INPUT-OUTPUT FACILITIES

a part of

EXTENDED ALGOL

for the

BURROUGHS B 5000

INPUT-OUTPUT FACILITIES

a part of

EXTENDED ALGOL

for the

BURROUGHS B 5000

Automatic Programing
BURROUGHS CORPORATION
460 Sierra Madre Villa
Pasadena, California

December, 1961

TABLE OF CONTENTS

PREFACE

One of the programing languages utilized by the Burroughs B 5000 Information Processing System is ALGOL 60. This algorithmic language was designed to describe computational processes, and is excellent for this purpose. The formulation of this language was restricted to areas which are machine independent. Implementation of machine-dependent elements was recognized to be the responsibility of the individual computer manufacturer. For example, ALGOL 60 alone is incomplete when a computer is to be used for the execution of computational processes, since the means of communicating data to and from a particular computer are not provided.

ALGOL 60, together with these Burroughs extensions, henceforth will be referred to in Burroughs literature as Extended ALGOL. Extended ALGOL provides the B 5000 programer with complete input-output facilities; STREAM PROCEDURE declarations which allow use of the B 5000 character mode functions; the ability to perform symbolic debugging; plus other useful miscellaneous facilities, including the ability to perform partial-word arithmetic and double-precision arithmetic.

This advance release completely describes the input-output portion of Extended ALGOL. The material presented herein will be included in forthcoming documents covering Extended ALGOL in its entirety.

This language has been patterned after familiar programing concepts and fitted into the structure of ALGOL 60. It is assumed that the reader is familiar with ALGOL 60 and the B 5000 Information Processing System. Background reading should include the following Burroughs material: An Introduction to ALGOL 60 (Bulletin 5000-21001-P); Master Control Program Characteristics for the Burroughs B 5000 Information Processing System (Bulletin 5000-21003-P); The Descriptor, a Definition of the B 5000 Information Processing System (Bulletin 5000-20002-P); and File Control on the Burroughs B 5000. In addition, the reader should be familiar with "Report on the Algorithmic Language ALGOL 60," Communications of the Association for Computing Machinery, Vol. 3, No. 5, May 1960.

SECTION ONE

INPUT

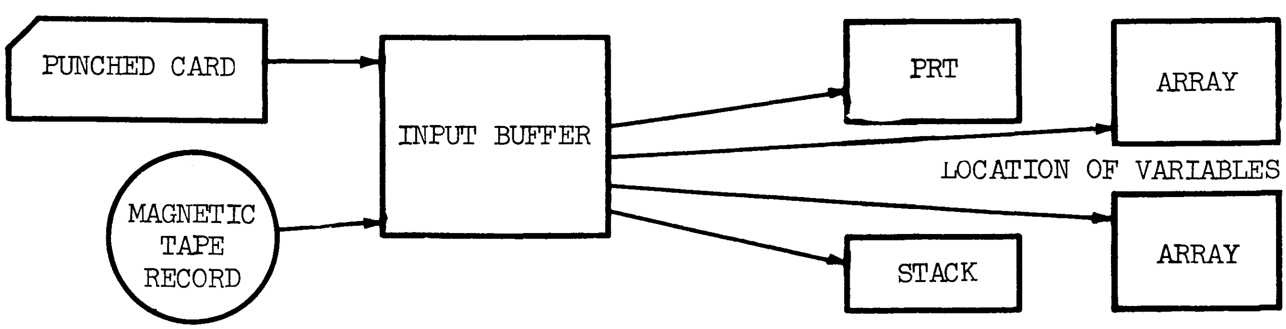
Input to a program is the means by which variables of a program are given initial values. This method is used for initializing variables, the beginning values of which vary from run to run. For variables with fixed initial values, the ⟨assignment statement⟩[1] may be used.

Input may be visualized as a communication from an external source to the program. The B 5000 provides two types of equipment through which this communication can be effected: card readers and magnetic tape units. Information may be recorded on punched cards and magnetic tape in either of two codes: alphanumeric or binary.

The contents of the communication (data) may be expressed in a variety of forms which are made available to accommodate the various kinds of data introduced to programs: alphabetic characters, integer numbers, decimal fractions, logical values, and numbers with exponents.

Any number of variables may be initialized with a single input of data. In addition, the programer has complete freedom to designate the location of each value on the input document.

The purpose of the input language portion of Extended ALGOL is to specify a fixed set of the above alternatives for every communication.



[1] The use of broken brackets ⟨ ⟩ is intended to direct the reader to the syntax of ALGOL 60 and of these extensions, for a statement of the specific form of the enclosed entity.

The need for program input is communicated to the B 5000 by means of a ⟨file declaration⟩. At such a point in a program, input buffer areas are established in memory and filled with data. Each buffer accommodates the unedited contents of one unit of input, that is, a punched card or a magnetic tape record.

When input is actually called for by means of a read statement, data is edited and selectively stored in memory in the locations previously reserved for those variables being initialized. As soon as the reading process is completed, two things occur: the buffer is refilled, and at the same time the program starts using the data which has just been provided it. More than one buffer may be used by indicating this need in the ⟨file declaration⟩.

Designation of Input Equipment Type

Whether data is to be read from punched cards or magnetic tape, and in which mode, is designated in the program parameter card. This card informs the Master Control Program (MCP) that a program is ready for processing and gives enough information about the program to enable the MCP to schedule it.

The contents of the program parameter card are not part of a program; therefore, they are not expressed in Extended ALGOL. This information is rather a message from the human operator of the system to the MCP. This card contains a complete list of ⟨file identifier⟩s which appear in the program and the type of component to be used by each.

As a consequence, the ⟨file identifier⟩s remain constant in the program. The associated components, on the other hand, may change from run to run, since their designation is from outside the program. Therefore, initial values may be obtained from punched cards for one run and from magnetic tape in the next run, without any change to the program.

The Programing Language

INPUT FILE DECLARATION

Syntax:

⟨file declaration⟩ ::= FILE ⟨input or output⟩ ⟨file part⟩
⟨input or output⟩ ::= IN | OUT
⟨file part⟩ ::= ⟨file identifier⟩ (⟨buffer part⟩) | ⟨file part⟩ ,
    ⟨file identifier⟩ (⟨buffer part⟩)
⟨buffer part⟩ ::= ⟨number of buffers⟩ , ⟨buffer size⟩
⟨number of buffers⟩ ::= ⟨unsigned integer⟩
⟨buffer size⟩ ::= ⟨unsigned integer⟩
⟨file identifier⟩ ::= ⟨identifier⟩

Examples:

FILE IN DATA1 (2,10), DATA2 (3,20)
FILE IN IN1 (1,1023)

Semantics:

The purpose of the input ⟨file declaration⟩ is to describe the buffers needed for handling the input of the file. The number of buffers desired and the size of each are called for in the file declaration. It, like all other ⟨declaration⟩s, must appear in some ⟨block head⟩ in the program.

An input ⟨file declaration⟩ results in the establishment of the designated number of buffer areas, each with the number of words indicated by ⟨buffer size⟩. If sufficient memory is not available to assign that ⟨number of buffers⟩, a lesser number is allocated by the MCP at run time.

In addition, the buffers are filled with data: the contents of one punched card or magnetic tape record per buffer. The size of the buffer area must be large enough to accommodate the entire contents of the particular unit of input being used. For instance, an 80-column card read in alpha mode requires a ⟨buffer size⟩

of 10. The same size card, read in binary mode, requires 20 words. A magnetic tape record may vary in size from one to 1023 words.

The buffer areas are retained in memory until an exit is made from the block in which the ⟨file declaration⟩ appeared.

INPUT FORMAT DECLARATION

Syntax:

```
⟨format declaration⟩ ::= FORMAT ⟨input or output⟩ ⟨format part⟩
⟨input or output⟩ ::= IN | OUT
⟨format part⟩ ::= ⟨format identifier⟩ ( ⟨editing specifications⟩ ) | ⟨format part⟩ ,
                  ⟨format identifier⟩ ( ⟨editing specifications⟩ )
⟨format identifier⟩ ::= ⟨identifier⟩
⟨editing specifications⟩ ::= ⟨editing segment⟩ | ⟨editing specifications ⟩ / |
                             / ⟨editing specifications⟩ | ⟨editing specifications⟩
                             / ⟨editing segment⟩
⟨editing segment⟩ ::= ⟨editing phrase⟩ | ⟨repeat part⟩ [ ⟨editing specifications⟩ ] |
                      ⟨editing segment⟩ , ⟨editing phrase⟩ | ⟨editing segment⟩ ,
                      ⟨repeat part⟩ [ ⟨editing specifications⟩ ]
⟨editing phrase⟩ ::= ⟨repeat part⟩ ⟨editing phrase type⟩ ⟨field part⟩ | ⟨string⟩
⟨repeat part⟩ ::= ⟨empty⟩ | ⟨unsigned integer⟩
⟨editing phrase type⟩ ::= A | D | E | F | I | L | O | P | X
⟨field part⟩ ::= ⟨empty⟩ | ⟨field width⟩ | ⟨field width⟩ . ⟨decimal places⟩
⟨field width⟩ ::= ⟨unsigned integer⟩
⟨decimal places⟩ ::= ⟨unsigned integer⟩
```

Examples:

```
FORMAT IN F1 (X4,2I6,E9.2,3F5.1)
FORMAT IN F21 (2L6,I8),F22 (18 O)
FORMAT IN F31 (A5,3A6,X5,A4),F32 (13A6,A2)
FORMAT IN F4 (A6,5[X3,2E9.2,2F6.1],3I7)
FORMAT IN F5 (8E10.3 / 16L5)
```

Semantics:

The input ⟨format declaration⟩ defines the editing necessary to be performed on the data to make it acceptable to the program. The input buffer contents may be a string of 6-bit characters (alpha mode) or a string of 48-bit binary words (binary mode). It is the responsibility of the input ⟨format declaration⟩ to indicate where, and in what form, the initial values of variables are to be found in this string. The ⟨editing phrase⟩ accomplishes this task.

The syntax above shows that the ⟨editing phrase⟩ may be in either of two forms. In the first form, the ⟨repeat part⟩ of the ⟨editing phrase⟩ is an integer which indicates the number of times an ⟨editing phrase⟩ is to be used. If the ⟨repeat part⟩ is ⟨empty⟩, it is taken to be equal to one. Its purpose is to eliminate the need for consecutively duplicating the same phrase. A series of ⟨editing phrase⟩s may also be designated for repetitive use by enclosing the set in square brackets. The number of uses is denoted by an integer immediately preceding the left bracket. Each use of an ⟨editing phrase⟩ of this form, except those which delete, accomplishes the initializing of one computer word.

The heart of the ⟨editing phrase⟩ is the ⟨editing phrase type⟩. There are eight different input types, which are grouped into two categories corresponding to the two basic representations of data, alphanumeric and binary. It is important to recognize the difference between the form of the input data upon entry into the buffer area and the essential nature of the data; that is, the form in which it is used by the program, since they do not necessarily correspond. A numeric value may be in decimal form upon read in and be converted to binary form for use by the program.

