ALPHABETICAL INDEX TO LIBRARY PROGRAMS

Description	Date of first issue	Name	Language	Chapter
ALGOL CARD READER PROCEDURES	February,1966	CARD	A	42
ALGOL C.B.S. ARRAY PROCEDURES (Version 1)	December,1965	CBSA	A	31
ALGOL C.B.S. ARRAY PROCEDURES (Version 2)	December,1965	CBSB	A	32
ALGOL 5 to 8	November,1963	ALGOL 5	S	5
ALGOL LINEPRINTER PROCEDURES	November,1964	LPRALG	A	24
ALGOL MAGNETIC TAPE PROCEDURES	May,1966	MTALG	A	43
ALGOL MATRIX PACKAGE	November,1963	MTX01	A	2
ALGOL MATRIX PACKAGE MARK 2	May,1966	MTX02	A	40
ALGOL PLOTTER PACKAGE	September,1965	PLOT	A	26
ARCTANGENT	November,1963	arctan	S	10
AUTOCODE TO ALGOL TRANSLATOR	January,1966	AUTALG	A	36
AUTOCODE MAGNETIC TAPE BATCH ROUTINE	July,1967	SETINT	C	50
AUTOCODE 5 to 8	November,1963	ACconv	S	6
BACKGROUND PROGRAM DEVICE ROUTINES (MTREAD, MTWRITE,EDIT8(BG),EDIT81 (BG),LPRINT(BG))	August,1966	BGPROG	S	47
BESSEL FUNCTIONS OF INTEGRAL ORDER	November,1963	BES01	C	3
BESSEL FUNCTIONS J0,J1, Y0,Y1	November,1963	bessel	S	18
DOUBLE LENGTH ARITHMETIC SUBROUTINES (ADDITION,SUBTRACTION, MULTIPLICATION and DIVISION)	January,1966	DADD DSUB DMULT DDIV	S S S S	33 33 34 35

Description	Date of first issue	Name	Language	Chapter
EDITAL	January,1964	EDITAL	S	21
EDIT8	January,1966	EDIT8	S	27
EDIT 8 INTERFACE	February,1966	EDIT8I	S	37
ERROR AND PROBABILITY FUNCTIONS	November,1963	approx	S	12
EXPONENTIAL	November,1963	exp	S	8
FORTRAN TO ALGOL TRANSLATOR	October,1965	FEAT	A	29
GAUSSIAN PROBABILITY INTEGRAL	November,1963	Gauss	A	4
GENERAL PURPOSE ROUTINES	July 1969	CP	S	44
GENERAL MAGNETIC TAPE ROUTINE	November,1967	GMT	S	48
GENERAL MAGNETIC TAPE READ ROUTINE	November,1967	GMTR	S	48
GENERAL MAGNETIC TAPE WRITE ROUTINE	November,1967	GMTW	S	48
GENERAL NUMBER PRINT (lineprinter and paper tape punches)	November,1965	NPRINT	S	28
GENERAL SORT	November,1963	gensor	S	14
INTEGRATION OF A SET OF SIMULTANEOUS FIRST ORDER DIFFERENTIAL EQUATIONS	November,1963	intdif	S	17
LINEPRINTER OUTPUT	February,1964	LPRINT	S	22
LINEPRINTER CONTROL ROUTINE - 8 CHANNEL TAPES	November,1967	LP 8	S	49
LINEPRINTER CONTROL ROUTINE - 5 CHANNEL TAPES	November,1967	LP 5	S	49
LINEPRINTER LINE ASSEMBLER	May,1966	LINEAS	S	45

Description	Date of first issue	Name	Language	Chapter
LOGARITHM	November,1963	log	S	7
MAGNETIC TAPE PROGRAM MATCHES (DUMP,BRING & LOAD)	Septmber,1965	BATCH	S	25
(DUMP MARK 2)	May,1966	DUMP2	S	46
(DUMP 2M)	October,1966	DUMP2M	S	46
MAGNETIC TAPE ROUTINES (HANDLER4)	July,1969	MTSTOR	S	41
MAGNETIC TAPE SORT PROGRAM	May,1966	INTERN	S	38
NUMERICAL INTEGRATION OF ORDINARY DIFFERENTIAL EQUATION	May,1966	NIODE	A	39
803 OPERATOR	December,1964	CON 01	S	1
POLYNOMIAL INTER-POLATION AND EXTRA-POLATION	November,1963	polint	S	16
POST MORTEM LISTING OF STORE ON LINE PRINTER	December,1965	PML	S	30
PTSPRINT	January,1964	PTSPR1	S	20
PTSREAD	January,1964	PTSREA	S	19
ROOTS OF A POLYNOMIAL	April,1964	BAIRST	S	23
SINE,COSINE or TANGENT	November,1965	trig	S	9
SINGLE WORD,MULTIPLE KEYFIELD SORT	November,1963	multke	S	15
SINGLE WORD SORT	November,1963	swsort	S	13
SQUARE ROOT	November,1963	sqrt	S	11

