. However, there are some noticible changes. For example, the CDC ALGOL procedure OUT REAL outputs quantities using their standard format, whereas, the OUT REAL here presented uses a preset format. Also, in the Berkeley style output presented by DeVogelaere, a call to the procedure OUTR(R1, FIXED, FIELD, DEC) followed by a call OUT REAL(UNIT, R2) will cause both R1 and R2 to be output with the format set by FIXED, FIELD, DEC. In other words, the variable format that appears in the argument list resets the preset format. The routines presented here have complete independence of the preset and variable format.

The subroutines can be found in Table 3. We shall limit ourselves here to a few general remarks that will make their use obvious. The idea behind their grouping is the following. To input/output quantities, we must specify a unit from which it will be read/written, a list of quantities to be transmitted, and a corresponding format for that transmission.

If all of these items appear in the argument list, then those values are used. For example, CALL OUTR 3(N, R, . TRUE., 5, 2) outputs on unit N the value of the real variable R using the fixed point format F5.2. The current output unit has now been set to the value of N. If any of the items are missing, then a standard choice is made for the missing item. A missing unit causes the current unit to be used. A missing format causes the appropriate preset format to be used. On output, these are the formats that are set using these routines: IFORMAT, RFORMAT, BFORMAT, OFORMAT. On input, the format selected is the standard (field free) format. For example, CALL OUTR1(R) causes the value of R to be output on the currently active output unit using the preset real format that was set either by the compiler via a data statement or by a subsequent call to RFORMAT.

The function subroutines that appear in Table 3 assume the value of the item read. Since these routines are used in arithmetic expressions, it is, in general, not sufficient to have one routine. For example, J = READ does not work too well because of the implicit mixed mode arithmetic that a FORTRAN such as CDC FORTRAN [6] allows. Thus, they are all explicitly typed. Again, the redundancy in argument for the functions READ I, etc., is because of the requirement that a function subroutine have at least one argument.

The naming of the subroutines is somewhat arbitrary; but, we have tried to adhere to short names (less than seven characters), for user convenience and word size limitations on identifiers, that identify the type of routine and, at the same time, preserve the names of previously defined input/output routines [2, 4] that perform similarly. Logical variants of the same routine have been sequentially numbered.

The subroutines OTI, IOI, and their variants have a STRING argument associated with them that can prove useful in some application. As was previously mentioned, STRING is a hollerith constant which is itself a suitable variable format including left and right parenthesis. A call such as CALL OTI(5, 5H(*B*), 3) will produce the output of B = 5. Thus, the string is assigned the value of the output quantity. Similarly, a call such as J = IOI(I, 5H(*B*), 3) will assign to I and J the value of the next item read from the currently active input unit and also it will write B = N on the currently active output unit; we assume that N was the value just read in. Since there is some disagreement in FORTRAN about the use of multiple statements per line of coding, these routines are in a sense limited to one output action per line. In CDC FORTRAN, one could add the $ delimiter and write multiple statements per line, but the last statement cannot have a $; thus, it is better to leave it out completely.

The next two sets of routines OUTAI, INAI and their variants can be used to output-input arrays. The array element a(ℓ) is the first element input and the element a(u) is the last element input. Multiple dimension arrays can, of course, be handled by simply considering the array as a large one-dimensional array.

The routines INPUT and OUTPUT are formated routines and can be used in either the FORTRAN or standard mode. Since these routines are defined using FORTRAN, they are separately written as INPUT 1, INPUT 2, ...; but, if they were written as system routines, it would seem natural to do as CDC [2] has done and have one routine in which the number of arguments is arbitrary. It is worth pointing out that these routines are very closely related to the READ, WRITE routines of FORTRAN, but have the added feature of being able to have any legal actual parameter as an argument. Thus, for example, the argument A1 could be a function subprogram, or arithmetic expression, as well as a simple variable.

5. CHARACTER-ORIENTED SUBROUTINES

The manipulation of characters using FORTRAN subroutines is usually expensive. Also, there always seems to be an infinite number of routines that can be found useful to have. The routines given here are patterned after similar ALGOL routines [1, 2] that have been set forth as basic character-oriented routines.

The first such routine is CLENGTH which has as its value the length in characters of the argument which is a STRING. A STRING is defined to be a hollerith constant of the form NH(*ANY VALID TEXT*) where the delimiters have been chosen to be (* and *). The delimiters are not counted and, as implemented here, a right delimiter cannot appear in the text.

The routines INCHARØ, OTCHARØ transmit data from the array SOURCE and to the array DESTINATION. A more precise definition is given in Table 4. The action of the routines INCHAR and OUTCHAR are similar to INCHARØ and OTCHARØ, but read/write their results from/to the specified unit. These routines contain the argument LENGTH, the length of the string. This was done because the simple definition of the STRING that is used here requires the actual counting of the characters to obtain its length. This is too expensive to do for every call to these routines, thus, it is furnished as a separate argument. The subroutine C LENGTH furnishes the appropriate length.

The routine EQUIV allows one to obtain the internal representation of an element in an essentially machine-independent manner.

The two routines CHARF, CHARS are character fetching and character storing routines. They are basic routines and are presented here since they are extensively used in the lower level subroutine package and are convenient routines to have available. A transfer of characters can easily be performed by the call such as CALL CHARS (DEST, N1, CHARF(SOURCE, N2)).

6. LOWER LEVEL SUBROUTINE

This set of subroutines exists solely for the purpose of implementing the previously discussed user level subroutine. How they are written, their names, and their coding is largely dependent on the computer used and the computing facilities available. A particular set of these routines suitable for the CDC 6000-series computers has been written in FORTRAN and they are listed in Table 5. The action taken by them is indicated there.

A few comments on that set of routines is given here. There is a data initialization routine INIODAT which initializes all common areas that contain I/O parameters. This would logically be a block data routine. It is presented here as a subroutine to insure its loading when using a system loader to satisfy the unsatisfied externals.

There are two format setting routines. One is for logical values and one for real values. Connected with this is a routine DCINTL that converts the integer N to an internal represention, in this case CDC display code [6] with blank fill, that is suitable for use in a FORTRAN FORMAT statement.

The routine CHNSF actually connects the input/output channels as currently active units and also stores the channel characteristics and is thus quite dependent on the channel organization.

There are two specialized routines READ N and WRITE N that do field-free reading and partial line writing. Connected with the partial line writing are two routines STORE and WRT.

A more detailed description of some of these routines can be found in Appendix A.

7. FIELD-FREE FORTRAN INPUT

The standard input, as defined here, is field-free input. By this is meant that the data input is recongized by the manner in which it is written and a FORMAT specification need not be specified. The following conventions have been chosen.