The ⟨field part⟩ of the ⟨editing phrase⟩ indicates the number of characters to be effected by that phrase. It may, also, in the case of numbers, indicate the presence of a decimal point and the number of digits after that decimal point. It serves no function in binary-type ⟨editing phrase⟩s, since each such phrase always refers to one word.

The second form, ⟨string⟩, is used for output only. It is not allowed in an input ⟨format declaration⟩.

ALPHANUMERIC ⟨EDITING PHRASE TYPE⟩S

The ⟨editing phrase type⟩s A,E,F,I,L and X are alphanumeric types. They are used for editing data which is in the alphanumeric form upon entry. Such data will be interpreted as being composed of 6-bit characters. Several ways are provided for expressing data in this form, which are syntactically defined as follows.

Syntax:

⟨character input data⟩ ::= ⟨string input⟩ | ⟨logical input⟩ | ⟨numeric input⟩ | ⟨character input data⟩ ⟨string input⟩ | ⟨character input data⟩ ⟨logical input⟩ | ⟨character input data⟩ ⟨numeric input⟩
⟨string input⟩ ::= ⟨any sequence of characters⟩
⟨logical input⟩ ::= TRUE | FALSE | ⟨space⟩ ⟨logical input⟩
⟨space⟩ ::= ⟨single space⟩ | ⟨space⟩ ⟨single space⟩
⟨numeric input⟩ ::= ⟨sign⟩ ⟨unsigned numeric input⟩ | ⟨space⟩ ⟨numeric input⟩
⟨sign⟩ ::= + | - | ⟨single space⟩
⟨unsigned numeric input⟩ ::= ⟨decimal number⟩ | 0 ⟨decimal fraction⟩ ⟨power of ten⟩
⟨exponent⟩ ::= ⟨digit⟩ ⟨digit⟩
⟨decimal number⟩ ::= ⟨unsigned integer⟩ | ⟨unsigned integer⟩ ⟨decimal fraction⟩ | ⟨decimal fraction⟩
⟨power of ten⟩ ::= , ⟨sign⟩ ⟨exponent⟩
⟨decimal fraction⟩ ::= . ⟨unsigned integer⟩
⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨unsigned integer⟩ ⟨digit⟩

The above syntax is not a part of Extended ALGOL, but is only a description of all forms of input acceptable to a program written in Extended ALGOL using the ⟨format declaration⟩.

Examples of Character Input Data

⟨string input⟩	⟨logical input⟩
ALGOL60	bTRUE
=≥+A3"	FALSE

⟨numeric input⟩

+5000	b4.625	b0.74,-01[2]
b961	+.125	b+0.18,b03
-237	-167.7	-0.1,+08

Note that in the above examples the numbers are separated into three groups: integers, numbers with decimal points, and numbers with exponents. The ⟨editing phrase type⟩s used with these numbers are I, F, and E, respectively. Type A is used for ⟨string input⟩ and type L for ⟨logical input⟩. Type X is used to delete characters from the input data. The ⟨field part⟩ indicates the number of characters to be deleted. The effect of each type is shown in the following illustration.

Assume that the input data, shown in the above examples, is read in from a card. The contents of this card are as follows:

```
Col.
 1   5   10   15   20   25   30   35   40   45   50   55   60   65   70   75   80
 ALGOL60=≥+A3"bTRUEFALSE+5000b961-237b4.625+.125-167.7b0.74,-01b+0.18,b03-0.1,+08
```

The input buffer would appear as follows:

A	L	G	O	L	6	0	=
≥	+	A	3	"	b	T	R
U	E	F	A	L	S	E	+
5	0	0	0	b	9	6	1
-	2	3	7	b	4	.	6
2	5	+	.	1	2	5	-
1	6	7	.	7	b	0	.
7	4	,	-	0	1	b	+
0	.	1	8	,	b	0	3
-	0	.	1	,	+	0	8

To illustrate how the ⟨editing phrase type⟩s function, the following ⟨format declaration⟩ will be applied to the above input data.

```
FORMAT IN F1 (A5,X2,A6,2L5,I5,2I4,F6.3,F5.3,F6.1,X9,E10.2,E8.1)
```

[2]The sign of an exponent must be indicated by either a minus (-) or a plus (+) sign, or a blank. A blank is taken to be a plus (+).

CHARACTERS TO WHICH EDITING PHRASE REFERS	EDITING PHRASE		FORM IN WHICH INITIAL VALUE IS SUPPLIED TO THE PROGRAM[3]	
ALGOL	A5		0 0 0 A L G O L	
60	X2		No Characters Supplied	
=≥+A3"	A6		0 0 = ≥ + A 3 "	
bTRUE	2L5	L5	F++00	0000000000001
FALSE		L5	F++00	0000000000000
+5000	I5		F++00	0000000011610
b961	2I4	I4	F++00	0000000001701
-237		I4	F-+00	0000000000355
b4.625	F6.3		F+-14	4500000000000
+.125	F5.3		F+-15	1000000000000
-167.7	F6.1		F--12	2475463146314
b0.74,-01	X9		No Characters Supplied	
b+0.18,b03	E10.2		F+-12	2640000000000
-0.1,+08	E8.1		F--05	4611320000000

BINARY ⟨EDITING PHRASE TYPE⟩S

The binary ⟨editing phrase type⟩s are D and O. They are used for editing data which is in binary form upon entry, that is, in the form to be used by the program; therefore, no conversion is necessary. Such input generally would be on cards punched with straight binary code or on magnetic tape recorded in the binary mode. The ⟨field part⟩ in these cases is irrelevant and should be ⟨empty⟩.

The ⟨editing specifications⟩ of an input ⟨format declaration⟩ must not contain a mixture of ⟨editing phrase type⟩s; that is, they must be either all alphanumeric or all binary.

[3] See Appendix A for explanation of constructs of B 5000 computer words.

The table below summarizes the actions of the ⟨editing phrase type⟩s.

ALPHANUMERIC

TYPE	WHEN USED	FORM OF INPUT STRING	EDITED FORM FOR PROGRAM USE
A	when data is to remain in alpha form	one field of characters	one alpha word[4]
E	when data is numeric in nature but externally represented in alpha as a decimal fraction with an exponent	one field of characters	one binary word[5]
F	when data is numeric in nature but externally represented in alpha as a decimal number without an exponent	one field of characters	one binary word[5]
I	when data is numeric in nature but externally represented in alpha as an integer	one field of characters	one binary word[5]
L	when data is logical in nature and externally represented in alpha as a ⟨logical value⟩	one field of characters	one binary word
X	when data in alpha form is to be deleted	one field of characters	nothing

BINARY

TYPE	WHEN USED	FORM OF INPUT STRING	EDITED FORM FOR PROGRAM USE
O	when data is externally represented in binary	one binary word	one binary word
D	when data in binary form is to be deleted	one binary word	nothing

[4]These alpha words contain not more than six characters.

[5]Decimal integers and decimal mantissas have a maximum value of 549755813887.

INPUT LIST DECLARATION

Syntax:

⟨list declaration⟩ ::= LIST ⟨list part⟩

⟨list part⟩ ::= ⟨list identifier⟩ (⟨list⟩) | ⟨list part⟩ , ⟨list identifier⟩ (⟨list⟩)

⟨list identifier⟩ ::= ⟨identifier⟩

⟨list⟩ ::= ⟨list segment⟩ | ⟨list⟩ , ⟨list segment⟩

⟨list segment⟩ ::= ⟨expression part⟩ | ⟨for clause⟩ ⟨list segment⟩ | ⟨for clause⟩ [⟨expression list⟩]

⟨expression part⟩ ::= ⟨list identifier⟩ | ⟨arithmetic expression⟩ | ⟨Boolean expression⟩

⟨expression list⟩ ::= ⟨expression list⟩ , ⟨expression part⟩ | ⟨expression part⟩

Examples:

```
LIST L1 (X,Y,Z,PQ2)
LIST L2 (X[I],Y,R[J,K],Z), L21 (A,B,C[I])
LIST L3 (FOR I←X STEP 1 UNTIL 5 DO B[I],T,U)
LIST L4 (FOR I←1 STEP 1 UNTIL 10 DO FOR J←1 STEP 1 UNTIL 15 DO A[I,J])
```

Semantics:

A ⟨list declaration⟩ can be used for both input and output. However, when used for input, expressions in the ⟨expression part⟩ must be ⟨variable⟩s only.

The input ⟨list declaration⟩ specifies a list of variables to be initialized and also designates the order. The input ⟨list⟩ may consist of any or all of the following constructs: variables, list identifiers, and FOR clauses.

Variables are of two types: simple and subscripted. Both are single valued. They form the basic element of a list declaration, and may be either local or non-local to the ⟨list declaration⟩ block. If local, their declaration must precede the ⟨list declaration⟩ in which they appear.

If a ⟨list declaration⟩ contains variables which have already been declared in another ⟨list declaration⟩, it is not necessary to list them again. Use of the previously declared ⟨list identifier⟩ is sufficient. Recursive use of ⟨list identifier⟩s is meaningless and not allowed.

The ⟨for clause⟩ is used to initialize arrays either in whole or in part.

READ STATEMENT

Syntax:

⟨I-O statement⟩ ::= ⟨read statement⟩ | ⟨release statement⟩ | ⟨write statement⟩
⟨read statement⟩ ::= READ (⟨input parameters⟩)
⟨input parameters⟩ ::= ⟨file identifier⟩ , ⟨format identifier⟩ , ⟨list⟩

Examples:

```
READ (FILE1, FORMAT2, LIST3)
READ (FIN1, FORM1, FOR I←0 STEP 1 UNTIL 13 DO HEAD [I])
READ (F6, FORM3, VARY5)
READ (FILL1, MAT2, U, V, W, X, VARY1)
READ (F2, F3, L1)
```

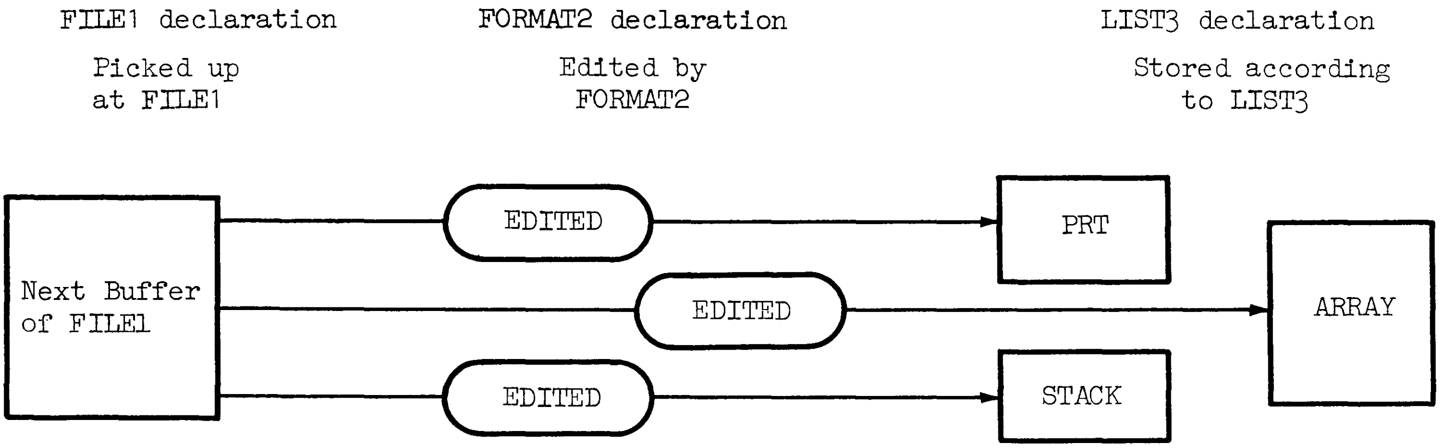
Semantics:

Thus far, three kinds of declarations have been discussed: file declaration, which creates and initially fills input buffers with data; format declaration, which describes the editing to be performed on the data found in an input buffer; and list declaration, which provides the names of variables to be given initial values.