LANGUAGE CODE

A ALGOL

C AUTOCODE

S SAC

CONFIGURATION CODE	ISSUE NO.	DESCRIPTION
P	3	ALGOL 1 PLOTTER PACKAGE/ALG
PCBSMT	1	ALGOL 3 PLOTTER PACKAGE/ALG
P	1	EDITPLOT
PCBSMT	1	CHAROU (PLOTTER VERSION) /SAC
BP	1	PLOTTERMOD/BIN
CBSMT	2	A3C/BIN
CBSMT	2	A3D/BIN
CBS	2	ALGOLB/SAC
CBSNOLP	2	ALGOLB (NO LINEPRINTER) /SAC
		THE FOLLOWING TAPES ARE AVAILABLE ON REQUEST TO THE ADDRESS GIVEN IN PAGE 5 OF NEWSLETTER P13.
A	2	PCCP/SAC
MT	1	MTFEAT/ALG
MT	1	MTSTOR/SAC
CBSMT	2	FEAT STANDARD PROCEDURES/TAPE 1/ALG.
*CBSMT	2	FEAT STANDARD PROCEDURES/TAPE 1c/ALG.
MT	2	INTERN/SAC
CBSMT	2	A3C/TI
CBSMT	2	A3D/TI
CBSMT	1	A3L/TI
CBSMT	1	A3C,A3D,A3L EDITS (COMBINED TAPE)
A	1	T2/SAC
A	1	TRANSA/BIN
LP	1	FILIP/ALG
CBSLP	1	FILIP (FOR RGCBS)/ALG
CBS	1	EDITALL (FOR RGCBS)/SAC
CBS	1	RGCBS/SAC

*THIS ROUTINE ALSO REQUIRES A CARD READER

CHAPTER 1: 803 OPERATOR

CODE CON O1S

FUNCTION

The function of this program is to make the control of the 503 obey the word generator. The control of the 503 normally obeys the control typewriter, whereas the control of an 803 is via the word generator or its equivalent. This enables the 503 to be run 'as an 803' and hence makes it possible to use almost all the Library and Applications Programs of the 803 on the 503. This includes 803 Algol and 803 Autocode. Appendix 1 describes special provisions for 803 C2, and Appendix 2 describes the tape copying routines available for use on the 503.

Further information about 803 Programs can be obtained from the 803 Librarian and the Applications Group Secretary, Scientific Computing Division.

STORE USED

Locations 8162—8191 inclusive.

METHOD OF USE

Input of 803 Operator tape

The 803 Operator uses locations 8162 to 8191 which are normally used by sum checked binary tapes to contain the input routine.

Thus:—

All sum checked binary tapes and any others which require locations 8162 to 8191 during input must be read in before the 803 Operator.

When all other such tapes are in:

(a) Ensure that the NOPROTN button is depressed.

(b) Clear the word generator.

(c) Press the RESET button.

(d) Place the 803 Operator tape in Reader 1 and press the Initial Instructions button.

One or more of the characters listed under 'Action' should be displayed on the typewriter. If nothing is displayed and/or spaces are output on punch 1, repeat from (c).

If the last character displayed is an I then the next depression of the MESSAGE button will cause the word generator to be read but if any other character is displayed repeat from (c) above.

Action

The 803 Operator is used in conjunction with the MESSAGE button. Each depression of the MESSAGE button causes either I, R, D, or DA to be displayed.

CHARACTER	MEANING
I	The current program has been Interrupted and control is now with the 803 Operator. The next depression of the MESSAGE button causes R to be displayed.
R	The word generator has been Read. The next depression of the MESSAGE button causes D or DA to be displayed.

D	The word generator has been obeyed (Done).
A	Obeying the word generator did not cause a transfer of control. The next depression of the MESSAGE button will cause the word generator to be read Again.

Note 1 Only the F_1 N_1 digits of the word generator are obeyed (as for the 803).

2 After the word generator has been read (i.e., R displayed) there is no way to avoid obeying it.

Error Indications

If at any time an error interrupt occurs, 'E' is displayed on a new line followed by one or more of 'a', 'b', 'd' or 'p' with the significance indicated below, and the program waits:

a	The error was due to an impermissible reference to the protected reserved area.
b	The error was due to an impermissible reference to a peripheral device.
d	The error was due to main store parity failure.
p	The error was due to floating point overflow.

The next depression of the MESSAGE button causes the word generator to be read.

Special Facilities

1. If the last character displayed is I, a, b, d, p, or A then the main program may be continued by causing the Operator to obey 44 8189.
2. After the word generator is obeyed then, provided the word generator did not contain a transfer order (GROUP 4), the value of the accumulator is stored. The accumulator is restored to this value immediately before the next instruction on the word generator is obeyed.

RESTRICTIONS

1. 803 Programs using Film, the Card Reader or Line Printer cannot be used on the 503 as it has no Film and the Card Reader and Line Printer have different codes. However, if Film, Card or Line Printer orders are given the 803 Operator will not be destroyed.
2. The 803 Operator cannot be used with programs that make use of locations 8162-8191 during running, except to start them initially.
3. A restriction on the use of the 503 as an 803 is that any 5 hole output for the 803 direct teleprinter (punch 3) will be printed on the output writer on the 503, which is set for 8 channel code.

   The 5 channel output will thus appear unintelligible, although it can be partially decoded.

FORMAT OF RESULTS

Only the print out on the direct output writer is a 'result' of the 803 Operator itself. Other results are as indicated by the appropriate 803 program.

CONFIGURATION

The basic 503 computer and five hole paper tape equipment [reader(s) and punch(es).]

TAPE

Pure binary with a self-contained sum check routine. A tape is provided.

EXAMPLE OF USE OF THE 803 OPERATOR

Translate and run an Autocode program in which checks are to be used.