An integer will be of the form $\pm$ NN. . . N where N are decimal digits. A real member will be of the form $\pm$ NN . . . N. NN. . . NE$\pm$NNN. The distinction between the integers and reals is made by supplying the decimal point for real numbers. If the E is supplied, the real number will be read in an appropriate E format; otherwise, it will be read using an appropriate F format. An octal number can be either $\bar{O}\pm$NN...N or else $\pm$NN...NB where N are octal digits. A logical value is specified by T, TRUE, F, FALSE. Any number (integer, real, octal) value can be followed by RNN...N when N are decimal digits. This will cause that quantity to be read NN...N times. Thus, 5R3 causes the number 5 to be input three times; that is, the next three input requests assign 5 to the input quantity. A comment can be inserted anywhere as */text/* and it will be skipped during input. A string can be input as (*TEXT*) and the array into which it is input will contain (*TEXT*); thus, one can input and then subsequently output a string. Items to be input are separated by a deliminator. This has been chosen to be either a comma or else k or more blanks where k is initially set to 2.

A field may be skipped by inclosing an empty field with two commas such , , . Such fields cause the field to be skipped and the corresponding location to which the value would be assigned is also skipped. The card width has no significance on field free format. The quantities are read as they are encountered.

The following example will illustrate some valid data:

```
+ 5.2,     3   */THIS IS AN EXAMPLE/*
(*A STRING IS READ*)    -6.3E-1R5, +6R2, Ō-777, +11BR2.
```

Thirteen items are read. The first is a real number, the second an integer, the third a string, then five real numbers, two integers, and finally, three octal numbers. These can be placed anywhere on any number of cards. Note, however, that it is possible to delimit the input line (card) length by setting the input left and right margins with subroutine IOPARAM. See Table II and Table 7.

Double precision numbers are written the same as a single precision number except that the E is replaced by a D. Thus, 6.3D - IR5 would denote five double precision numbers. One double precision number is considered as one item in the input lists; however, it occupies two consecutive locations internally. Presently, double precision numbers cannot be skipped with an empty field.

APPENDIX A

Subroutine Descriptions

The information presented here pertains to selected subroutines from the Input-Output Package. It is primarily meant to serve as a guide in understanding the operation of these routines and to point out some of the system type dependencies.

SUBROUTINE INIODAT

This routine is used exclusively as a data setting routine. To insure that it will be loaded when loading programs using a system loader such as the Lawrence Radiation Laboratory's loader, LODE, it has been made a subroutine. The variables appearing in this routine are, essentially, all of the pertinent I/O variables and are defined in Table 7.

SUBROUTINE DCINTL (N, RESULT)

Since this routine converts integer numbers to an internal representation suitable for use in a FORMAT statement, it is machine dependent. The characters per word, CHARPW, is set to 10 and the internal code is assumed to be CDC 6000-series display code [6].

INTEGER FUNCTION CHARF (SOURCE, N)

This routine fetches a character from an array and thus is machine dependent. The characters per word, CHAPWOR, is set to ten and the bits per character, BITPWOR, is set to six. M1 assumes a 60-bit word. The two shift functions LEFT and RIGHT are used. This routine is presently written in CDC FORTRAN and CDC COMPASS. The two routines perform identically. Because of the frequent use made of this routine, the COMPASS version is to be preferred.

SUBROUTINE CHARS (DEST, N, ITEM)

This is, essentially, the inverse of CHARF and the above comments apply to this routine also.

SUBROUTINE STORE (ITEM, UNIT)

STORE is not machine dependent. It is used only in WRITEN and performs the specific task of filling the one line output buffer BUFFER3. To do this it uses subroutine CHARS. As it fills this one line buffer, it keeps track of the right margin, RTMARG, and if the current position of the write pointer, COLCNT3, exceeds the right margin, it then writes out the one line of data, advances the line counter, LNCT, and resets the write pointer to the left margin, LFTMARG. If the line count is larger than the number of lines allowed on a page, RP, then it writes a line with a 1 in column one to cause a page eject, and then spaces the correct number of lines to establish the top margin. The actually emptying of the buffer is done by subroutine WRT.

SUBROUTINE WRT (UNIT, L, U, A)

This routine empties the array A by using a standard FORTRAN WRITE statement. It also reinitializes A to all blanks thus reestablishing

A as a blank line.

SUBROUTINE WRITEN (N, UNIT, FMT, A)

This subroutine, and the two subroutines STORE and WRT that it utilizes, could be replaced by the standard FORTRAN routine WRITE if only there were an option that would let WRITE output less than a record. As presently written, the system routine OUTPUTC associated with the CDC FORTRAN WRITE statement finishes by writing an end of record zero byte thus making it unsuitable for the writing of partial lines since it always writes at least one record.

In order to overcome this difficulty, the following, rather expensive, approach was taken. The CDC FORTRAN [6] routine ENCODE is used to make all formatted writes when in the standard (partial line writing) output mode. These writes are written as 140 character lines into the output buffer BUFFER1. Thus, any formatted write with records (line lengths) less than or equal to 140 characters can be written using the standard FORTRAN formats. This write is done at statement 312.

In this mode of output, there are three formats that are considered special. These are the (/), ( ), (IH1) that represent a new record (new line carriage return), the writing of a blank into the output line (actually BUFFER3), and the page eject operation. Because of the way that WRITEN is constructed, the only way that these operations can be performed is to call the subroutine WRITEN with these special formats.

Thus, the 1, the repetition of parenthesis, and the page eject will not produce the desired results if they appear in a FORTRAN style format. They will be correctly handled by ENCODE, but our subsequent action will destroy this effect. The 1 will be ignored, as will the repetition of parenthesis or repetition of Format. The page eject symbol in Column 1 may or may not end up in Column 1.

After the write operation by ENCODE, the rest of the code is devoted to fetching the written characters out of BUFFER1 and storing them into the one line output buffer, BUFFER3. Initially, BUFFER1 is set to all zeros. The ENCODE write will fill one 140 character line with data. Starting with character 1 in BUFFER3, the characters are fetched one-by-one. The end of the write is signified by obtaining the ↓ ≥ characters. As the characters are obtained, they are stored by the routine STORE.

Restrictions: The format must be exausted in any one write statement. Also, repetition of format or record slashes are illegal. Any one write must be ≤ 138 characters. Thus, the write statements

```
CALL WRITEN(24,UNIT,13H(12A10/12A10),A) or
CALL WRITEN(24,UNIT,7H(12A10),A)
```

are illegal.

Again, all this expensive effort arises because ENCODE has a limit on the number of characters that can be written into a record, and because it is not presently possible to know how many characters were written per record. If this were not the case, one could simply fetch from BUFFER1 and store in BUFFER3 until a zero character 00_8 was obtained.