The <read statement> calls for the actual initializing of variables by associating a set of these declarations. A list of variables can be indicated in two ways. The programer can write a <list declaration> and use the <list identifier> as an input parameter, or he can include the list of variables in the <read statement> itself.

The reading process is illustrated below:

READ (FILE1, FORMAT2, LIST3)



Since the ⟨file declaration⟩ precedes the read statement, the buffer is already filled with data before the read statement is first encountered. If more than one buffer is being utilized, they are sequenced so that a first-in, first-out operation results.

The data is selected and edited according to the ⟨format declaration⟩. The number of words represented by the sum of the ⟨field width⟩s in the ⟨editing specifications⟩ of the ⟨format declaration⟩ is normally equal to the ⟨buffer size⟩ in the ⟨file declaration⟩. In all cases, it must be equal to or less than ⟨buffer size⟩.

The data which is to be stored is then put in the locations previously assigned to the variables in the ⟨list⟩.

The reading process is terminated when all variables in the ⟨list⟩ have been initialized. More than one unit of input, card or magnetic tape record, may be used with a single read statement.

An additional unit of input is called for each time one of the following occurs before the ⟨list⟩ is exhausted:

1. A slash is encountered in the format declaration.
2. The end of the format declaration is reached.

The slash is used when formats differ from card to card or from tape record to tape record. After a slash has caused an additional unit of input to be read, subsequent data is formulated according to the ⟨editing segment⟩ to the right of the slash. Two adjacent slashes cause one unit of input to be ignored. One slash at the beginning or end of a format declaration has the same effect.

When all editing phrases have been used before the ⟨list⟩ is exhausted, editing of the next unit of input proceeds from the beginning of the format declaration.

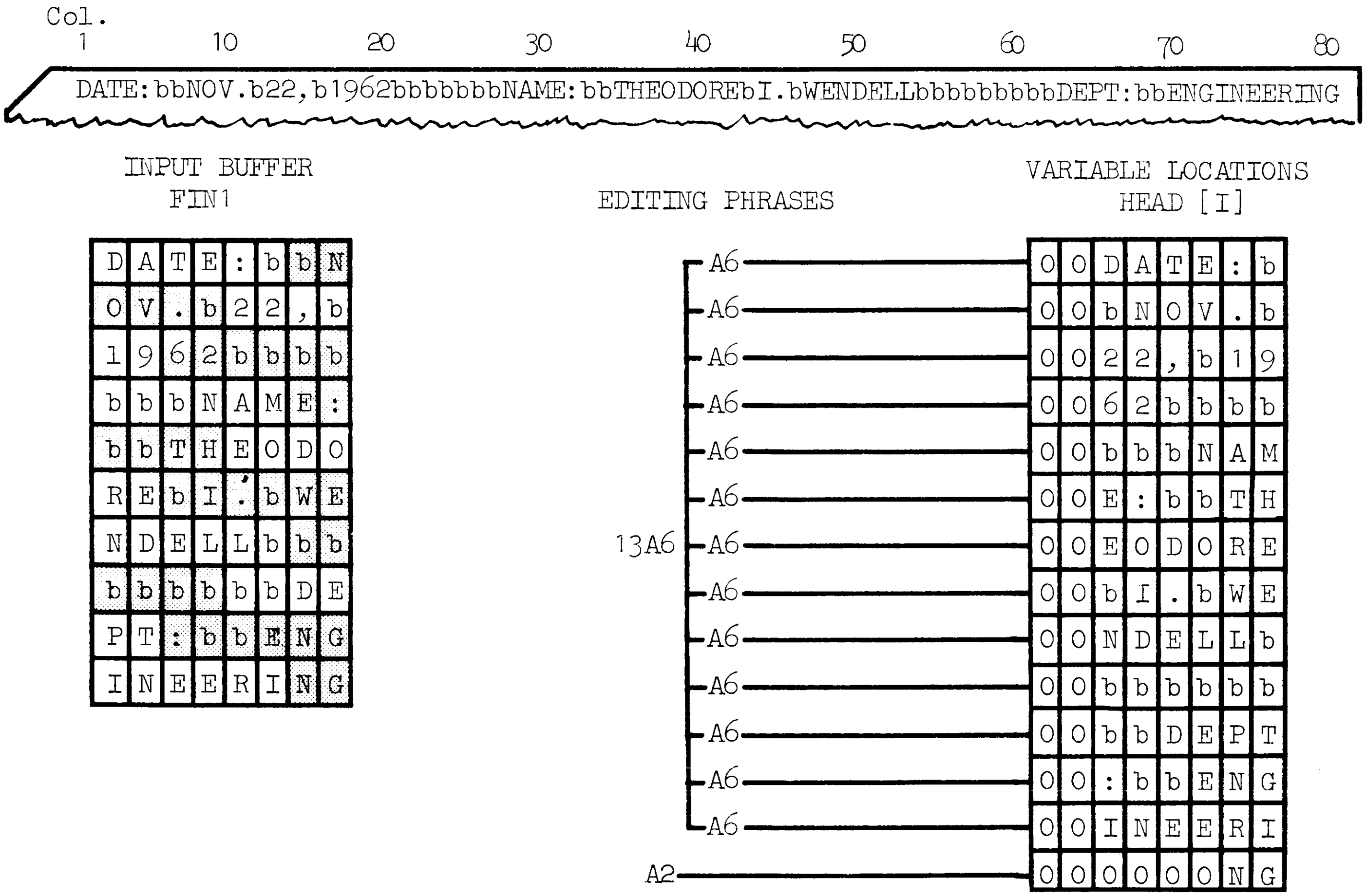
After reading is completed, the buffer(s) used in the process is automatically refilled, and all buffers are properly sequenced in anticipation of the next statement involving this file.

PROBLEM 1. To demonstrate the use of the type-A editing phrase and the ⟨string⟩ phrase, assume the program makes use of a heading used for a printout which varies from run to run. Assume this information includes the date, the name of the person using the program, and his department number. Input is from a card punched in alpha mode and the date is in columns 1-20, the name is in columns 28-55, and the department number is in columns 63-80.

Selected portions of the program might appear as follows:

```
BEGIN
   ARRAY HEAD [0:13]; INTEGER I;
   FILE IN FIN1 (1,10);
   FORMAT IN FORM1 (13A6, A2);
   READ (FIN1, FORM1, FOR I←0 STEP 1 UNTIL 13 DO HEAD[I])
```

The punched card from which the buffer is filled reads as follows:



PROBLEM 2: To illustrate the input of numeric and logical initial values, consider the following program elements. Assume that input is from an alpha card read.

Note that the contents of two buffers are used to initiate variables with one ⟨read statement⟩. As a consequence, the editing specifications are used twice. The sum of the field widths in FORM3 is equal to 80, which is equivalent to 10 words, the buffer size of F6.

```
BEGIN
   BOOLEAN ARRAY B [0:4];
   BOOLEAN A1, A2, A3, A4, A5;
   REAL X, Y, Z, X1, X2, X3;
   INTEGER I, J, K, L, M;
   FILE IN F6 (2, 10);
   FORMAT IN FORM3 (2E10.3, F8.3, I6, I13, X8, 5L5);
   LIST VARY5 (X,Y,Z,I,J,FOR M←0 STEP 1 UNTIL 4 DO B[M],X1,X2,X3,K,L,
     A1,A2,A3,A4,A5);
   READ (F6,FORM3,VARY5)
```

The cards from which the buffers are filled are as follows:

```
Col.
 1       10        20        30        40        50        60        70        80
 b0.645,+02+0.165,b02+735.125-13892bbbbbb-136421+137.931FALSEbTRUEbTRUEbTRUEFALSE
Col.
 1       10        20        30        40        50        60        70        80
 -0.173,+07-1.000,+07bb16.250b+1416b549755813887bb-3.692bTRUEFALSEFALSEbTRUEFALSE
```

INPUT BUFFERS
F6

b	0	.	6	4	5	,	+
0	2	+	0	.	1	6	5
,	b	0	2	+	7	3	5
.	1	2	5	-	1	3	8
9	2	b	b	b	b	b	b
-	1	3	6	4	2	1	+
1	3	7	.	9	3	1	F
A	L	S	E	b	T	R	U
E	b	T	R	U	E	b	T
R	U	E	F	A	L	S	E

-	0	.	1	7	3	,	+
0	7	-	1	.	0	0	0
,	+	0	7	b	b	1	6
.	2	5	0	b	+	1	4
1	6	b	5	4	9	7	5
5	8	1	3	8	8	7	b
b	-	3	.	6	9	2	b
T	R	U	E	F	A	L	S
E	F	A	L	S	E	b	T
R	U	E	F	A	L	S	E

EDITING PHRASES FORM3		VARIABLE LOCATIONS VARY5		
2E10.3	E10.3	F+-12	1004000000000	X
	E10.3	F++00	0000000040164	Y
F8.3		F+-11	1337100000000	Z
I6		F-+00	0000000033104	I
I13		F--00	0000000412345	J
X8		none		
5L5	L5	F++00	0000000000000	B[0]
	L5	F++00	0000000000001	B[1]
	L5	F++00	0000000000001	B[2]
	L5	F++00	0000000000001	B[3]
	L5	F++00	0000000000000	B[4]
2E10.3	E10.3	F--00	0000006462720	X1
	E10.3	F--00	0000046113200	X2
F8.3		F+-13	2020000000000	X3
I6		F++00	0000000002610	K
I13		F++00	7777777777777	L
X8		none		
5L5	L5	F++00	0000000000001	A1
	L5	F++00	0000000000000	A2
	L5	F++00	0000000000000	A3
	L5	F++00	0000000000001	A4
	L5	F++00	0000000000000	A5

PROBLEM 3. This problem illustrates the use of the binary-type editing phrases. Normally, input is from magnetic tape which has been produced by another program and recorded in the binary mode.