	Operation	Display
(a)	Place 803 A103 Tape 1 in Reader 1 (Make sure that the MODE button is depressed).	
(b)	Press Initial Instructions button.	
(c)	Press Initial Instructions button again.	
(d)	Place 803 A103 Tape 2 in Reader 1.	
(e)	Press Initial Instructions button.	
	Clear word generator.	
	Press RESET button.	
	Place 803 Operator in Reader 1.	
	Press Initial Instructions button.	I
(f)	Place mnemonic tape in Reader 1.	
(g)	Set 40 7B on word generator	
	Press MESSAGE button	R
	Press MESSAGE button again	D
	After the dynamic stop at the end of the translation has been reached, press MESSAGE button.	I
(j)	Place data tape in reader.	
(k + *l*)	Set 40 16B on word generator	
	Press MESSAGE button	R
(m)	Press MESSAGE button	D

The letters on the left hand side correspond to the paragraphs on pages 38-40 in the 803 A3 and A103 description.

S. M. Quinn. *December*, 1963.

APPENDIX 1

Use of the 803 Operator with 803 C2

To print the contents of the store use the Operator in the normal way.

However, paragraphs (i), (v), (vii) of page 2 (Issue 2) of the 803 C2 description should be replaced by:—

(i) If it is intended to print the content of the sequence control register, read and obey each of the following instructions:—

```
30    8181
51       1
20       4
```

(The sequence control register is the device which holds the address of the instruction which was due to be obeyed when interrupt occurred.)

(v) When used with the 803 Operator, the auxiliary register has address 8180 and the accumulator has address 8179. The content of location 4 is assumed to be the content of the sequence control register and if it is printed it is preceded by SCR instead of the address.

(vii) When all the required information has been obtained the accumulator, auxiliary register, overflow register and sequence control register may be reset by reading and obeying the instruction 44 8189.

APPENDIX 2

503 Tape Copying Routines

The recognised method of copying tapes on the 503 is using EDIT ALL (2.2.3.21.) but the following routines are also available. These are modified versions of 803 C1(B) with a copy of the 803 Operator program included on the tape. 803 C1(B) cannot be used with the 803 Operator since it clears the rest of the store when it is entered and therefore 503 C1(B) is issued.

There are 2 tapes provided, they are:—

503 C1 (B) (7 bit)

503 C1 (B) (5 bit)

Both tapes are coded in 5-hole binary and the operating instructions are the same as for 803 C1(B) (see the 803 Program Library) using the 803 Operator program.

R. A. Finch. *April*, 1964.

CHAPTER 2: ALGOL MATRIX PACKAGE

CODE MTX 01A (Issue 2)

FUNCTION

The following set of procedures perform the standard operations of matrix arithmetic. Certain optimising techniques have been used within the procedures which ensure that they will run faster than corresponding procedures written in conventional ALGOL.

STORE USED

About 1240 locations in all. If storage space is critical, unused procedures may be omitted.

CONFIGURATION

The requirements are as for 503 ALGOL.

PARAMETERS

The parameters of the routines are, in general, array names. The vectors and matrices which are used as actual parameters must have been declared in the main program as TWO-DIMENSIONAL arrays with appropriate subscript bounds, e.g., A column—or row—vector A should be declared as A [1 : m, 1 : 1] or A [1 : 1, 1 : m].

The first parameter usually gives the array in which the result of the operation is stored, and subsequent parameters specify the operands. Storage space for results is allocated when the procedure is called, the subscript bounds of the operands then being known. In general, the procedures make all tests for compatibility, etc., which are necessary for the performance of the operation intended.

SUMMARY OF PROCEDURES

Throughout this summary

A, B and C	represent two-dimensional real arrays (matrices)
x	represents a real scalar, e.g., a matrix element
i and j	represent integers used as suffices to indicate particular array elements.

N.B. An alternative print procedure for larger matrices is given in Appendix 1.

```
procedure   mxaux (A, B, C, d, e);
            value d, e; boolean d, e; array A, B, C;
            comment this procedure is used in mxsum, mxdiff, mxcopy, mxneg and mxquot as
                    an auxiliary procedure. If any of these procedures are used outside the
                    matrix package then mxaux must also be in store. The procedure tests that
                    the arrays A, B, and C are the same size. If they are not then the message
                    "mxaux error" is displayed and the program is terminated;

procedure   mxdiff (A) becomes: (B) minus: (C);
            array A, B, C;
            comment this procedure which uses mxaux subtracts array C from array B and
                    stores the result in array A. A may be the same as either B or C;

procedure   mxsum (A) becomes: (B) plus: (C);
            array A, B, C;
            comment this procedure which uses mxaux adds array B to array C and stores the
                    the result in array A. A may be the same as either B or C;
```

procedure mxcopy (A) becomes: (B);

array A, B;

comment this procedure which uses mxaux and is used by mxquot copies array B, the copy becoming array A;

procedure mxneg (A) becomes minus: (B);

array A, B;

comment this procedure which uses mxaux negates array B. The result is stored in array A. Array A may be the same as array B;

procedure mxprod (A) becomes: (B) times: (C);

array A, B, C;

comment this procedure forms the matrix product of arrays B and C. The result is stored in array A.

A must not be the same array as B or C. If the arrays are incompatible the message "mxprod error" is displayed and the program is terminated;

procedure formmx (A) becomes: (x) with respect to: (i) and: (j);

real x; **array** A; **integer** i, j;

comment this procedure forms an array A where each element A [i, j] is a function of i and j. This is done by use of Jensen's device (see E.W. Dijkstra's "A Primer of Algol Programming", Academic Press 1962, P.57). Briefly, Jensen's device allows a procedure to treat a formal parameter t as a function of the actual parameter corresponding to another formal parameter i.