SUBROUTINE READN (A)

This is a basic field free input routine. If the FORTRAN READ routine had a suitable mode that allowed the reading of partial lines of data in a field free format, then this routine could be replaced by that routine.

The actual data to be read is input via a FORTRAN READ statement at statement 1600. The left margin, INL, and right margins, COLMAX are observed when using READN, data to the left or right of these margins will not be input. The right margin check is made after statement 300. Once the one line input buffer BUFFER3 is filled, the characters are fetched, statement 304, from this array one at a time and are identified in the next statement by checking their position in the array ALPHBT. The character table ALPHBT is taken from Appendix A [7] sequentially starting at letter A and ending at ;. The value of the j-th position in array ALBHBT identifies in octal the j-th character of that Appendix. For example, TYPE = CHARF(ALBHBT, 2) returns TYPE = 12_8; thus recognizing B in the octal numbers written as NN...NB.

Upon entry into READN, the pointers P(I) are set to -1, the buffer NUMBUF(I) to a blank card, and FORMAT 2 to all blanks. If the number being read has just previously been read under a repeat option, that is, the RNN...N was appended to the number, then the read operation consists of a simple assignment and the repeat counter NUMRPT is decreased by one. This happens until the requested number of repetitions has been satisfied. This action is controlled by the logical variable REPEAT just before statement 204.

Each call to READN reads N items before returning. In the case of numbers, this is N numbers requiring N words of A, but in the case of a string, this would be N strings each taking up the space that is required to store (*TEXT*) and in the case of characters, it would be N characters. That is, an item may be a number or a string and the actual storage required to input N of these into A depends on the items. The appropriate counting for this operation is done by setting N1, N2, N3, N4 at statement 200. The 10 assumes that there are ten characters per word.

The rest of READN is broken into small sections that deal with the quantities that are labled in the program. Thus, the section BLANK counts blanks to recognize the delimiter made up of NUMBKS of blanks, currently set to 2.

The section ASTRIK will recognize and skip */TEXT/*. The section LEFTPARANTHESIS will recognize (*TEXT*) and store this string starting at the next available A(I). The section COMMA recognizes the delimiter , . The next two sections TRUE and FALSE recognize the logical values. Any valid display code character, [7] Appendix A, that does not direct the program to a labled section will go to section ALPHABET and be skipped. Any +, -, •, digit will go to the section PLUS, MINUS, POINT, DIGIT. The B, $\overline{O}$, E, R options are recognized in the sections B AND OHO, E, REPEAT.

When a deliminator has been encountered, NUMBKS or more blanks or a comma, a transfer is made to READNUMBER. If the field between deliminators was empty P(1) < 0, then that item is skipped, statement 1500, and it causes the next item to be stored in the <u>next</u> A position; that is, a word is skipped in A. Presently, logical values are excluded from the repeat option. P(1) = 0 shows a logical value was read. If a number was encountered, P(1) > 0, then the numerical field widths are appropriately set into FORMAT2, and the number is read using the CDC routine DECODE, [6] from the number buffer NUMBUF.

Thus, to summarize, the read operation consists of filling a line buffer BUFFER and then recognizing and constructing a number in NUMBUFF. At the same time, the appropriate format is built in FORMAT2. A reference to Figure 1 will explain the significance of the pointers

P(1),..., P(5) which are all initialized to -1.

Figure 1: Number constructed in NUMBUF.

```
Real     P1        P2        P3            P4          P5
         |         |         |             |           |
         ±NN...N  .NN...N  E ± NN...N    RNN...N       ⊔
                                                       ,

Octal      P1          P5                 P1          P5
           |           |                  |           |
         Ō ±NN...N     ⊔                  ±NN...N  B  ⊔
                       ,                           ↑  ,
                                                   |  Omitted from count
                                                   since it is skipped and not
                                                   stored in NUMBUFF

Integer  P1          P5
         |           |
         ±NN...N     ⊔
                     ,
```

SUBROUTINE CNTUNIT(UNIT, MODE)

CNTUNIT connects UNIT either as an input, MODE = 2HIN, or output, MODE = 3HOUT, unit. The last connected units are LSTIN and LSTOUT for input and output. If the UNIT to be connected is already connected, nothing is done. Otherwise, the current unit is stored and UNIT is connected. When UNIT is connected as a new unit, it is placed in the IO buffer area, IOBUFF, and also it is activated as the current unit. If there is no storage space available in IOBUFF, then it is simply connected as a currently active unit and the next request for another unit will cause it to be dropped. Thus, UNIT will always be connected, but may not always be stored.

The actual finding of the units is done by subroutine FNDUNIT which returns a value NAME such that IOBUFF(NAME) contains the name of the unit it was supposed to find. A value < 0 means it was not found.

The actual setting, storing, and fetching of the parameters is done by CHNSF.

SUBROUTINE CHNSF(SF, UNIT, NAME, MODE)

The easiest way to understand CHNSF is to look at the channel structure given in Figure 2. The definitions of the common variables are given in Table 7.

Whenever a new unit is connected, the current value of the channel characteristics residing in the common block IO are used. These can easily be set using HLIM and VLIM for output, or else IOPARAM. These characteristics plus the blank filled 1 line buffer BUFFER are stored, 21 words, in the first available location in IOBUFF. There is room for six units ($6 \times 21 = 126$).

The currently active units are defined by common blocks /IO/ and /BUFFERS/ as indicated in Figures 2 and 3. CHNSF fetches from IOBUFF and stores in these commons, or fetches from these commons and stores in IOBUFF, to establish different channels. The use of these small temporary working areas enables the multiple switching of channels without loosing the channel characteristics or the partially constructed line.

This structure is patterned somewhat after the CDC ALGOL [2] channel structure. These channel characteristics and buffers can be set up internally in the internal buffer areas as has been done for the CDC ALGOL, but the above one line channel structure was chosen in order to have a machine independent FORTRAN code.

Figure 3 shows schematically how the input channels are arrayed.

Figure 2. Channel Structure

Word	1	2	3	4	5	6	7	8 . . . 21
Input	name	INL	INR	INLP	INRP	INRHO	INRHOP	14 word 1 line buffer
Common /IO/ location	1	15	16	18	19	21	22	COMMON /BUFFER/ BUFFER(1),...,BUFFER(14)
Output	name	OTL	OTR	OTLP	OTRP	OUTRHO	OUTRHOP	14 word 1 line buffer
Common /IO/ location	2	9	10	12	13	23	24	COMMON /BUFFER/ BUFFER(52),...,BUFFER(65)

Figure 3. Channel organization of the array IOBUFF in common IOBUFF

1 | name | characteristics | 1 line buffer | 22 | name | characteristics | 1 line buffer | . . .