The relevant ALGOL construct might be as follows.

```
BEGIN
    BOOLEAN Y1, Y2, Z1, Z2;
    REAL T, U, V, W, X;
    INTEGER M, N, O;
    FILE IN FILL1 (1,11);
    FORMAT IN MAT2 (80, 2X);
    LIST VARY1 (Y1,Y2,Z1,Z2);

    READ (FILL1, MAT2, U,V,W,X,VARY1)
```

INPUT BUFFER FILL1		EDITING PHRASES MAT2	VARIABLE LOCATIONS VARY1		
F+-16	5321401000000	0	F+-16	5321401000000	U
F+-16	7342100521306	0	F+-16	7342100521306	V
F--07	1056733520000	0	F--07	1056733520000	W
F--12	5632100000000	0	F--12	5632100000000	X
F++00	0000000000000	0	F++00	0000000000000	Y1
F++00	0000000000000	0	F++00	0000000000000	Y2
F++00	0000000000001	0	F++00	0000000000001	Z1
F++00	0000000000000	0	F++00	0000000000000	Z2
F+-13	1133200000000	X	none		
F++00	0000000001024	X	none		

PROBLEM 4. To illustrate the use of a ⟨for clause⟩ in a ⟨list declaration⟩, consider the following problem:

A 2-dimensional array A exists, and it is now desired to establish new values for two of its rows. These values are read in from an alpha punched card.

```
BEGIN
   ARRAY A [0:4, 0:2];
   FILE IN F2 (1,10);
   FORMAT IN F3 (6E13.6);
   LIST L1 (FOR I←2 STEP 1 UNTIL 3 DO FOR J←0 STEP 1 UNTIL 2 DO A[I,J]);

   READ (F2, F3, L1)
```

The card from which the buffer is filled reads as follows:

```
Col.
 1       10      20      30      40      50      60      70      80
 b0.645000,+02+0.165000,b05-0.173000,+07+0.735125,b03-0.136421,+06+0.162500,+02bb
```

INPUT BUFFER F2

b	0	.	6	4	5	0	0
0	,	+	0	2	+	0	.
1	6	5	0	0	0	,	b
0	5	-	0	.	1	7	3
0	0	0	,	+	0	7	+
0	.	7	3	5	1	2	5
,	b	0	3	-	0	.	1
3	6	4	2	1	,	+	0
6	+	0	.	1	6	2	5
0	0	,	+	0	2	b	b

EDITING PHRASES F3		VARIABLE LOCATIONS		
6.E13.6	E13.6	F+-12	1004000000000	A[2,0]
	E13.6	F+-10	4016400000000	A[2,1]
	E13.6	F--06	6462720000000	A[2,2]
	E13.6	F+-11	1337100000000	A[3,0]
	E13.6	F--07	4123450000000	A[3,1]
	E13.6	F+-13	2020000000000	A[3,2]

SUMMARY OF READ STATEMENT. The ⟨read statement⟩, together with its associated file, format, and list, has been designed to take care of input conditions which occur in scientific problems.

The programer, however, is not required to use the ⟨read statement⟩. He has the option of filling the input buffer or buffers by using the ⟨file declaration⟩, and then operating upon the buffer contents with a STREAM PROCEDURE in any way desired. In order to do this, the file identifier is passed as an actual parameter to a STREAM PROCEDURE.

When the reading process connected with a ⟨read statement⟩ is completed, the affected buffer contents are automatically destroyed by the input of more data. A statement to cause the refilling of input buffers is needed, therefore, when the ⟨read statement⟩ is not used. The ⟨release statement⟩ serves this purpose.

INPUT RELEASE STATEMENT

Syntax:

⟨release statement⟩ ::= RELEASE (⟨file identifier⟩)
⟨stream release statement⟩ ::= RELEASE (⟨formal parameter⟩)

Examples:

```
RELEASE (FILE3)
RELEASE (FILL5)
RELEASE (F)
```

Semantics:

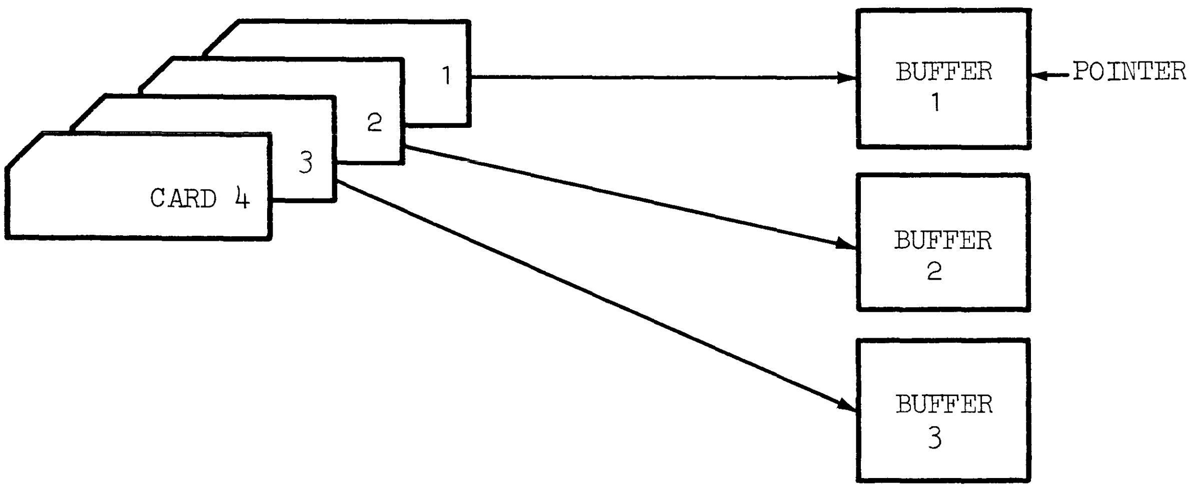
The input ⟨release statement⟩ causes one input buffer of the file indicated to be filled with new data. If more than one buffer is being used, a reordering occurs which maintains the first-in, first-out operation.

The release statement is the only part of Extended Algol dealing with input-output which may be included in a STREAM PROCEDURE. When so used it is metalinguistically referred to as a ⟨stream release statement⟩. It looks the same wherever it is used. The difference in syntax is to point up the fact that in a STREAM PROCEDURE a formal parameter must be used to indicate the file rather than the file identifier itself.

PROBLEM 1. To illustrate the buffer-ordering procedure, assume an input file which uses three buffers. Only one name is used for all three buffers; therefore, some means is required to distinguish between them. The means employed can be visualized as a pointer, which at any one time is directed toward the buffer which will be used if the file is called for.

```
BEGIN
   FILE IN FILE3 (3,10);
```

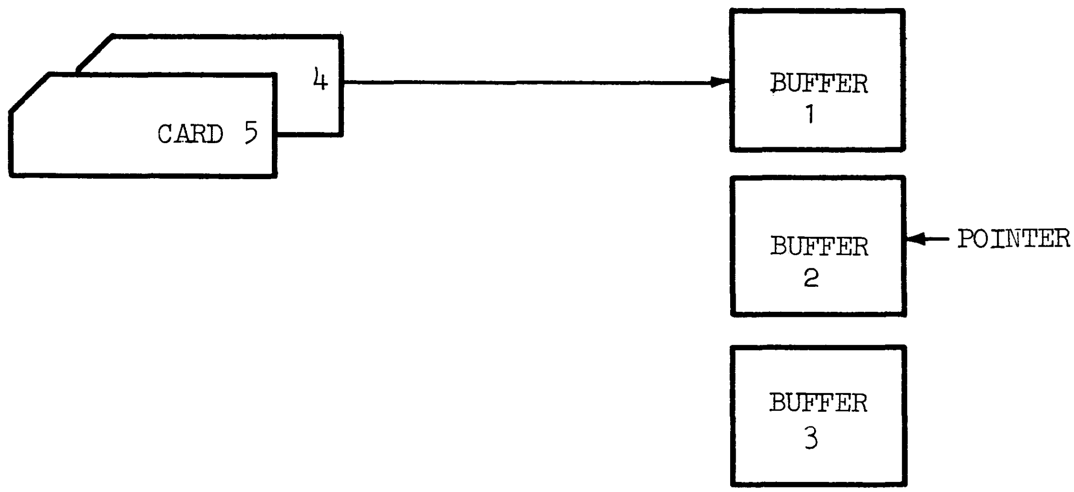
This would result in three cards being read into the buffers. The pointer is at buffer 1 as soon as that buffer is filled.




```
   STREAM PROCEDURE EDIT (A,B,C);
      BEGIN
      - - - - RELEASE(A);- - - -
      END;
   - - - - - - - - - - - - - - - - - - - - - - -
   - - - - - - - - - - - - - - - - - - - - - - -
   RELEASE (FILE3); EDIT (FILE3,KEY,TABLE);
END
```

Any use of the file identifier (FILE3) at this point refers to buffer 1. Either the ⟨release statement⟩ or the ⟨stream release statement⟩ above would result in the pointer being shifted to buffer 2. Then buffer 1 would be refilled with the contents of card 4.

Since a read statement has an implied release at the end of its operation, the effect on the buffers is the same.



SECTION TWO

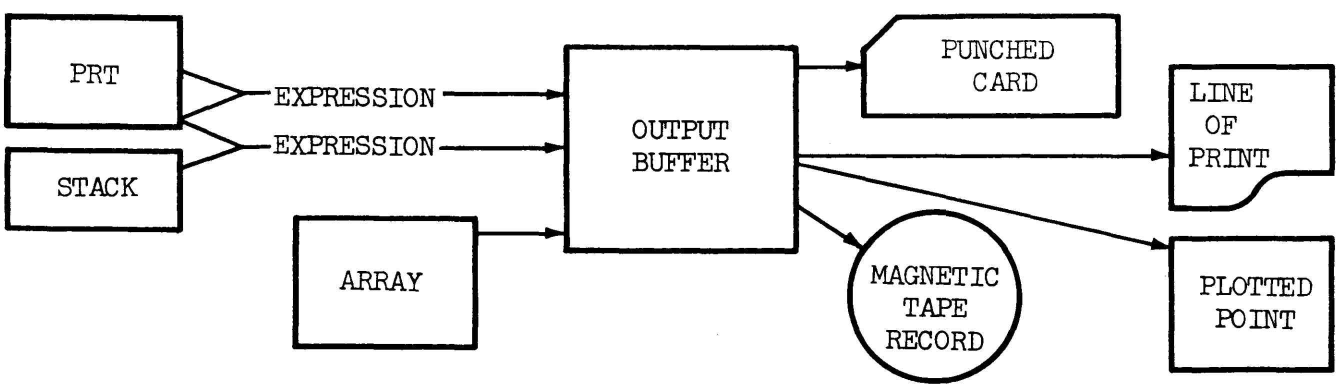
OUTPUT

Output is the means by which the program communicates the results it has obtained to the programer. The B 5000 provides several types of equipment through which output can be recorded: line printer, card punch, magnetic tape, and plotter. The storage drum and message printer are reserved for use by the MCP and its associated compilers.

Due to the variety of possible output communications, several formats are available to the programer. Numeric values, for instance, may be expressed as integers, decimal numbers, or decimal fractions with an associated power of ten.

The information to be externally recorded normally involves the values of certain variables in the program.

The purpose of the output language portion of Extended ALGOL is to enable every communication to specifically designate the output equipment, the desired format, and the expressions to be evaluated.