The actual parameters are:

A a two-dimensional real array.

x a real expression defining the value of the element A [i, j].

i, j two integer parameters which specify respectively the row and column number of the array element. The lower limit of both these parameters is 1;

procedure mxtrans (A) becomes transpose of: (B);

array A, B;

comment this procedure forms the transpose of array B and stores the result in array A. Array A may not be the same as array B. If A and B are incompatible the message "mxtrans error" is displayed and the program is terminated;

procedure scprod (A) becomes A times the scalar: (x);

value x; **real** x; **array** A;

comment this procedure multiplies array A by the scalar x in situ;

procedure mxquot (B) becomes: (A) to minus one times: (C);

value A; **array** A, B, C;

comment this procedure which uses mxaux and mxcopy solves a set of simultaneous equations $A \times B = C$ in one procedure call. If the call by value is

omitted, both space and time will be saved, but A will be destroyed during the computation. The method used is Gaussian elimination. The solution array B, has the same number of rows and columns as the right-hand side, array C.

The procedure contains certain checks by which, if the B-digit is depressed at translation and run time, at each stage of the elimination the ratio of the new pivot to the greatest pivot to date is output.

If at any stage the ratio of the new pivot to the greatest pivot is less in magnitude than 10^{-6}, the following message is displayed

" mxquot : pivot ratio = α size β stage γ "

and a data wait is entered. It is possible to continue, but the results are unlikely to contain any correct significant figures. Failure at the first stage is specially distinguished. If the arrays A and B are incompatible, the message "mxquot error" is displayed and the program is terminated;

procedure invmx (A);

array A;

comment this procedure inverts the non-singular array A in situ. The program uses the Gaussian elimination method, searching for the maximum element in each column and using these elements as pivots.

The procedure contains certain checks by which, if the B-digit is depressed at translation and run time, at each stage of the elimination the ratio of the new pivot to the greatest pivot to date is output.

If at any stage the ratio of the new pivot to the greatest pivot is less in magnitude than 10^{-6}, the following message is displayed

" invmx : pivot ratio = α size β stage γ "

and a data wait is entered. It is possible to continue, but the results are unlikely to contain any correct significant figures. Failure at the first stage is specially distinguished.

If A is not square the message "invmx error" is displayed and the program is terminated;

procedure printmx (A);

array A;

comment this procedure prints an array by rows. Each element is printed on the device and in the format current when the procedure is called.

The format is specified before the procedure call, e.g.:

punch (2)

prefix (£, £ S2 ? ?)

freepoint (4)

printmx (A)

Each row is printed on a new line. No row or column numbers are printed nor is any facility included for the printing of large matrices;

procedure readmx (A);

array A;

comment this procedure reads any real number and places the result in an element of the matrix A. Successive elements are placed in the same row of the matrix until the row has been filled. Data is punched as described on page 6, section 2.1.3.1 of the 503 Technical Manual.

Reading takes place on the device current when the procedure is called. The device setting may be altered before readmx is called.

e.g. reader (2)
readmx (B);

METHOD OF USE

The package ends with a H haltcode (telecode value 76) sign. The Master program requires an additional **'end'.**

ACCURACY

Single-length floating point arithmetic is used in all the procedures.

STORAGE

When the MATRIX PACKAGE is in store, locations 20-4900 (approx.) are available for program and data.

TAPE

A mnemonic tape is provided.

TIME

The procedures have been timed using 5 x 5, 10 x 10 and 20 x 20 matrices on the 503.

Times where available

5 x 5		10 x 10	20 x 20
PROCEDURE			
FORMMX		Depends on the function	
MXSUM	.005 secs	.019 secs	.065 secs
MXDIFF	.005 secs	.019 secs	.065 secs
MXCOPY	.005 secs	.019 secs	.065 secs
MXNEG	.005 secs	.019 secs	.065 secs
MXPROD	.033 secs	.207 secs	* 1.51 secs
SCPROD	* .005 secs	* .019 secs	.065 secs
INVMX	.065 secs	.330 secs	2.1 secs
MXTRANS	* .005 secs	* .019 secs	* .065 secs
MXQUOT		Depends on the matrices	
READMX	†	†	†
PRINTMX	†	†	†

* The times marked with an asterisk are approximate

† Reading and printing take place at the full speed of the reader and punch on the 503

EXAMPLE

Evaluate E: = A * B + C^{-1}

where E and C are 5 x 5 matrices
A is a 5 x 10 matrix
and B is a 10 x 5 matrix.

A and B are to be read from paper tape and C is defined by C [i, j] = 1 + 1/ (i + 2*j).

```
evaluate A * B + C ↑ (- 1);
begin array a [1 : 5, 1 : 10], b [1 : 10, 1 : 5], c, d, e [1 : 5, 1 : 5];
integer  i, j;
comment  d is work space;
         formmx (c, 1 + 1/ (i + 2*j), i, j);
         readmx (a); readmx (b);
         mxprod (d, a, b);          comment D:=A * B;
         invmx  (c);                comment C:=C^-1;
         mxsum  (e, c, d);          comment E:=A * B + C^-1;
         printmx(e);
end      evaluate
end      matrix package;
```

R. V. Wood.
P. Simmons.

April, 1965.

APPENDIX 1

FUNCTION

In cases where the number of columns is too great for the matrix to be printed on one level either of the following output procedures may be used.

procedure printcol (A);

array A;

comment this procedure prints the array A by columns. The elements of a column are all output on the same line and each column is output on a new line. The format for printing is set before the procedure is called. This procedure is due to Mr. R. Tobler, National Cash Register Company, Zurich, Switzerland;

procedure mxoutput (A, m, n);

value m,n;

integer m,n;

array A;

comment this procedure prints the array A by rows (see example) together with its row and column numbers. The row and column numbers are printed in the format digits (3). If the array has so many columns that it will not fit on the available output sheet, then the procedure will arrange for the array to be output on more than one level (see example).