106 | name | characteristics | 1 line buffer |

currently active units in common IO

| 1st in | characteristics | 1 line buffer | | 1st out | characteristics | 1 line buffer |

APPENDIX B

Use of Routines by LRL Users

The use of these routines is quite simple and is illustrated by an example given here. The information presented in Tables 1 - 7 should prove sufficient to use them correctly.

A few comments should, however, be made:

1. The standard input and output units are 60 and 61, respectively. If these are suitable, then no units need ever be referenced.
2. The routines compile and execute under RUNF and FTN (2.3).
3. The best use of these routines is made using the library feature of LODE.
4. The subroutine organization is given in Table 3 and Table 8. If a loader is not used to load the routines by satisfying unsatisfied externals, then these subdivision will prove useful. Deck A is required. Essentially, all these routines are used. Deck I is required to set the formats for those routines of Table 3 that do not have a format. The rest of the decks are independent and can be used as desired. The numbers in Table 3 refer to decks; for example, deck 1.1, etc.

Table 1

Basic IO Subroutines

Subroutine	Comment
s inmode (mode) s outmode (mode)	For input, if mode = 1HF, then normal FORTRAN formatted reading is performed. If mode = 1HS, then the standard field free input is used. For output, if mode = 1HF, then normal FORTRAN formatted writing is assumed. If mode = 1HS, then the standard partial line writing routine is used.
if fnd unit (unit)	The IO buffer area is searched for the channel with name -unit-. If unit is found, then fnd unit = name where IOBUFF (name) contains the name -unit-. If the channel -unit- is not found, then fnd unit = -1. Note, an empty IOBUFFER area channel has unit 0 assigned to it.
if drp unit (unit)	The IO buffer area is searched. If the channel with name -unit- is found, then it is dropped from the IO buffer area. If it is not found, then drp unit = -1.
if cnt unit (unit, mode)	The channel with name -unit- is connected to the standard input/output channel area. If mode = 2HIN, then it is connected as an input channel; if mode = 3HOUT, then it is connected as an output channel. The value of cnt unit is the name of the unit connected.

Table 1 - contd.

Subroutine	Comment
s in (n, unit, fmt, a) s out (n, unit, fmt, a)	The n quantities a(1), ..., a(n) are transmitted from or to -unit- using the format -fmt-. When in the FORTRAN writing mode, fmt is any valid FORTRAN variable FORMAT including left and right parenthesis. If the standard input/output mode is used, then for input fmt is field free input and fmt = 1HA is character input. That is, n characters, six bits/character are packed in a left justified. Whereas, for output, fmt can be a valid FORTRAN variable FORMAT; provided / and repetition of parenthesis, without repetition factor, and repetition of the FORMAT before the a(i) are transmitted, are excluded. That is, the line feed carriage return and/or paging operations are not done by the format while transmitting the items a(1),...,a(n). A standard output format can be invoked by setting fmt = 1HR, 1HI, or 1HL for real, integer, and logical values.
s cards (n)	n - 1 cards are skipped, the n-th card is the current data card for standard input.
s lines (n)	n new line carriage returns are performed on the current output unit.
s spaces (n)	n blank spaces are written on the current output unit.
s page	A page eject is performed along with a carriage return.
s s(string)	The string -string- is output on the current output unit.
s nlcr	A new line carriage return is performed on the current out unit. Forces write on teletype.

Table 2

Additional Subroutines for Setting IO Parameters

Subroutine	Comment
s h lim (left, right)	The left and right margins are set on the current output unit. Left = 1 and right = 132 gives a full CDC print line.
s v lim (top, bot)	The top and bottom margins are set on the current output unit. Top = 1 in the first line the printer prints, bot = 60 would thus give 60 lines/page. The actual margins obtained is dependent on the local printer margins.
if readp(p)	The present value of the reading position pointer is returned. This is the next position that will be read by a standard (field free) read on the current input unit.
if printp(p)	The present value of the print position pointer is returned. This is the next position that will be printed on the current output unit in standard (partial line writing) output mode.
s in tab(colm)	The reading position pointer is set to the value of colm. Thus, the next position read from the current input unit will be colm.
s out tab(colm)	The print position pointer is set to colm. Thus, the next position printed on the current output unit will be colm.
s ioparam(num, modes names, value)	Num values of the input/output variables can be changed using ioparam. Modes[i] = 0, (i = 1,..., num), causes iov[name[i]] = value[i]. Mode[i] = 1 causes value[i] = iov[name[i]]. For the input/output variables iov, we have the following:

Table 2 - contd.

Subroutine	Comment	
	Names	items in common block IO
	1 – 8	inunit, outunit, ifield, bfield, rfield, rdec, rfixed, ofield,
	9 – 14	otl, otr, otp, otlp, otrp, otpp,
	15 – 20	inl, inr, inp, inlp, inrop, inpp,
	21 – 24	inrho, inrhop, outrho, outrhop,
	25 – 26	std, fortrn,
	27 – 44	ifmt(3), rfmt(6), lfmt(3), ofmt(3), stdfmt(3),
	45 – 59	psifmt(3), psrfmt(6), pslfmt(3), psofmt(3),
	60 – 67	lefts, rights, lefts 1, rights 1, ℓ1, r1, ℓ2, r2
	Their definitions are given in Table 7.	
s printer s punch	The current output unit becomes 61 for the printer or 14 for the punch.	
s inunit(unit) s outunit(unit)	The current input/output channels are unit.	
s iformat(field) s rformat(fixed, field, dec) s bformat(field) s oformat(field)	The format can be preset for those routines that are a preset format. The formats are: integer (I field) real fixed = .true. (F field. dec) fixed = .false. (E field. dec)	

Table 2 - contd.

Subroutine	Comment
	logical (L field) octal (O field) The formats are separately set and are not destroyed when routines using a variable format are called.
s efile(unit)	When using this set of input/output procedures, it is necessary to use this subroutine to write an end file.

Table 3

Derived IO Subroutines

Note: See Table 6 for argument definitions.

Current Unit Variable Format	Current Unit Preset Format	Variable Unit Preset Format	Variable Unit Variable Format
	(1.1) if readi(i) rf readr(r) lf readb(b) if readio(o) rf readro(o)	(1.2) if readi1(unit) rf readr1(unit) lf readb1(unit) if readio1(unit) rf readro1(unit)	
(2.0) s outi(i, field) s outr(r, fixed, field, dec) s outb(b, field) s outo(o, field)	(2.1) s outi1(i) s outr1(r) s outb1(b) s outo1(o)	(2.2) s outi2(unit, i) s outr2(unit, r) s outb2(unit, b) s outo2(unit, o)	(2.3) s outi3(unit, i, field) s outr3(unit, r, fixed, field, dec) s outb3(unit, b, field) s outo3(unit, o, field)
		(2.22) s outint(unit, i) s outreal(unit, r) s outbool(unit, b) s outoct(unit, o) s otarray(type, n, unit) s outstr(unit, s)	
	(3.1) s ini1(i) s inr1(r) s inb1(b) s ino1(o)	(3.22) s inint(unit, i) s inreal(unit, r) s inbool(unit, b) s inoct(unit, o) s inarray(n, unit, a) s instr(unit, string)	

Table 3 - contd.