THE OUTPUT PROCESS

The fact that a program involves output is indicated by means of an output ⟨file declaration⟩. The output ⟨file declaration⟩ results in the establishment of output buffers in memory.

Actual output is called for by a ⟨write statement⟩. This statement causes the expressions in the ⟨list declaration⟩ to be evaluated and stored into the output buffer in the form indicated by the associated ⟨format declaration⟩.

When a buffer has been filled,the data is transmitted to the specified output equipment.

Designation of Output Equipment Type

Whether results are to be punched, plotted, printed, or recorded on magnetic tape is specified by the program parameter card. This card informs the MCP that a program is ready for processing, and provides sufficient information to enable the MCP to schedule it.

The contents of the program parameter card are not a part of the program; therefore they are not expressed in Extended ALGOL. This information is a message from the operator of the system to the MCP. The message contains a complete list of ⟨file identifier⟩s which appear in the program and the type of output equipment to be used by each.

As a consequence, it is not necessary to alter the program, even though the peripheral equipment may change from run to run. Therefore results may be printed during one run and recorded on magnetic tape during the next run, using the same program.

OUTPUT FILE DECLARATION

Syntax:

```
⟨file declaration⟩ ::= FILE ⟨input or output⟩ ⟨file part⟩
⟨input or output⟩ ::= IN | OUT
⟨file part⟩ ::= ⟨file identifier⟩ ( ⟨buffer part⟩ ) | ⟨file part⟩ ,
                ⟨file identifier⟩ ( ⟨buffer part⟩ )
⟨buffer part⟩ ::= ⟨number of buffers⟩ , ⟨buffer size⟩
⟨number of buffers⟩ ::= ⟨unsigned integer⟩
⟨buffer size⟩ ::= ⟨unsigned integer⟩
⟨file identifier⟩ ::= ⟨identifier⟩
```

Examples:

```
FILE OUT RESULTS (2,115), ANS (1,1)
FILE OUT PRINT (2,15)
```

Semantics:

The only difference in syntax between an input ⟨file declaration⟩ and an output ⟨file declaration⟩ is ⟨input or output⟩. They both result in the establishment of buffer areas in memory.

The output ⟨buffer size⟩ must be sufficiently large to contain the contents of one unit of output. These minimum sizes are as follows:

Line Printer	1 line	15 words
Card Punch	1 80-Col. Card	10 words
Magnetic Tape	1 record	May vary from 1 to 1023 words
Plotter	1 point	1 word

OUTPUT FORMAT DECLARATION

Syntax:

```
⟨format declaration⟩ ::= FORMAT ⟨input or output⟩ ⟨format part⟩
⟨input or output⟩ ::= IN | OUT
⟨format part⟩ ::= ⟨format identifier⟩ ( ⟨editing specifications⟩ ) |
                  ⟨format part⟩ , ⟨format identifier⟩ ( ⟨editing specifications⟩ )
⟨format identifier⟩ ::= ⟨identifier⟩
⟨editing specifications⟩ ::= ⟨editing segment⟩ | ⟨editing specifications⟩ / |
                             / ⟨editing specifications⟩ | ⟨editing specifications⟩ /
                             ⟨editing segment⟩
⟨editing segment⟩ ::= ⟨editing phrase⟩ | ⟨repeat part⟩ [ ⟨editing specifications⟩ ] |
                      ⟨editing segment⟩ , ⟨editing phrase⟩ | ⟨editing segment⟩ ,
                      ⟨repeat part⟩ [ ⟨editing specifications⟩ ]
⟨editing phrase⟩ ::= ⟨repeat part⟩ ⟨editing phrase type⟩ ⟨field part⟩ | ⟨string⟩
⟨repeat part⟩ ::= ⟨empty⟩ | ⟨unsigned integer⟩
⟨editing phrase type⟩ ::= A | D | E | F | I | L | O | P | X
⟨field part⟩ ::= ⟨empty⟩ | ⟨field width⟩ | ⟨field width⟩ . ⟨decimal places⟩
⟨field width⟩ ::= ⟨unsigned integer⟩
⟨decimal places⟩ ::= ⟨unsigned integer⟩
```

Examples:

```
FORMAT OUT F6 (X56, "HEADING", X57),F7 (P4.1)
FORMAT OUT F8 (X5,F5.1,2[X14,F11.3],X60),F81 (1023 O)
FORMAT OUT F9 (X4,I4,X8,F7.1,X12,F9.1,X76),F91 (8E15.4)
FORMAT OUT F10 (X6,"N",X12,"L",X19,"S",X80)
FORMAT OUT F11 (3[X6,E10.2],X72)
```

The output ⟨format declaration⟩ defines the editing necessary in order for the output to be meaningful. The output buffer may be filled with either 6-bit characters or 48-bit binary words. The function of the output ⟨format declaration⟩ is to indicate where and in what form the values of the list expressions are to be placed. The ⟨editing phrase⟩ accomplishes this task.

An output ⟨editing phrase⟩ may be in either of two forms. In the first form, the ⟨repeat part⟩ of the ⟨editing phrase⟩ is an integer which indicates the number of times an ⟨editing phrase⟩ is to be used. If the ⟨repeat part⟩ is ⟨empty⟩, it is taken to be equal to one. The purpose of the ⟨repeat part⟩ is to eliminate the need for consecutively duplicating the same phrase. A series of ⟨editing phrase⟩s may also be designated for repetitive use by enclosing the set in square brackets. The number of uses is denoted by an integer immediately preceding the left bracket.

Each use of an ⟨editing phrase⟩ of this form, except the X and D types, takes the contents of one computer word as the expression value to be edited and stored into the output string.

The controlling factor in an ⟨editing phrase⟩ is the ⟨editing phrase type⟩. There are nine different output types which are grouped into three categories: one for those which produce alphanumeric output, one for those which produce binary output, and one designed for producing output for the plotter.

The ⟨field part⟩ of the ⟨editing phrase⟩ indicates the number of characters in the output buffer to be filled. It may, also, in the case of numbers, indicate the need for a decimal point and the number of digits after the decimal point. For plotter output, the ⟨field part⟩ does not control the amount of the output buffer to be filled, but indicates how the plot is to be accomplished. It serves no function in the binary-type editing phrases (O and D).

The second form of an ⟨editing phrase⟩ is a ⟨string⟩. This functions as a literal; that is, output data is supplied by the ⟨editing phrase⟩ itself and not by an expression value of the program.

ALPHANUMERIC ⟨EDITING PHRASE TYPE⟩S

The following are alphanumeric types: A,E,F,I,L and X. These types are used for editing data, so that it is in alphanumeric form for output. This form of output is used for the line printer, the card punch and magnetic tape unit (alpha mode). This data can be expressed in several ways, which are syntactically defined as follows:

⟨character output data⟩ ::= ⟨string output⟩ | ⟨numeric output⟩ | ⟨logical output⟩ |
    ⟨character output data⟩ ⟨string output⟩ |
    ⟨character output data⟩ ⟨numeric output⟩ |
    ⟨character output data⟩ ⟨logical output⟩

⟨string output⟩ ::= ⟨any sequence of characters⟩

⟨numeric output⟩ ::= ⟨single space⟩ ⟨unsigned numeric output⟩ |
    - ⟨unsigned numeric output⟩ | ⟨space⟩ ⟨numeric output⟩

⟨unsigned numeric output⟩ ::= ⟨decimal number⟩ | 0 ⟨decimal fraction⟩
    ⟨scale factor⟩

⟨scale factor⟩ ::= , + ⟨exponent⟩ | , - ⟨exponent⟩

⟨logical output⟩ ::= TRUE | FALSE | ⟨space⟩ ⟨logical output⟩

⟨decimal number⟩ ::= ⟨unsigned integer⟩ | ⟨unsigned integer⟩ ⟨decimal fraction⟩ |
    ⟨decimal fraction⟩

⟨decimal fraction⟩ ::= . ⟨unsigned integer⟩

⟨exponent⟩ ::= ⟨digit⟩ ⟨digit⟩

The above syntax is not a part of Extended ALGOL but is only a description of forms of output possible from a program written in Extended ALGOL which uses the ⟨format declaration⟩.

Examples of Character Output Data

⟨string output⟩	⟨numeric output⟩
ALGOL	-13892
Y=	bb1416
⟨logical output⟩	b735.125
bTRUE	bb-735.13
FALSE	
	b0.645,+02
	bb-0.64500,+02

In these examples the numbers are separated into three groups: integers, numbers with a decimal point, and numbers with exponents. The ⟨editing phrase type⟩s used with these numbers are respectively, I, F and E. Type A and the ⟨string⟩ editing phrase are used for ⟨string output⟩, and type L is used for ⟨logical output⟩. Type X serves to place blanks in the output string. The effect of these types is shown in the following illustration.

Assume that the following expression values are to be punched in a card. To illustrate how the ⟨editing phrase type⟩s function, the following ⟨format declaration⟩ will be applied.

```
FORMAT OUT F2 (A5,A2,2L5,2I6,X10,F8.3,F9.2,E10.3,E14.5)
```

EXPRESSION VALUES		EDITING PHRASE		CHARACTERS PRODUCED FOR OUTPUT BUFFER
0 0 0 A L G O L		A5		ALGOL
0 0 0 0 0 0 Y =		A2		Y=
F++00	0000000000001	2L5	L5	bTRUE
F++00	0000000000000		L5	FALSE
F-+00	0000000033104	2I6	I6	-13892
F++00	0000000002610		I6	bb1416
none		X10		bbbbbbbbbb
F+-11	1337100000000	F8.3		b735.125
F--11	1337100000000	F9.2		bb-735.13
F+-12	1004000000000	E10.3		b0.645,+02
F--12	1004000000000	E14.5		bb-0.64500,+02

BINARY ⟨EDITING PHRASE TYPE⟩

Types O and D are binary ⟨editing phrase type⟩s. Type O is used when output is desired which identically reflects the expression values as used in the program. No conversion takes place. Type D inserts a zero value into the output. These types are used only for recording data on magnetic tape in the binary mode. In this case, the ⟨field part⟩ is irrelevant and should be ⟨empty⟩.

PLOTTER ⟨EDITING PHRASE TYPE⟩

Type P is designed for producing output for the plotter. It differs from all other types in that a pair of expression values (instead of one) is used to develop each element of the output string. All information necessary for plotting one point is contained in one word. Therefore the P type produces one word in the following form:

O	c	s	X	X	Y	Y	Y

where:

The first character is irrelevant

The second character (c) is a control digit which determines the following:

1. whether or not to allow a grid to be printed
2. whether or not to allow paper to move before plotting
3. whether or not to print the units digit of the ordinate value.

This digit (c) is developed from the ⟨field part⟩ of the ⟨editing phrase⟩. The effect of various ⟨field part⟩ values is shown in the following table.

⟨field part⟩	ALLOW GRID TO BE PRINTED	ALLOW PAPER TO BE MOVED BEFORE PLOT	PRINT UNITS DIGIT OF ORDINATE VALUE
0	Yes	Yes	No
1	Yes	Yes	Yes
2	Yes	No	No
3	Yes	No	Yes
4	No	Yes	No
5	No	Yes	Yes
6	No	No	No
7	No	No	Yes

The third character (s) determines the plotting symbol and is obtained from the ⟨decimal places⟩ of the ⟨editing phrase⟩. The symbols available are shown below.