The parameters of the procedure are:—

A, a real array,

m, the number of characters available across the output sheet and
n, the number of characters occupied by each element of the array.

The format for printing elements of the array is set by the programmer before the procedure is called. From this n, the number of characters per element, may be calculated, e.g.

freepoint(t): t+2 characters are required by each element.

aligned(r,s): r+s+2 characters are required by each element.

For further details of the character requirements of output formats see 2.1.3.2 of the 503 of the Technical Manual;

STORE USED

About 260 extra locations.

METHOD OF USE

These procedures are on a separate tape from the main package. When these procedures are required their tape should be input immediately after the main package.

CONFIGURATION

The Algol Matrix Package plus these two procedures will fill the available store during translation and hence facilities for storing the owncode version on Core Backing Store are desirable but not essential.

TAPE

A mnemonic tape is provided. An H Halt Code (76) is punched at the end.

R. V. Wood. *April*, 1964.

Example using mxoutput with the format set to freepoint (5).

ROW/COL	1	2	3	4	5	6	7
1	.00000	14.000	21.000	28.000	35.000	42.000	49.000
2	.00000	.00000	21.000	28.000	35.000	42.000	49.000
3	.00000	.00000	.00000	28.000	35.000	42.000	49.000

ROW/COL	8	9	10	11	12	13	14
1	56.000	63.000	70.000	77.000	84.000	91.000	98.000
2	56.000	63.000	70.000	77.000	84.000	91.000	98.000
3	56.000	63.000	70.000	77.000	84.000	91.000	98.000

ROW/COL	15	16	17	18	19	20
1	105.00	112.00	119.00	126.00	133.00	140.00
2	105.00	112.00	119.00	126.00	133.00	140.00
3	105.00	112.00	119.00	126.00	133.00	140.00

END OF MATRIX

CHAPTER 3 : BESSEL FUNCTIONS OF INTEGRAL ORDER
(Autocode Subroutine)

CODE BES 01C

FUNCTION

To calculate $J_n(x)$. The value of $Y_n(x)$ may be made available at the same time, if required.

CONFIGURATION

The requirements are as for 503 Autocode.

SETTING AND OTHER REQUIREMENTS

Integer variables	N(2)
Floating–point variables	I(2), O(2), B(3)
Functions	INT, LOG
Reference numbers used	1 to 18 inclusive
Storage required	about 220 locations, not including INT and LOG

If the main program using BES 01C as a subroutine employs any integer from 1 to 18 inclusive as a reference number, the reference numbering of BES 01C must be altered. The first instruction of BES 01C has been assigned the reference number 18, and all positive integers below are used elsewhere in the subroutine.

TAPES

The library tape is punched in the mnemonic code of the 503 Autocode. A tape is provided.

METHOD OF USE

Before entering BES 01C to calculate $J_n(x)$, the main program must set I1 to x and N to n. The value of $J_n(x)$ will be found in location O1 on exit.

If $Y_n(x)$ is also required, a non–zero marker should be set in I2; O2 will then be set to $Y_n(x)$ on exit. If $Y_n(x)$ is not required, I2 should be cleared; I2 and O2 will be clear on exit.

Entry is by means of the instruction SUBR 18 unless the reference numbering on the library tape has been altered.

ERROR EXITS

Negative Input

is displayed and control is transferred to RAP, if either n or x is negative.

If $Y_n(x)$ is required, and the given values of n and x are such that $Y_n(x)$ cannot be held as a standard floating–point number,

Overflow

is displayed, and control is transferred to RAP.

TIME

The following formulae give a very approximate idea of the time involved in calculating $J_n(x)$ and $Y_n(x)$:

If $J_n(x)$ only is required, about $(4 + 3k) / 5$ milliseconds.

If $Y_n(x)$ is also required, the total time is about k milliseconds,

where if $x<10$ and $n<x$ then $k = 2x + 9$

$x<10$ and $n>x$ then $k = 2x + 9 + n$

$x>10$ and $n<x$ then $k = 1.05x + 25$

$x>10$ and $n>x$ then $k = 1.05x + 25 + n$

ACCURACY

Results are correct to 8 decimal places.

PROCESS USED

k is chosen so large that ϵ in the expression

$J_{k+1} : J_k = \epsilon : 1$ may be neglected. With $S(k + 1) = 0$,

$S(k) = 10^{-10}$, the sequence S(p) is generated downwards from the expression $S(p-1) = 2p\, S(p)x^{-1} - S(p + 1)$, $(1 \leqslant p \leqslant k)$.

$J_n(x)$ is then taken as $\dfrac{S(n)}{S(0) + 2 \sum_{p=1}^{k} S(p)}$

$Y_n(x)$ is calculated from the expression

$$Y_{p+1} = \frac{2p\,Y_p}{x - Y_{p-1}}$$

$Y_o(x)$ and $Y_1(x)$ being obtained from the calculated values of $J_o(x)$ and $J_1(x)$.

S.M. Quinn and P. Schwar. *November, 1963*

CHAPTER 4: GAUSSIAN PROBABILITY INTEGRAL

CODE Gauss A

STORE USED

93 locations

CONFIGURATION

The requirements are as for 503 Algol.

ACCURACY

The results are accurate to 8 significant figures.

TIME

Less than 5 milliseconds in the worst case.