Current Unit Variable Format	Current Unit Preset Format	Variable Unit Preset Format	Variable Unit Variable Format
(4.0) s oti(i, string, field) s otr(r, string, fixed, field, dec) s ot b(b, string, field) s oto(o, string, field)	(4.1) s oti(i, string) s otr1(r, string) s otb1(b, string) s oto1(o, string)	(4.2) s oti2(unit, i, string) s otr2(unit, r, string) s otb2(unit, b, string) s oto2(unit, o, string)	(4.3) s oti3(unit, i, string, field) s otr3(unit, r, string, fixed, field, dec) s otb3(unit, b, string, field) s oto3(unit, o, string, field)
(5.0) if ioi(i, string, field) rf ior(r, string, fixed, field dec) lf iob(b, string, field) if ioo(o, string, field)	(5.1) if ioi1(string) rf ior1(string) lf iob1(string) if ioo1(string)		
(6.0) s outai(ia, ℓ, u, field) s outar(ra, ℓ, u, fixed, field, dec) s outab(ba, ℓ, u, field) s outao(oa, ℓ, u, field)	(6.1) s outai1(ia, ℓ, u) s outar1(ra, ℓ, u) s outab1(ba, ℓ. u) s outao1(oa, ℓ. u)	(6.2) s outai2(unit, ia, ℓ, u) s outab2(unit, ba, ℓ, u) s outab2(unit, ba, ℓ, u,) s outao2(unit, oa, ℓ, u)	(6.3) s outai3(unit, ia, ℓ, u, field) s outar3(unit, ra, ℓ, u, fixed field, dec) s outab3(unit, ba, ℓ, u, field) s outao3(unit, oa, ℓ, u, field)
	(7.1) s inai1(ia, ℓ, u) s inar1(ra, ℓ, u) s inab1(ba, ℓ, u) s inao1(oa, ℓ, u)	(7.2) s inai2(unit, ia, ℓ, u) s inar2(unit, ra, ℓ, u) s inab2(unit, ba, ℓ, u) s inao2(unit, oa, ℓ, u)	

Table 3 - contd.

Current Unit Variable Format	Current Unit Preset Format	Variable Unit Preset Format	Variable Unit Variable Format
			(8, 3) s input1(unit, fmt, a1) . . . s input5(unit, fmt, a1, . . . a5) s inputn(n, unit, fmt, a1, . . . , an)
			(9, 3) s output1(unit, fmt, a1) . . . s output5(unit, fmt, a1, . . . , a5) s outputn(n, unit, fmt, a1, . . . , an)

Table 4

Character Oriented Subroutines

Subroutine	Comment
if clength(string)	The length (number of characters) of the string -string- is returned as the value cf c length.
s incharφ(source, colct, string, i, length) s otcharφ(dest, colct, string, i, length)	If the character in position -colct- of array -source- is found in the string -string- of length -length-, then i is the position count(from the left) of that character in string with the first character having position 1. If the character is not found in string, then i = 0. If 00_8 is in position colct, then i = -1. The i-th character of the string -string- with length -length- is stored in position -colct- of the array -dest-. If i > -1, then 00_8 is stored.
s inchar(unit, string, i, length)	The next character is read from the input channel -unit-. The string -string- with length -length- is searched; if the character is found, then i is its position in string with the first character having position 1. If the character is not found, then i = 0. If the internal representation of the character input is 00_8, then i = -1.
s outchar(unit, string, i, length)	The i-th character of the string -string- with length -length- is output on the channel -unit-. If i = -1, then 00_8 is output. If i > length, then nothing happens.
if equiv(string)	The value of equiv is the internal representation of the string -string-. Thus, if string (without the delimiters) was read using an A format into the variable x, x = equiv(string) would be true. Restriction, only on word is tranferred by equiv.

Table 4 - contd.

Subroutine	Comment
if charf(source, n)	The internal representation of the n-th character, right justified zero fill, is fetched from source and returned as the value of charf.
s chars(dest, n, item)	The integer item is stored as the n-th character in dest. Item is assumed to be right justified zero fill. This routine is the inverse of the routine charf.

Table 5

Lower Level Subroutines

Subroutine	Comment
s iniodat	This subroutine defines and unitializes the input/output variables in the common block IO.
s set lfmt(b, field, lfmt) s set rfmt(fixed, field, dec, rfmt)	A true or false logical format is set in lfmt with field width -field-. A fixed (F) or floating(E) format is set in rfmt with field width -field-. There are dec digits after the decimal point.
s dcintl(n, result)	The integer n is converted to CDC display code and stored left shifted with blank fill in -result-. This converts numbers to an internal representation suitable for use in a FORTRAN variable FORMAT.
if chnsf(sf, unit, name, mode)	Chnsf establishes the characteristics for the input (mode = 2hin) /output (mode = 3hout) channel with name -unit-. If unit already exists, an exchange is performed with the iobuffer area. If the unit does not exist, it is established either as a new or temporary unit.
s store(item, unit)	The integer -item-, right justified zero fill, is stored in the one line output buffer, buffer 3 of common block buffers. Carriage return, line feed, and paging operation are performed as required. Item is assumed to be the internal representation of a valid character.
s wrt(unit, ℓ, u, a)	The array a(ℓ), . . . , a(u) is written. via a FORTRAN WRITE statement, on the output unit -unit-. After the completion of the write, the array elements a(ℓ), . . . , a(u) are reset to blank characters.

Table 5 - contd.

Subroutine	Comment
s readn(n, unit, a)	This is the field free input routine. The data is identified and the appropriate format is established. Then the incore formatted read routine DECODE reads the data.
s writen(n, unit, fmt, a)	An incore formatted write is performed using the subroutine ENCODE. It uses the subroutine store to transfer to a one line holding buffer. Appropriate action is taken for the special formats (/), ( ), (1H1) representing a line feed carriage return, blank character, page eject. It is assumed that not more than two 140 character lines are written for one call to writen.