⟨decimal places⟩	PLOTTING SYMBOL
0	▽
1	△
2	□
3	○
4	(random symbol)

The fourth and fifth characters are two decimal digits which stipulate the X-abscissa increment. These digits are the decimal equivalent of the first expression value, which may vary from 0 to 99 decimal.

The direction of paper movement depends on the sign of this expression value. If paper movement is allowed by the control digit, forward movement takes place when the sign is positive; movement is backward when the sign is negative.

The sixth, seventh, and eighth characters are three decimal digits which stipulate the Y-ordinate value to be plotted. These digits are the decimal equivalent of the second expression value, which may range from 0 to 399 decimal.

The ⟨editing specifications⟩ of an output ⟨format declaration⟩ must not contain a mixture of ⟨editing phrase type⟩s; that is, they must be either entirely alpha-numeric, binary, or of the plotter type.

The table which follows summarizes the actions of the ⟨editing phrase type⟩s.

TYPE	WHEN USED	FORM OF EXPRESSION VALUE	EDITED FORM FOR OUTPUT
ALPHANUMERIC			
A	when expression value is alpha in form	one alpha word	one field of characters
E	when expression value is to be represented as decimal fraction with an exponent	one binary word	one field of characters
F	when expression value is to be represented as decimal number without exponent	one binary word	one field of characters
I	when expression value is to be represented as an integer	one binary word	one field of characters
L	when expression value is to be represented as a logical value	one binary word	one field of characters
X	for inserting blank characters in output string	none	one field of characters
BINARY			
O	when expression values are to be communicated in exactly the same form as they appear in the program	one binary word	one binary word
D	for inserting zero words in output string	none	one binary word
PLOTTER			
P	when expression values are to be plotted	two binary words	one alpha word

OUTPUT LIST DECLARATION

Syntax:

⟨list declaration⟩ ::= LIST ⟨list part⟩

⟨list part⟩ ::= ⟨list identifier⟩ (⟨list⟩) | ⟨list part⟩ , ⟨list identifier⟩ (⟨list⟩)

⟨list identifier⟩ ::= ⟨identifier⟩

⟨list⟩ ::= ⟨list segment⟩ | ⟨list⟩ , ⟨list segment⟩

⟨list segment⟩ ::= ⟨expression part⟩ | ⟨for clause⟩ [⟨expression list⟩] | ⟨for clause⟩ ⟨list segment⟩

⟨expression part⟩ ::= ⟨arithmetic expression⟩ | ⟨list identifier⟩ | ⟨Boolean expression⟩

⟨expression list⟩ ::= ⟨expression list⟩ , ⟨expression part⟩ | ⟨expression part⟩

Examples:

```
LIST PLOT (X,Y)
LIST ANS (P+Q,Z,SQRT (Z), A[I])
LIST L1 (FOR I←0 STEP 1 UNTIL T DO FOR J←0 STEP 1 UNTIL U DO A[I,J])
```

Semantics:

The output ⟨list declaration⟩ specifies a list of expressions, the values of which are to be included in one output communication. The expression values are placed in the output string in the same order as their corresponding expressions in the ⟨list declaration⟩.

A ⟨for clause⟩ may be used to reduce the amount of writing required when the elements of an array are included in the output.

If a ⟨list declaration⟩ contains expressions which have already appeared in another ⟨list declaration⟩ , it is not necessary to list them again. Use of the previously declared ⟨list identifier⟩ is sufficient. Recursive use of ⟨list identifier⟩s is not allowed.

WRITE STATEMENT

Syntax:

⟨write statement⟩ ::= WRITE (⟨output parameters⟩)
⟨output parameters⟩ ::= ⟨file identifier⟩ ⟨format and list part⟩
⟨format and list part⟩ ::= , ⟨format identifier⟩ ⟨list part⟩ | ⟨empty⟩ |
[⟨carriage control⟩] ⟨format identifier⟩ ⟨list part⟩
⟨list part⟩ ::= , ⟨list⟩ | ⟨empty⟩
⟨carriage control⟩ ::= ⟨skip to next page⟩ | ⟨skip to channel⟩ | ⟨double space⟩ |
⟨no space⟩
⟨skip to next page⟩ ::= PAGE
⟨skip to channel⟩ ::= ⟨unsigned integer⟩
⟨double space⟩ ::= DBL
⟨no space⟩ ::= NO

Examples:

```
WRITE (A,C,M)
WRITE (C[DBL]F2)
WRITE (F2 [3]FORM,L2)
WRITE (F5 [PAGE])
WRITE (FILE2)
WRITE (FOUT2,FORM2,FOR I←0 STEP 1 UNTIL 13 DO HEAD[I])
```

Semantics:

Three kinds of declarations have been presented: file declaration, which establishes output buffer areas in memory; format declaration, which describes the form of the data needed for output; and list declaration, which provides the expression values that are to constitute the output.

The ⟨write statement⟩ serves to identify the specific place in the program where output is to occur. It also associates the declarations necessary for producing a given output. The ⟨list declaration⟩ is not required. The ⟨list⟩ can be specified directly in the ⟨write statement⟩.

More than one unit of output: printed line, punched card, magnetic tape record, or plotted point can be produced with a single write statement. An additional unit of output is called for each time one of the following occurs before the ⟨list⟩ is exhausted:

1. A slash appears in the format declaration
2. The end of the format declaration is reached.

The slash is used when units of output are to be given different formats. A format is assigned to each unit of output, according to the editing phrases contained between slashes. Two adjacent slashes produce a blank line when printing, or a blank card when punching. One slash at the beginning or end of a format declaration has the same effect.

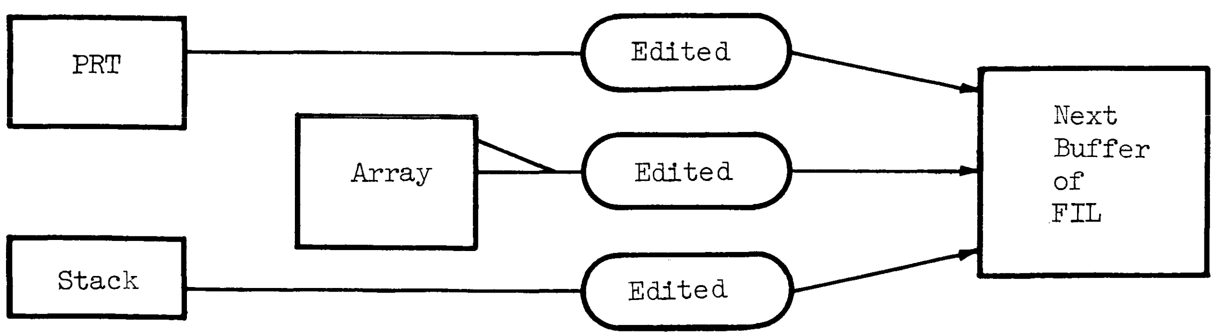
When all editing phrases in a format declaration have been used before the ⟨list⟩ is exhausted, editing of the next unit of output proceeds from the beginning.

The ⟨write statement⟩ may be expressed in several forms and perform a different function for each. The most common form is:

```
WRITE (FILE2,FORMAT3,LIST1)
```

This form is used when the output contains expression values. The process involved is shown in the following illustration.

LIST1 DECLARATION (Selected Values According to LIST1) | FORMAT3 DECLARATION (Edited by FORMAT3) | FILE2 DECLARATION (Stored in FILE2)



The expressions contained in the ⟨list⟩ are evaluated one at a time, from left to right. Their values are then edited according to the ⟨format declaration⟩. The ⟨editing phrase⟩s are applied to the expression values in left to right order and placed in the output buffer in the same order.

The writing process is completed when all expressions in the ⟨list⟩ have been evaluated and the values placed in the output buffer. If the buffer has not been filled and the next ⟨editing phrase⟩ is a ⟨string⟩, the ⟨string⟩ is placed in the buffer. In any case, the bottom of the buffer will be filled with blanks if the number of characters affected by the format declaration is less than the buffer size. After the buffer is filled, its contents are transferred to the output device automatically.

Another form of the ⟨write statement⟩ is one in which the ⟨list part⟩ is empty. This construct occurs when the entire output buffer is to be filled with data obtained from the ⟨format declaration⟩. Therefore, the ⟨editing specifications⟩ must contain only X-type and ⟨string⟩ editing phrases. No expression values are involved. This kind of output would occur, for instance, when headings are being printed.

```
WRITE (FILE1,F2)
WRITE (FILE2[DBL]FORM)
```

The ⟨carriage control⟩ has meaning only when output is being produced on a line printer. It serves to activate the carriage control tape on the line printer after printing a line. If other than a line printer is being used, the ⟨carriage control⟩, if any, is ignored.

```
WRITE(F2[3]FORM4,L2)
```

Several options are provided for ⟨carriage control⟩. If an integer is given, the tape skips to the next hole in the channel indicated by the integer. Channels are numbered from 1 to 12. If [DBL] is used, a double space occurs after printing. When no ⟨carriage control⟩ is present in a ⟨write statement⟩ which produces line printer output, the paper is single spaced after printing. If [PAGE] is used, the paper skips to the top of the next page (channel 1). [NO] will result in no spacing of the paper.

When it is necessary to space the paper before printing, the following form of ⟨write statement⟩ is used:

```
WRITE(F2)
WRITE(F2[PAGE])
```

PROBLEM 1. To illustrate the use of the type-A editing phrase, assume the same problem that was presented in Problem 1 of the Input Section. In that case, the date, programer's name, and department were read into an array called HEAD. Now consider what language is necessary to print that information. First assume it is all printed on one line and the next print line is controlled by channel 3 of the carriage control tape. Selected portions of the program would then be as follows:

```
BEGIN
   ARRAY HEAD [0:13]; INTEGER I;
   FILE OUT FOUT2 (1,15);
   FORMAT OUT FORM2 (X8,13A6,A2);
   LIST LOUT2 FOR I←0 STEP 1 UNTIL 13 DO HEAD [I]);

   WRITE (FOUT2[3] FORM2, LOUT2)
```

EXPRESSION VALUES HEAD [I]	EDITING PHRASES
none	X8
0 0 D A T E : b	A6 (13A6)
0 0 b N O V . b	A6
0 0 2 2 , b 1 9	A6
0 0 6 2 b b b b	A6
0 0 b b b N A M	A6
0 0 E : b b T H	A6
0 0 E O D O R E	A6
0 0 b I . b W E	A6
0 0 N D E L L b	A6
0 0 b b b b b b	A6
0 0 b b D E P T	A6
0 0 : b b E N G	A6
0 0 I N E E R I	A6
0 0 0 0 0 0 N G	A2