TAPE

A tape is provided.

```
real procedure Gauss (x); value x; real x;
comment This procedure, which is based on 803 B 109, calculates
        the integral from -infinity to x of
                exp (-0.5*x↑2)dx/sqrt(2*pi);

begin real y, z, w;
        if x=0 then z:=0
        else
        begin y:=abs(x)/2;
              if y > 3 then z:=1
              else if y < 1 then
              begin w:=y*y;
                    z:=(((((((0.000124818987*w
                          -0.001075204047)*w
                          +0.005198775019)*w
                          -0.019198292004)*w
                          +0.059054035642)*w
                          -0.151968751364)*w
                          +0.319152932694)*w
                          -0.531923007300)*w
                          +0.797884560593)*y*2
        end
        else
```

```
        begin y: = y − 2;
              z: = ((((((((((((−0.000045255559*y
              +0.000152529290)*y
              −0.000019538132)*y
              −0.000676904986)*y
              +0.001390604284)*y
              −0.000794620820)*y
              −0.002034254874)*y
              +0.006549791214)*y
              −0.010557625006)*y
              +0.011630447319)*y
              −0.009279453341)*y
              +0.005353579108)*y
              −0.002141268741)*y
              +0.000535310849)*y
              +0.999936657524
           end
     end
     Gauss : = if x> 0 then ( z + 1 ) / 2 else ( 1 − z ) / 2
end Gauss ;
```

D. Ibbetson. *November, 1963.*

CHAPTER 5 : ALGOL 5 TO 8

CODE ALGOL 5S

FUNCTION

To produce an eight hole ALGOL tape from a five hole tape.

STORE USED

1590 locations.

METHOD OF USE

Running Instructions

1) Translate the SAC program in the normal manner.

2) Place the 5 hole Algol tape in reader 1 and type ALGOL 5 to 8.

3) If the symbol **$** is encountered outside a string or a title the program waits until the sign of the word generator is changed. If the B–digit is set when the sign digit is changed, the stop code (76) is output before translation continues.

4) After converting a program (or detecting an error) the routine returns to RAP.

Error Indications

If the program detects an error, normal output is suspended, the next 64 non–blank characters are output (in 8–hole code) on the output writer and control returns to RAP (see (4) above).

N.B. The majority of syntactic errors will pass undetected.

Restriction

1) If a parameter delimiter in a standard function is a letter string containing a basic word it will be incorrectly converted.

For example the parameter delimiter

) STEP : (

will be converted to

) step : (

if it occurs in a call of a standard procedure.

2) If a formal parameter of a procedure, which is specified to be an array, has the same name as a previously declared procedure, then the latter procedure will be given square brackets in all calls until the end of the block in which the procedure with the offending array identifier is declared.

3) % followed by any other character is treated as though it were % 1.

FORMAT OF RESULTS

The eight hole ALGOL tape will be produced by punch 1. Brackets, basic words and operators such as ⩽ and ↑ are correctly produced. If an error is detected the next 64 non–blank characters will be printed on the output writer (see **Error Indications**).

CONFIGURATION

This program uses tape reader 1, tape punch 1 and the output writer.

TIME

The new tape is produced at the full speed of the output punch.

TAPE

The program is written in SAC and the tape is provided.

PROCESS USED

The following information is stored in locations STACK + 1000 downwards:–

Zero	for each begin
40 0 : 00 89	for each [
40 0 / 00 9	for each (appearing outermost in a procedure declaration call.

40 0 : 00 9	for each (otherwise
Array and switch identifiers	Store as left justified strings of six or less characters.
Procedure identifiers	Similarly and with a marker in bit 38.

An identifier which is followed by a left bracket is a procedure, an array, or a switch. The stack is scanned to find out which it is.

A right bracket is checked to see if it is a parameter delimiter. If so the following letter string is copied.

Any identifier which contains letters only is tested to see whether it corresponds to any 5-hole basic word. A binary search is used and special actions performed if necessary.

When the % symbol is encountered all characters up to and including the next semicolon are ignored.

J. Blake and D. Pullin. *November, 1963.*

CHAPTER 6: AUTOCODE 5 TO 8 Issue 2

CODE ACconv Issue 2 S

FUNCTION

To produce an 8-hole Autocode mnemonic tape from a 5-hole tape, with an optional copy on punch 2 and/or a print up on the line printer.

STORE USED

About 300 locations.

METHOD OF USE

Running Instructions

1) Translate the SAC program in the normal manner.

2) Place the 5 channel program tape in reader 1. An 8-hole copy is always produced on punch 1.
   (a) for an additional copy on punch 2, depress key 2.
   (b) for a print up on the line printer, depress key 1.

3) To treat the characters preceding the first shift character as on:—
   (a) letter shift, type ACconv; 1.
   (b) figure shift, type ACconv; 2.

4) Output is on punch 1, and on the additional devices specified by keys 1 and 2 (see para. 2 above).

5) The program must be stopped manually on the trailing blanks.

6) To check the 8-hole tapes produced, place each tape in reader 1 in turn and enter ACconv; 3.

Error Indications

1) If the line printer is unavailable,
   (a) LPMANL is displayed if it is in the manual state.
   (b) LPERR2 is displayed if it is in error 2 condition.

2) If the sumcheck fails, SUMERR is displayed.

FORMAT OF RESULTS

Except for the items below, the 5-hole characters are copied literally into 8-hole characters. (See under METHOD OF USE (3) for characters preceding the first shift character).

803 5-hole telecode	503 8-hole telecode
,	↑
@	[
cr	(neglected)
lf	new line

TIME

The new tape is produced at the full speed of the output punch.

TAPE A SAC tape is provided.

RESTRICTIONS

OUTPUT instructions, the interpretation of the results of INPUT instructions, the contents of TITLE instructions and 74 instructions in machine-code blocks may need modification. Dynamic stops in machine-code blocks should be replaced by jumps to RAP. For further information reference should be made to the specifications of the two Autocodes.