Table 6

Definitions

Name	Comment
mode	A hollerith constant specifying a mode of operation. For example, INMODE (1HS) gives standard field free input. While CNTUNIT(2, 2HIN) connects unit 2 as an input unit.
unit	An integer specifying an input/output unit.
n	An integer representing how many. For example, LINES(N) gives n line feed carriage returns, OUTPUT(N,...) outputs N items.
fmt	A FORTRAN hollerith constant of the form nH(...) where ... is any legal FORTRAN FORMAT. When in standard (partial line) output mode, /, repetion of parenthesis, and repetition of the format before exhaustion of the list are not permitted. In addition, the following are permitted for output formats: 1HR standard format real E23.14 1HI standard format integer I23 1HL standard format logical L23
string	If the string is to be printed, for example OUTSTR (STRING), then a hollerith constant of the form nH(...) where ... is the usual FORTRAN text such as *TEXT* or else nHTEXT. If the string is used in a character routine such as C LENGTH(STRING), then a hollerith constant of the form nH(*...*) where ... consists of any valid alphanumeric characters. Note that (*...*) is also a valid CDC FORTRAN [6] format string so that there need be no conflict if all strings are written as (*...*). If the string is read in, then it is of the form (*TEXT*). INSTR(STRING), OUTSTR(STRING) will input and then output, but the delimiters are missing from the printed string and must be supplied to again input the string. Note: The characters *), asterisk right parenthesis with no blank, cannot appear within the string. In partial line writing mode, the string must be of the form (*TEXT*). The length of the string is unlimited.

Table 6 - contd.

Name	Comment
a	An array
left	An integer specifying the left margin. The first character printed appears in column left.
right	An integer specifying the right margin. The last character printed on a line will always be in a column less than or equal to right. Overflow is printed on the next line starting at the left margin.
top	An integer specifying the top margin of the page. The actual position of the margins depends on the printer overflow margins. A page eject is performed by writing a 1 in Column 1 and filling that line with blanks. The next print line has top = 1.
bot	An integer specifying the bottom margin of the page. Counting from top = 1, bot is the last line printed before a page eject is performed.
p	The column position of the next item to be read/printed.
field	The total field width that the printed item will occupy. The number, or logical value, is right justified in the field.
dec	The number of digits to the right of the decimal point.
fixed	The value .TRUE. means F format. The value .FALSE. means E format.
i	integer
r	real
b	logical (Boolean)
o	octal
ia	integer array
ra	real array

Table 6 - contd.

Name	Comment
ba	logical (Boolean) array
oa	octal array
ℓ	Integer specifying the lower bound, first element a[ℓ].
u	Integer specifying the upper bound, last element a[u].
type	1HR real array 1HI integer array 1HL logical(Boolean) array 1H$\bar{O}$ octal array
a1, a2, . . .	An argument of an arbitrary type. Obviously, it must agree with the format specification.
source	An array from which quantities are read.
dest	An array into which quantities are stored.
<u>if</u>	integer function
<u>rf</u>	real function
<u>ℓf</u>	logical function
<u>s</u>	subroutine

Table 7

Common Input/Output Variables

Common /IO/	Variable Name	Preset Value	Variable Definition
1	inunit	60	The current input unit. Sometimes called lstin.
2	outunit	61	The present output unit. Sometimes called lstout.
3	ifield	23	Preset integer field width.
4	bfield	10	Preset logical field width.
5	rfield	23	Preset real field width.
6	rdec	14	Preset number of decimals in the preset real field.
7	rfixed	.false.	Preset selection of fixed(.true.) or floating (.false.) point representation of real numbers.
8	ofield	23	Preset value of the octal field width.
9	otl	1	left margin - output
10	otr	132	right margin - output
11	otp	132	number of characters per line - output.
12	otlp	5	top margin - output
13	otrp	60	bottom margin - output
14	otpp	60	number of lines per page - output
15	inl	1	left margin - input
16	inr	73	right margin - input
17	inp	80	Number of characters per card (line)- input

Table 7 - contd.

Common /IO/	Variable Name	Preset Value	Variable Definition
18	inlp	1	first card(line) - input
19	inrp	1,000,000	last card (line) - input
20	inrpp	1,000,000	
21	inrho	81	Next character read is in column inrho
22	inrhop	0	The number of cards (lines) that have been read.
23	outrho	1	The next character output is in column outrho
24	outrhop	5	The current output line is outrhop.
25	std	1HS	
26	fortrn	1HF	
27	ifmt(1)	2H(I	The variable integer format is ifmt. The field width is placed
28	ifmt(2)		in ifmt(2).
29	ifmt(3)	1H)	
30	rfmt(1)	1H(	The variable real format is ifmt
31	rfmt(2)		E or F is placed in rfmt(2)
32	rfmt(3)		The field width is placed in rfmt(3)
33	rfmt(4)	1H.	
34	rfmt(5)		The number of decimal digits goes in ifmt(5).
35	rfmt(6)	1H)	
36	lfmt(1)	2H(L	The variable logical format is lfmt.
37	lfmt(2)	2H23	The field width is placed in lfmt(2)

Table 7 - contd.

Common /IO/	Variable Name	Preset Value	Variable Definition
38	lfmt(3)	1H)	
39	ofmt(1)	2H(0	The variable octal format is ofmt.
40	ofmt(2)	2H23	The field width is placed in ofmt(2).
41	ofmt(3)	1H)	
42	stdfmt(1)	2H(0	A standard octal format is furnished by stdfmt.
43	stdfmt(2)	2H(23	
44	stdfmt(3)	1H)	
45	psifmt(1)	2H(I	The preset integer format is psifmt.
46	psifmt(2)	2H23	The field width goes in psifmt(2).
47	psifmt(3)	1H)	
48	psrfmt(1)	1H(	Preset real format.
49	psrfmt(2)	1HE	E or F is placed in psrfmt(2).
50	psrfmt(3)	2H23	The field width is placed in psrfmt(3).
51	psrfmt(4)	1H.	
52	psrfmt(5)	2H14	The number of decimal digits is placed in psrfmt(5)
53	psrfmt(6)	1H)	
54	pslfmt(1)	2H(L	Preset logical format.
55	pslfmt(2)	2H10	The field width is placed in pslfmt(2).
56	pslfmt(3)	1H)	
57	psofmt(1)	2H(0	Preset octal Format. The field width is placed in psofmt(2).
58	psofmt(2)	2H23	
59	psofmt(3)	1H)	
60	lefts	2H(*	Left string delimiter - internal.

Table 7 - contd.

Common /IO/	Variable Name	Preset Value	Variable Definition
61	right s	2H*)	Right string delimiter - internal.
62	lefts 1	2H(*	Left string delimiter - external
63	rights 1	2H*)	Right string delimiter - external
64	L1	2	The number of characters in lefts.
65	R1	2	The number of characters in rights.
66	L2	2	The number of characters in lefts 1.
67	R2	2	The number of characters in rights 1.