OUTPUT BUFFER FOUT2

b	b	b	b	b	b	b	b
D	A	T	E	:	b	b	N
O	V	.	b	2	2	,	b
1	9	6	2	b	b	b	b
b	b	b	N	A	M	E	:
b	b	T	H	E	O	D	O
R	E	b	I	.	b	W	E
N	D	E	L	L	b	b	b
b	b	b	b	b	b	D	E
P	T	:	b	b	E	N	G
I	N	E	E	R	I	N	G
b	b	b	b	b	b	b	b
b	b	b	b	b	b	b	b
b	b	b	b	b	b	b	b
b	b	b	b	b	b	b	b[6]

The resulting line of print would appear as follows:

Col.
1 10 20 30 40 50 60 70 80 90

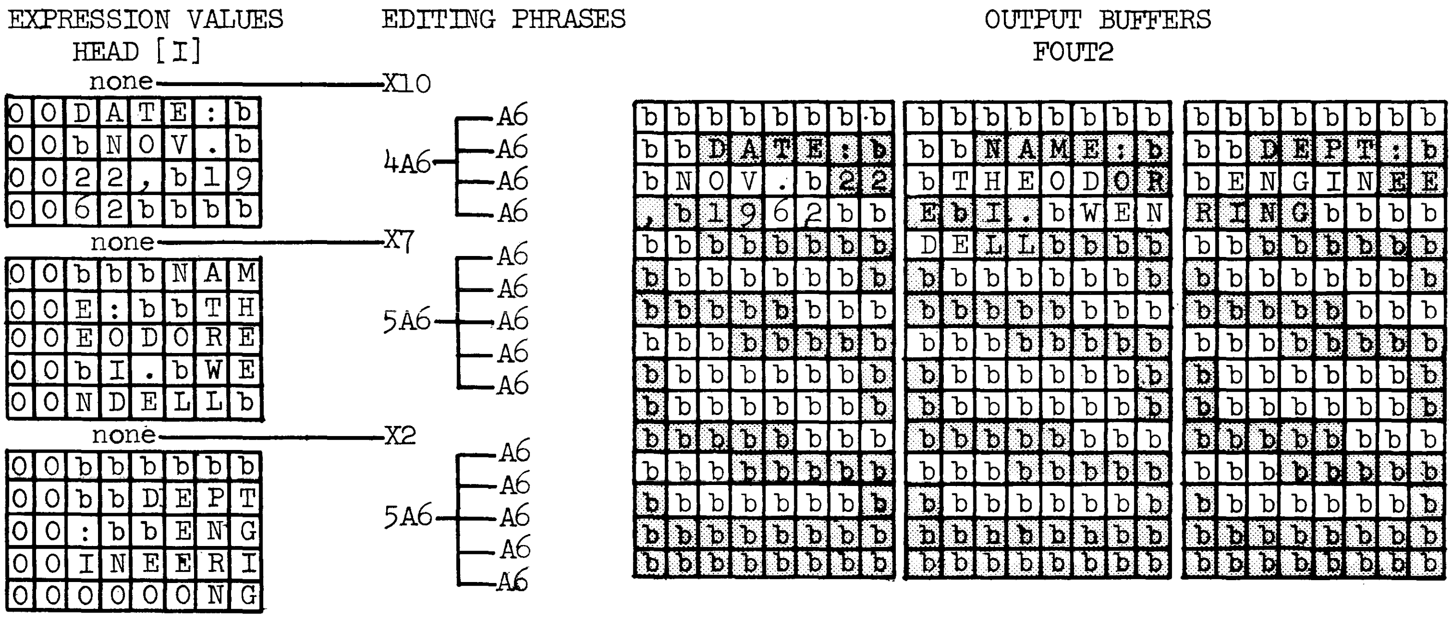
DATE: NOV. 22, 1962 NAME: THEODORE I. WENDELL DEPT: ENGINEERING

[6] Since the editing part of FORM2 affects only 88 characters, the last 32 blanks are supplied automatically to fill the buffer.

The following example assumes the heading information is printed with different formats on three different lines. Selected portions of the program might appear as follows:

```
BEGIN
    ARRAY HEAD [0:13]; INTEGER I;
    FILE OUT FOUT2 (3,15);
    FORMAT OUT FORM2 (X10,4A6/X.7,5A6/X2,5A6);

    WRITE (FOUT2, FORM2, FOR I←0 STEP 1 UNTIL 13 DO HEAD [I]);
    WRITE (FOUT2 [3])
```



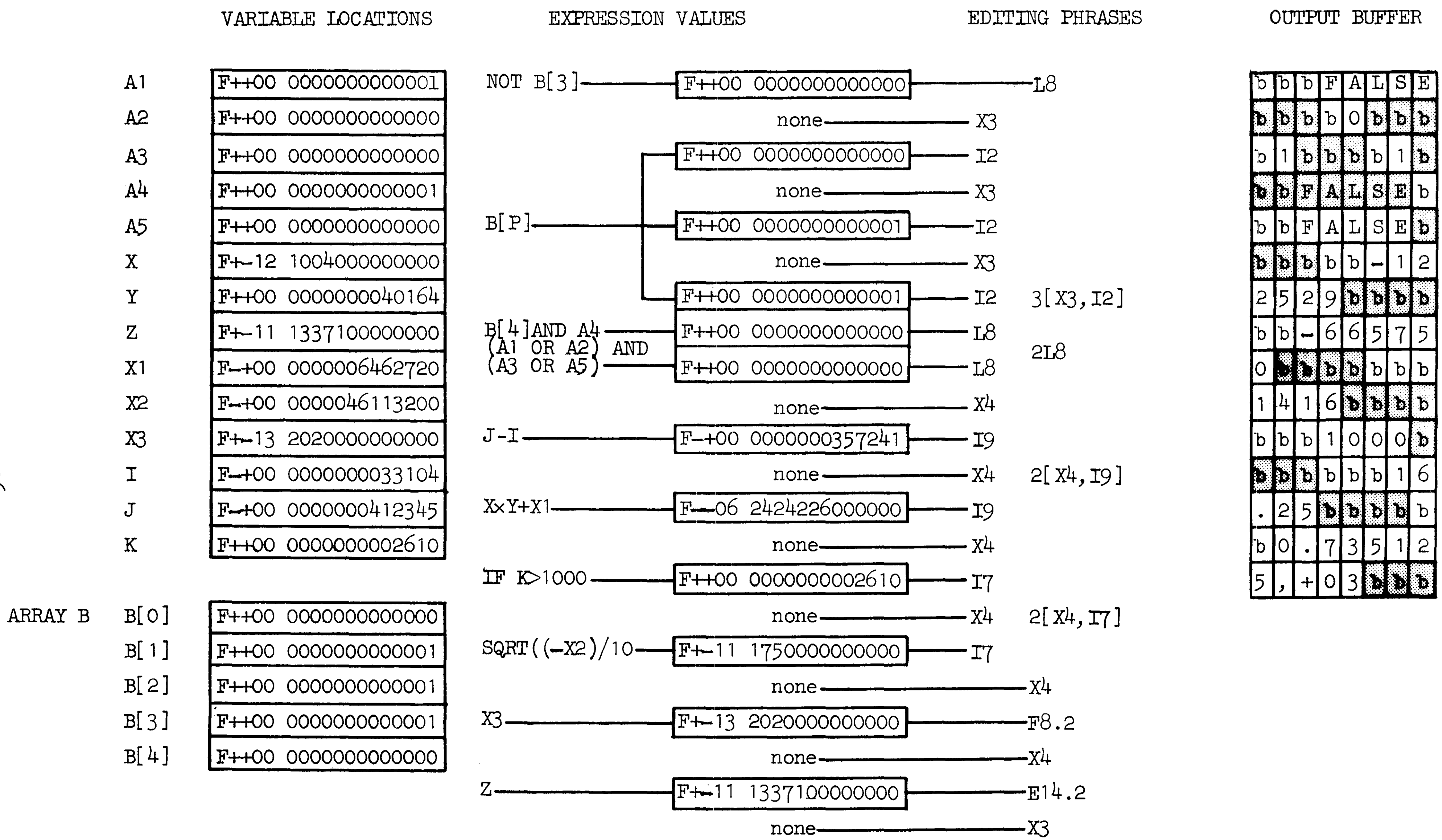
The resulting printout would appear as follows:

```
COL.
 1       10        20        30        40        50        60        70        80

          DATE:   NOV. 22, 1962
          NAME:   THEODORE I. WENDELL
          DEPT:   ENGINEERING
```

PROBLEM 2. To illustrate the output of numeric and logical values, consider the following: Assume that output is produced by the line printer. One line is to be printed with single spacing after printing. The variables and their values are taken from Problem 2 of the Input Section.

```
BEGIN
   BOOLEAN ARRAY B [0:4];
   BOOLEAN A1, A2, A3, A4, A5;
   REAL X, Y, Z, X1, X2, X3;
   INTEGER I, J, K, P;
   FILE OUT FI1 (1,15);
   FORMAT OUT F02 (L8,3[X3,I2], 2L8, 2[X4,I9], 2[X4,I7], X4, F8.2, X4, E14.6, X3)
   LIST LI3 (NOT B[3], FOR P←0 STEP 1 UNTIL 2 DO B[P], B[4] AND A4,
            (A1 OR A2) AND (A3 OR A5), J-I, X×Y+X1, IF K> 1000 THEN K ELSE
            1000, SQRT ( (-X2)/10), X3, Z);
   WRITE (FI1, F02, LI3)
```



The resulting line of print would appear as follows:

FALSE 0 1 1 FALSE FALSE -122529 -665750 1416 1000 16.25 0.735125,+03

PROBLEM 3. This example describes the process of output to the plotter. Assume that values have been computed and stored in an array. A program for plotting the above results would appear in part as follows:

```
BEGIN
   ARRAY W[0:10];
   INTEGER X,I;
   FILE OUT F(2,1);
   FORMAT OUT FM (P5.3);
   LIST PT(FOR I←0 STEP 1 UNTIL 10 DO[X,W[I]×100]);

   X←1;
   WRITE (F,FM,PT)
```

This write statement will result in the plotting of eleven points. The symbol used is a small circle. The printing of a grid is inhibited since the plotting paper is pre-printed. Each point is plotted one increment of spacing (.025 inches) to the right of the previous point. The printing of the least significant digit of each ordinate value is designated. This digit is printed directly under the associated point at the bottom of the page.

SUMMARY OF WRITE STATEMENT. The write statement and the associated file, format and list have been designed to accommodate output requirements for scientific problems.

The programer, however, is not restricted to the output features available to him in the write statement. He has the option of creating output buffers by use of the file declaration and then filling those buffers in any way desired. This is done by writing a STREAM PROCEDURE to fill the buffer instead of a format declaration. The file identifier is then used as an actual parameter to the STREAM PROCEDURE.

When the write statement is used, automatic transfer of data from buffer to output device is accomplished as soon as a buffer is filled. Therefore a statement to initiate this transfer is needed when the write statement is not used. Such a statement is the release statement with an output file identifier as a parameter.