PROCESS USED

A look-up table is used; the 8-hole character value obtained depends on the 5-hole character value and the last shift read.

M. Beat *June, 1965*

CHAPTER 7 : LOGARITHM
(Subroutine)

CODE log S

FUNCTION

To evaluate $\log_e A$ where A is the standard floating–point number in the accumulator, satisfying A>0.

STORE USED

47 locations

METHOD OF USE

1) Global Declarations

Block name: log

2) Entry and Exit

Entry: SUBR, log

Enter with A in standard floating–point form in the accumulator.

Exit: $\log_e A$ is in the accumulator on exit.

3) Error Indication

If, on entry, A≤0. 'log error' is displayed and control is transferred to RAP to await a message.

TIME

The time taken is approximately 760 microseconds.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

Let $A = a.2^b$, and let $\frac{1}{2}a = x$ and $b - 255 = m$

Then the program uses

$$Z = \frac{x - \sqrt{2}/4}{x\,(3-2\sqrt{2}) + (3\sqrt{2}-4) / 4}$$

in evaluating

$$H = \sum_{r=0}^{4} B_{2r+1} Z^{2r+1}$$

where

$$B_9 = 8415 \times 2^{-38}$$

$$B_7 = 343173 \times 2^{-38}$$

$$B_5 = 16347428 \times 2^{-38}$$

$$B_3 = 925538404 \times 2^{-38}$$

$$B_1 = 94323185678 \times 2^{-38}$$

and forms $\log_e A = m \log_e 2 - \frac{3}{2} \log_e 2 + H$

November, 1963.

CHAPTER 8 : EXPONENTIAL
(Subroutine)

CODE exp S

FUNCTION

To evaluate e^A where A is the standard floating–point number in the accumulator, and satisfies the condition :

$A \leqslant 254 \log_e 2$ (176.059383)

STORE USED

58 locations.

METHOD OF USE

1) **Global Declarations**

Block name : exp

2) **Entry and Exit**

Entry : SUBR, exp

Enter with A in standard floating–point form in the accumulator.

Exit : e^A is in the accumulator on exit.

3) **Error Indication**

If $A > 254 \log_e 2$, 'exp error' is displayed on a new line and control is transferred to RAP to await a message.

ACCURACY

Answers are accurate to 8 significant figures.

2. 2. 3. 8.
exp S

TIME

The time taken is approximately 1250 microseconds.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

Let $x = y.2^b$ where y and b are the binary mantissa and exponent of the automatic floating–point notation.

exp evaluates

$$e^{-|y|} = \sum_{n=0}^{n=8} a_n y^n$$

where $a_0, a_1, \ldots, a_8$ are given constants, and squares the result b times. Then if $x < 0$, the reciprocal is taken.

Special notice is taken of the following cases:

$x = 0$	–	answer taken to be 1
$x < -254 \log_e 2$	–	answer taken to be 0
$b < -38$	–	answer taken to be 1

November, 1963.

CHAPTER 9 : SINE COSINE OR TANGENT
(Subroutine)

CODE trig S

FUNCTION

To find the sine, cosine or tangent of πA where A is the standard floating–point number in the accumulator, satisfying $|A| < 2^{28}$.

STORE USED

64 locations

METHOD OF USE

1) **Global Declarations**

(i) Block name and global labels : trig (sin, cos, tan)

(ii) Global data : cos 1 * trig

2) **Entry and Exit**

Entry: Enter with A in the accumulator in standard floating–point form by instruction

SUBR, sin * trig to find sin πA

SUBR, cos * trig to find cos πA

SUBR, tan * trig to find tan πA

Exit: On exit, the required result is in the accumulator, and cos πA is in location cos 1*trig

3) **Error Indication**

If $|A| \geqslant 2^{28}$, 'trig error' is displayed and control is transferred to RAP to await a message.

TIME AND ACCURACY

The time taken for sine or cosine is 880 microseconds. For tangent it is 980 microseconds. Answers are accurate to eight significant figures.

TAPE

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

For $-2^{-15} \leqslant x < 2^{-15}$ the program puts $\sin \pi x = \pi x$ and $\cos \pi x = 1$. For all other values, the following process is used.

Let $x = \frac{1}{2}n + \frac{1}{2}y$, where n is an integer and $-\frac{1}{2} \leqslant y < \frac{1}{2}$.

n', the 2 least significant binary digits of n, is found and then trig computes

$$z = \tan \frac{\pi y}{4} = \frac{4y}{4 - y^2} P(y^2)$$

where P is a power series which converges rapidly for $y^2 \leqslant \frac{1}{4}$

Then if $S = \frac{1}{2} \sin \frac{\pi y}{2} = \frac{z}{1 + z^2}$ and

$$C = \frac{1}{2} \cos \frac{\pi y}{2} = \frac{1}{2} \frac{1 - z^2}{1 + z^2}$$

$\frac{1}{2} \sin \pi x$ and $\frac{1}{2} \cos \pi x$ are chosen according to the following table.

n'	$\frac{1}{2} \sin \pi x$	$\frac{1}{2} \cos \pi x$
0	S	C
1	C	-S
2	-S	-C
3	-C	S

The necessary steps are then taken to standardise these two results and obtain the final required results from them.

November, 1963.

CHAPTER 10 : ARCTANGENT
(Subroutine)

CODE arctan S

FUNCTION

To evaluate $\frac{1}{\pi}$ arctan A, where A is the standard floating–point number in the accumulator.