Common /BUFFERS/			
Buffer(1) . . . Buffer(14)	1OH . . . 1OH		A one line input buffer into which data is read by a standard FORTRAN READ statement. It is initially set to all blanks.
Buffer2(1) . . . Buffer2(37)	0 . . . 0		The incode formatted write using ENCODE writes into this buffer.
Buffer3(1) . . . Buffer3(14)	1OH . . . 1OH		Output lines are constructed in buffer3 which is then written using a standard FORTRAN WRITE statement.

Table 7 - contd.

Common /IO BUFF/	Variable Name	Preset Value	Variable Definition
1	Name		A variable that is used to locate the names of the units in the input/output channels stored in the array iobuff. The name of the unit is in iobuff[name] .
2	Max Name	106	The maximum location in which an input/output name can be found in array iobuff.
3	Bufflth	14	The length of the one line buffer associated with a channel.
4	Chlth	7	The length of the channels in which are stored the unit characteristics.
5	Chnpbuf	21	The total length of the channel and one line buffer. Thus, Chnpbuf = Chlth + bufflth.
6-131	Iobuff		Unit 60 for input and unit 61 for output are originally set. This array is used to store up to six input/output channels with their associated characteristics.

Table 8

Subroutine Origination

```
*deck A
  iniodat
  set lfmt
  set rfmt
  deintl
  chnof
  store
  wrt
  inmode
  outmode
  fndunit
  drpunit
  cntunit
  in
  readn
  out
  writen
  charf
  chars
  c length
*deck AA
  in
  read 1
  datai
  rdnum
*deck B
  cards
  lines
  spaces
  page
  s
*deck C
  hlim
  vlim
*deck D
  read p
  print p
*deck E
  intab
  outtab
*deck F
  ioparam
*deck G
  printer
  punch
*deck H
  inunit
  outunit
*deck I
  informat
  rformat
  bformat
  oformat
*deck J
  efile
*deck K
  inchar0
  otchar0
*deck L
  inchar
  outchar
*deck M
  equiv
```

EXAMPLES

See the following pages.

```
                PROGRAM TEST(INPUT=1000,OUTPUT=1000,TAPE60=INPUT,TAPE61=OUTPUT,
              *          TAPE1=INPUT,TAPE2=OUTPUT)
        *       BEGIN
000002                INTEGER I,J,READI
000002                INTEGER CNTUNIT
000002                LOGICAL B,READB
000002                REAL A,READR
000002                J= CNTUNIT(2,3HOUT)
000004                J= CNTUNIT(1,2HIN)
000010                CALL OUTMODE(1HF)
000011               CALL HLIM(2,130)
000013                DO 10 J=1,2
000015                CALL OUTSTR(2,19H(* THIS IS OUTSTR*))
000017                CALL S(15H(* THIS IS S *))
000021                CALL OUTPUT(2,19H(* THIS IS OUTPUT*))
000023                CALL LINES(2)
000025                B= .TRUE.
000026                A= 1.0
000027                I= 2
000031                CALL OUTPUT1(2,11H(* A=*F6.2),A)
000033                CALL OUTPUT1(2, 9H(* I=*I5),I)
000036                CALL LINES(2)
000040                CALL OUTI(I,2)
000042                CALL OUTI(I,10)
000044                CALL OUTR(A,.TRUE.,3,1)
000047                CALL OUTR(A,.TRUE.,10,3)
000052                CALL OUTR(A,.FALSE.,10,3)
000055                CALL LINES(2)
000057                CALL OUTB(B,8)
000061                I= READI(1)
000063                A= READR(1)
000065                B= READB(1)
000070                CALL IFORMAT(5)
000071                CALL RFORMAT(.TRUE.,6,3)
000074                CALL BFORMAT(10)
000076                CALL OUTINT(2,I)
000100                CALL OUTREAL(2,A)
000102                CALL OUTBOOL(2,B)
000104                CALL LINES(2)
000106                CALL OUTMODE(1HS)
000110       10       CONTINUE
000112                CALL PAGE
000113                CALL PLOT
000114          END
```