OUTPUT RELEASE STATEMENT

Syntax:

⟨release statement⟩ ::= RELEASE (⟨file identifier⟩)

Examples:

```
RELEASE (F1)
RELEASE (POUT)
```

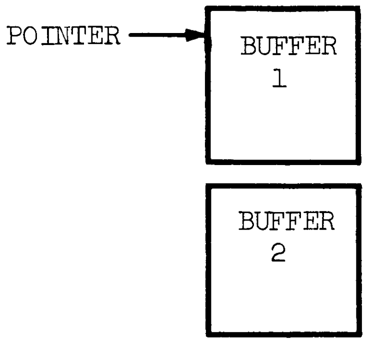
Semantics:

The output release statement causes the contents of one output buffer to be transferred to the appropriate output device. If more than one buffer is being used, a reordering occurs which maintains the first-in, first-out operation.

PROBLEM 1. To illustrate the buffer-ordering procedure, assume an output file which uses two buffers. Since there is only one name (file identifier) for both of these buffers, some way is needed to distinguish between them. The means employed is a pointer, which is pointing at the buffer to be used the next time the file is called for.

```
BEGIN
   FILE OUT F2 (2,15);
```

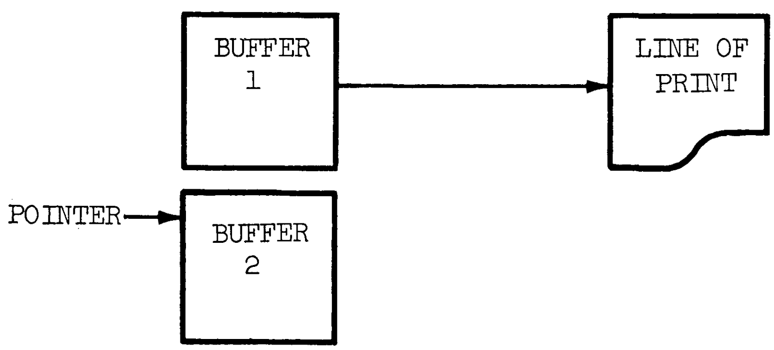
This would establish two output buffers, each 15 words in length.



RELEASE (F2);

Any use of the file identifier (F2) at this point in the program refers to buffer 1. The release statement results in the pointer being shifted to buffer 2, and the contents of buffer 1 being transferred to the line printer, assuming it to be the specified output device.

Since a write statement has an implied release at the end of its operation, the effect on the buffers when it is used is the same as shown here.



APPENDIX A

COMPUTER WORD STRUCTURES IN THE B 5000

The following information is presented here for reference purposes only. A complete description of the B 5000 is contained in The Descriptor, Bulletin 5000-20002-P.

There are two basic forms in which data can be represented in the B 5000. This manual depicts them as follows:

NUMERIC OPERAND

BITS	48	47	46	45–39	39–1
	Identification Flag	Sign of Mantissa	Sign of Exponent	Exponent 2 Digits	Mantissa 13 Digits

The illustrations in this manual which show numeric values are in the above form. The exponent and mantissa are composed of octal digits and each illustration contains the octal values.

CHARACTER WORD

BITS	48–43	42–37	36–31	30–25	24–19	18–13	12–7	6–1
	First Character	Second Character	Third Character	Fourth Character	Fifth Character	Sixth Character	Seventh Character	Eighth Character

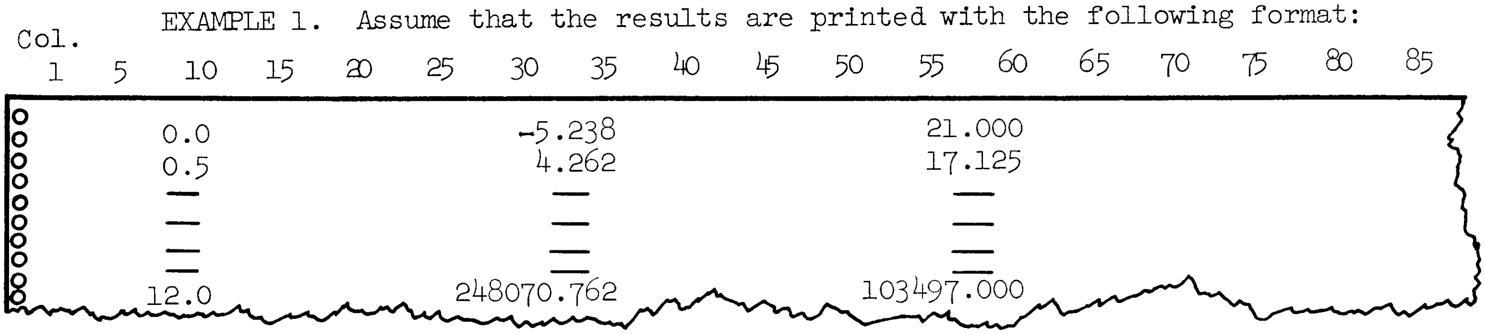
The illustrations which show alphabetic data are in the above form. Each character is composed of 6 bits.

APPENDIX B

PROBLEM SOLUTIONS

An Introduction to Algol 60, Bulletin 5000-21001-P, (June 1961) contains sample problems. The solutions to these problems do not include the input-output portion since the language was not completely formulated at that time. The complete solutions to those problems are given below.

EXAMPLE 1. Assume that the results are printed with the following format:



The program will then be:

```
BEGIN
   REAL X,Y,YPRIME;
   FILE OUT A(1,15);
   FORMAT OUT C (X5,F5.1,2[X14,F11.3]);

   FOR X←0 STEP .5 UNTIL 12 DO
        BEGIN
           Y←X*5-(X*3)/4-4×X*2+21×X-5.238;
           YPRIME←5×X*4-(3×X*2)/4-8×X+21;
           WRITE (A,C,X,Y,YPRIME)
        END
END
```

Example 2. Assume the results are printed with the following format:

```
COL.
 1  5   10   15   20   25   30   35   40   45   50   55   60   65   70

          N            L                  S

         50          360.0              9425.0
         75          535.0             20700.0
        100          710.0             36350.0
         —             —                  —
         —             —                  —
         —             —                  —
         —             —                  —
        300         2110.0            319050.0
```

The program will then be:

```
BEGIN
   REAL L,S; INTEGER N;
   FILE OUT C (1,15);
   FORMAT OUT I (X4,I4,X8,F7.1,X12,F9.1),J (X6,"N",X12,"L",X19,"S");
   LIST O (N,L,S);

   WRITE (C[DBL]J);
   FOR N←50 STEP 25 UNTIL 300 DO
        BEGIN
           L←17+(N-1)×7;
           S←N/2×(17+L);
           WRITE (C,I,O)
        END
END
```

Example 3. Assume the results are printed with the following format:

```
COL.
 1   5    10   15   20   25   30   35   40   45   50   55   60   65   70
          0.1740,+04          0.1176,+02          0.3748,+01
```

```
BEGIN
   REAL A, RS, RT; INTEGER W, L, H;
   FILE OUT J (1,15);
   FORMAT OUT M (3[X5,E11.4]);

   W←12; L←27; H←14; A←2(W×L+W×H+L×H);
   RS←SQRT (A/(4×3.14159265359));
   RT←A/(4×RS×3.14159265359*2);
   WRITE (J,M,A,RS,RT)
END
```

Example 4. Assume the input to be from cards in the following form:

The first card contains the value of N in columns 1-4. Subsequent cards contain the values of the N elements of array X. They are punched as decimal numbers with two places after the decimal point. Every ten columns contain one value and the last ten columns are used for a sorting sequence number. Therefore each subsequent card contains seven values.

Assume the two possible outputs of the program to be printed in the following formats:

```
COL.     1   5   10   15   20   25   30   35   40   45   50   55   60   65   70   75   80   85   90   95

Line  1
      2                       COMPUTATION OF MEAN VALUE, STANDARD DEVIATION, AND GREATEST DEVIATION
      3
      4                                  ±nnnnnn.nn        ±nnnnnn.nn        ±nnnnnn.nn    nnnn
      5
      6
If J=1 7†                                       NO VALUES OF X[I] ARE EQUAL TO A


If J=1 7†                              SEQUENCE NUMBERS OF TERMS EQUAL TO THE MEAN VALUE
      8
      9          nnnn   nnnn   nnnn   nnnn   nnnn   nnnn   nnnn   nnnn
```

†Assume skip to line 7 is controlled by a punch in channel 3 of printer's carriage control tape.

The program will then be:

```
BEGIN
   REAL A,SUM,SUMSQ,STANDEV,MAXDEV,Z;
   INTEGER N,I,J,INDEX;
   INTEGER ARRAY Y [1:1000];
   REAL ARRAY X [1:1000];
   FILE IN F1 (2,10);
   FILE OUT F2 (2,15);
   FORMAT IN FORM1 (I4),FORM2 (7[F10.2]);
   FORMAT OUT FORM3(B24,"COMPUTATION OF MEAN VALUE, STANDARD DEVIATION, AND
        GREATEST DEVIATION"),FORM4 (X31,3[X8,F10.2],X3,I5),
        FORM5 (X43,"NO VALUES OF X[I] ARE EQUAL TO A")
        FORM6 (X35,"SEQUENCE NUMBERS OF TERMS EQUAL TO THE MEAN VALUE"),
        FORM7 (X8,15[X2,I5],X7);
   LIST L2(A,STANDEV,MAXDEV,INDEX);

   READ (F1,FORM1,N);

   BEGIN
        INTEGER P;
        LIST L3(FOR P←1 STEP 1 UNTIL N DO X[P]);

        READ (F1,FORM2,L3)
   END;

   SUM←0;
   FOR I←1 STEP 1 UNTIL N DO SUM←SUM+X[I];
   A←SUM/N; SUMSQ←0;
   FOR I←1 STEP 1 UNTIL N DO SUMSQ←SUMSQ+X[I]*2;
   STANDEV←SQRT (SUMSQ/N-A*2);
   MAXDEV←ABS(X[1]-A); INDEX←1;
   FOR I←2 STEP 1 UNTIL N DO
        BEGIN Z←ABS(X[I]-A); IF Z >MAXDEV THEN
             BEGIN MAXDEV←Z; INDEX←I END
        END;
```

```
  WRITE (F2[DBL]FORM3);
  WRITE (F2[3]FORM4,L2);
  J←1;
  FOR I←1 STEP 1 UNTIL N DO IF X[I]=A THEN
  BEGIN Y[J]←I; J←J+1 END;
  IF J=1 THEN WRITE (F2,FORM5) ELSE
  BEGIN
       INTEGER Q;
       LIST L4 (FOR Q←1 STEP 1 UNTIL J-1 DO Y [Q]);

       WRITE (F2[DBL]FORM6);
       WRITE (F2,FORM7,L4)
  END
END
```

APPENDIX C

MINIMUM BUFFER SIZES

Whenever buffers are established in memory by means of a file declaration, it is necessary to indicate their sizes. The following table is presented for reference purposes, and indicates the minimum buffer size required for each type of I-O equipment.

INPUT	FILE (Buffer Size)	
80-col. Alpha Card	10	
80-col. Binary Card	20	
Magnetic Tape	1-1023	Dependent upon number of words per record
OUTPUT		
80-col. Alpha Card	10	
Magnetic Tape	1-1023	Dependent upon number of words per record
Printer	15	
Plotter	1	