.NO.ID. 1

PROGRAM LENGTH INCLUDING I/O BUFFERS
002262

FUNCTION ASSIGNMENTS

STATEMENT ASSIGNMENTS

BLOCK NAMES AND LENGTHS

VARIABLE ASSIGNMENTS

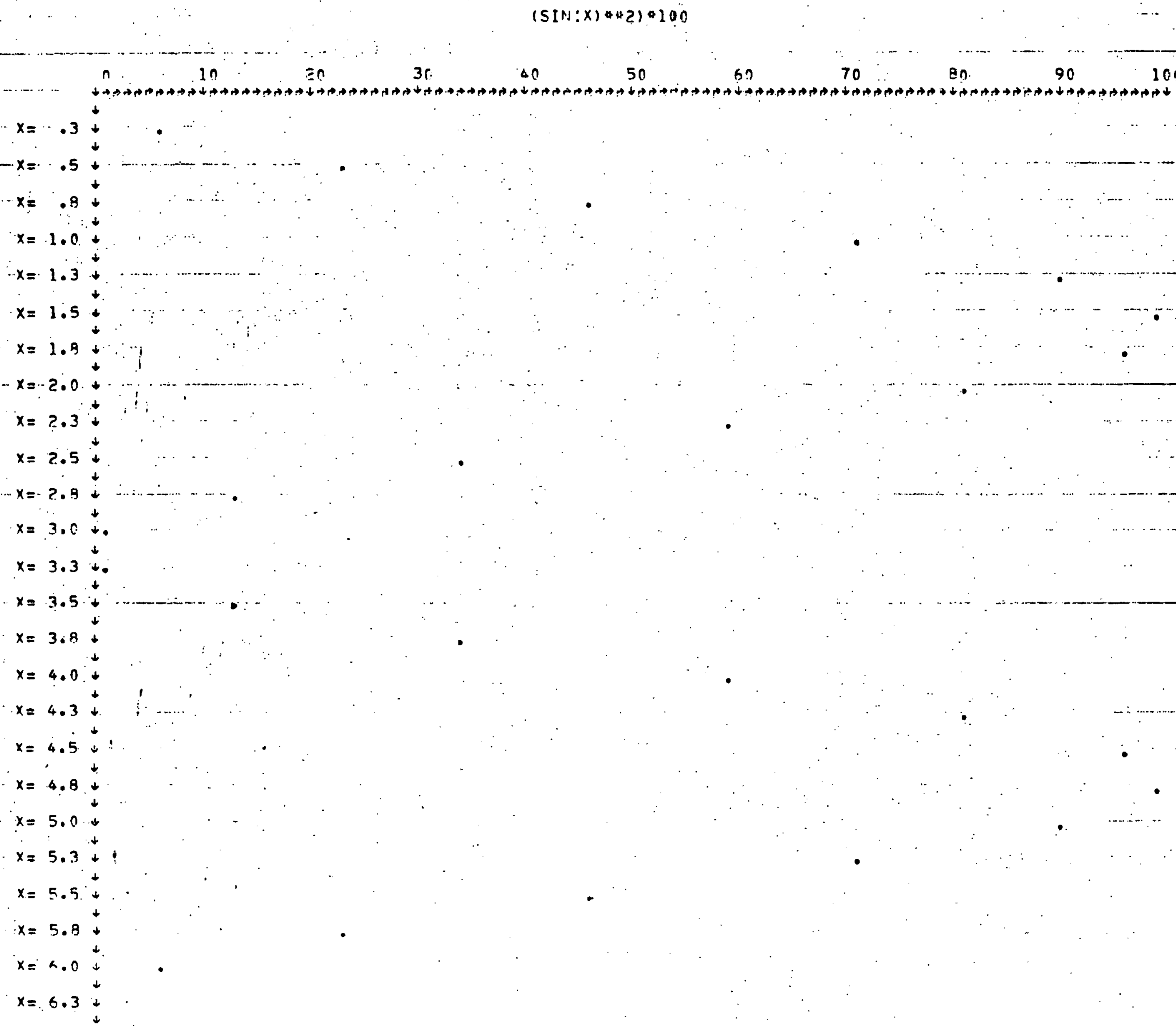
```
              SUBROUTINE PLOT
              BEGIN
000001             INTEGER J,I
000001             REAL PI,DELTAX,X
000001             PI= 3.141593
000003             CALL V LIM(1,61)
000005             CALL PAGE
000006             CALL LINES(1)
000010             CALL SPACES(50)
000012             CALL S(19H(*(SIN(X)**2)*100*))
000014             CALL LINES(3)
000016             CALL SPACES(8)
000020             CALL IFORMAT(3)
000022             DO 10 J=1,110,10
           *       BEGIN
000024                   CALL OUTI1(J-1)
000027       10          CALL SPACES(7)
           *       END
000033             CALL LINES(1)
000035             CALL SPACES(9)
000037             CALL S(5H(*↓*))
000041             DO 20 J= 1,100,1
           *       BEGIN
000043                   I=(J/10)*10
000046                   IF(I.EQ.J) CALL S(5H(*↓*))
000051                   IF(I.NE.J) CALL S(5H(*+*))
000055       20          CONTINUE
           *       END
000060             DELTAX= ((2.*PI)/50.)*2.
000062             X= DELTAX
000064             CALL LINES(1)
000065                   CALL SPACES(9)
000067             CALL S(5H(*↓*))
000071             CALL LINES(1)
000073             CALL REFORMAT(.TRUE.,3,1)
000076             DO 30 J= 1,25,1
           *       BEGIN
000100                   CALL SPACES(2)
000101                   CALL S(7H(*X= *))
000103                   CALL OUTF1(  X)
000105                   CALL SPACES(1)
000107                   CALL S(5H(*↓*))
000111                   I= (SIN(X)**2 * 1000.0)/10.0 + .00001
000120                   CALL SPACES(I-1)
000123                   IF(I.NE.0) CALL S(5H(*.*))
000126                   CALL LINES(1)
000130                   CALL SPACES(9)
000132                   CALL S(5H(*↓*))
000134                   CALL LINES(1)
000136       30          X= X+DELTAX
           *       END
000142             CALL LINES(1)
000144             CALL S(29H(* THE ABOVE IS A TEST PLOT*))
000146             CALL LINES(1)
000150             RETURN
000151        END
```

```
.0
    1.000
1.000E+00

    TRUE
      1
1.500
     FALSE
```

6

```
THIS IS OUTSTR THIS IS S  THIS IS OUTPUT
 A=  1.00 I=     2
 2          21.0      1.000 1.000E+00
     TRUE    2 2.500        TRUE
```



THE ABOVE IS A TEST PLOT

The program TEST on the next page will echo what is input as (*text*) and will stop on (*(STOP)*).

When run as

```
LODE(I= LGO, L= RLIB)

XEQ(TEST, TAPETTY, TAPETTY)
```

it will talk with a teletype.

```
                    PROGRAM TEST(INPUT=300,OUTPUT=300,TAPE1=INPUT,TAPE2=OUTPUT)
          *         BEGIN
000002                    INTEGER STR(10)
000002                    INTEGER I,J,CONTROL,READI,EQUIV
000002                    CALL INUNIT(1)
000004                    CALL OUTUNIT(2)
000006                    CALL S(19H(* START PROGRAM,*))
000010                    CALL S(25H(* TO STOP ENTER (STOP)*))
000012                    CALL NLCR
000013        1           CALL INSTR(1,STR)
000015                    CALL SPACES(7)
000017                    CALL S(8H(*//  *))
000021                    CALL OUTSTR(2,STR)
000023                    CALL NLCR
000024                    IF(EQUIV(STR).NE.6H(STOP)) GOTO 1
000030                    CALL S(19H(* END OF PROGRAM*))
000031                   CALL NLCR
000032                    STOP
000034              END


PROGRAM LENGTH INCLUDING I/O BUFFERS
000756

FUNCTION ASSIGNMENTS

STATEMENT ASSIGNMENTS
1       -   000014

BLOCK NAMES AND LENGTHS

VARIABLE ASSIGNMENTS
CONTROL-   000111    EQUIV   -   000113    I      -   000107    J      -   000110

START OF CONSTANTS-000037         TEMPS--000071     INDIRECTS-000075

ROUTINE COMPILES IN 041000
```

References

1. Kauth, D.E., et al, A Proposal for Input-Output Conventions in ALGOL 60. Comm. ACM 1(1964) 273-283.

2. 3000/6000 ALGOL Generic Reference Manual, Pub. No. 60214900, Control Data Corp., 3145 Porter Drive, Palo Alto, Calif.

3. BC ALGOL Manual. University of California, Computer Center, Berkeley, Oct. 1966 (third ed).

4. DeVogelaere, R., Algorithm 335, A Set of Basic Input-Output Procedures, Comm. ACM 11 (Aug., 1968), 567-573.

5. Naur, P. (Ed.), Revised Report on the Algorithmic Language ALGOL 60. Comm. ACM 5, 1 (Jan. 1963), 1.

6. 6400/6500/6600 Computer Systems FORTRAN Reference Manual. Pub. No. 60174900B, Rev., Nov. 1967, Control Data Corp., 3145 Porter Drive, Palo Alto, Calif.

7. 6400/6500/6600 Computer System SCOPE Reference Manual, Pub. No. 60189400, April, 1967, Control Data Corp., 3145 Porter Drive, Palo Alto, Calif.