STORE USED

43 locations.

METHOD OF USE

1) Global Declarations

Block name : arctan

2) Entry and Exit

Entry: Enter with A in standard floating–point form [in the accumulator by the instruction SUBR, arctan.]

Exit: $\frac{1}{\pi}$ arctan A is in the accumulator on exit.

TIME

The time taken is 1500 microseconds.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

(i) If $|x| < 1$ put $u = x$

Otherwise put $u = \frac{1}{x}$

(ii) Find $\frac{1}{\pi}$ arctan u by means of a Chebyshev power series.

(iii) If $|x| < 1$, exit with $\frac{1}{\pi}$ arctan u

If $x \geqslant 1$, exit with $-\frac{1}{2} - \frac{1}{\pi}$ arctan u

If $x \leqslant -1$, exit with $\frac{1}{2} - \frac{1}{\pi}$ arctan u

November, 1963.

CHAPTER 11 : SQUARE ROOT
(Subroutine)

CODE sqrt S

FUNCTION

To find the square root of the standard floating–point number, A, in the accumulator.

STORE USED

33 locations

METHOD OF USE

1) **Global Declarations**

Block name : sqrt

2) **Entry and Exit**

Entry: Enter with A in standard floating–point form in the accumulator by the instruction

SUBR, sqrt.

Exit: On exit the required result is in the accumulator.

3) **Error Indication**

If A < 0, 'sqrt error' is displayed and control is transferred to RAP to await a message.

TIME

The time taken is 450 microseconds.

ACCURACY

The results are accurate to 8 significant figures.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

sqrt separates the mantissa and exponent and adjusts them to have values a and b respectively such that

$$A = a \times 2^b, \qquad \text{where } \tfrac{1}{4} \leqslant a < 1, \qquad \text{and } b \text{ is even.}$$

$\sqrt{a}$ is then found by performing

$$B_{n+1} = \tfrac{1}{2}\left(B_n + \frac{a}{B_n}\right)$$

$$\text{using } B_0 = \frac{a}{2} + .4368$$

and taking B_3 as a satisfactory value.

Then mantissa B_3 is combined with exponent $\frac{b}{2}$ to give the required standard floating-point result $\sqrt{A}$

The case $A = 0$ is treated separately.

November, 1963.

CHAPTER 12 : ERROR AND PROBABILITY FUNCTIONS
(Subroutine)

CODE approx S

FUNCTION

To calculate $H(|x|) = \frac{2}{\sqrt{\pi}} \int_0^{|x|} e^{-t^2}\, dt$, or

$P(|x|) = \frac{1}{\sqrt{2\pi}} \int_{-|x|}^{|x|} e^{\frac{-t^2}{2}}\, dt$, where x is the standard floating–point number in the accumulator on entry.

STORE USED

62 locations.

METHOD OF USE

1) Global Declarations

Block name and global labels: approx (Hx, Px)

2 2) Entry and Exit

Entry: SUBR, Hx * approx to find H(|x|)
SUBR, Px * approx to find P(|x|)
In both cases, enter with x in standard floating–point form in the accumulator.

Exit: On exit the required result is in the accumulator.

ACCURACY

The results are accurate to 8 significant figures.

2. 2. 3. 12.
approx S

TIME

The time taken varies according to the magnitude of x:

Range of x in H(|x|)	Range of x in P(|x|)	Time taken in microseconds
$x = 0$	$x = 0$	50
$0 < x < \sqrt{2}$	$0 < x < 2$	1050
$\sqrt{2} < x < 3\sqrt{2}$	$2 \leqslant x < 6$	1650
$x \geqslant 3\sqrt{2}$	$x \geqslant 6$	250

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

P(|x|) is obtained as follows:

P(0) is set equal to zero.

In the range $0<|x|<2$, the power series for $\frac{1}{|x|} P(|x|)$ has been economised, using Chebyshev polynomials and a series of nine terms in x^2 obtained. This is evaluated and multiplied by $|x|$.

In the range $2<|x|<6$, a polynomial of 15 terms similarly obtained is used to obtain P(|x|).

For $|x| \geqslant 6$, P(|x|) is set equal to unity.

H(|x|) is obtained using the identity $H(|x|) = P(\sqrt{2|x|})$

ACKNOWLEDGEMENT

The National Bureau of Standards Tables of Probability Functions, Vol.II (New York 1942) were used for checking purposes.

Dr. A.M. Murray.
University of Aberdeen.

November, 1963.

CHAPTER 13 : SINGLE WORD SORT
(Subroutine)

CODE swsort S

FUNCTION

To sort into ascending sequence any number of single–word items (numbers) located sequentially anywhere in the store.

A separate entry checks that the sequence is correct.

The data is sorted within the area it occupies.

STORE USED

39 locations.

METHOD OF USE

1) Global Declarations

(i) Block names and global labels: swsort (sort, check, B1)

(ii) Global data: ws * swsort

2) Entry and Exit

Entry: To sort : SUBR, sort * swsort with 2
words of parameters after the entry instruction.

To check: SUBR, check * swsort with 2
words of parameters after the entry instruction.

Exit: If the check finds an item out of sequence the accumulator will be negative on exit, and the address of the offending item is found by adding the address part of the second instruction in location B1 * swsort to the content of ws*swsort.

3) Restrictions

The numerical difference between any two words compared must be less than one. This is assured if all the numbers are non-negative. Failure to observe this restriction will cause an error, and set the overflow indicator.

A program using swsort as a subroutine cannot be entered more than once because an order in swsort is overwritten during the running of the program. Therefore the program using this subroutine should be assembled with key 35 depressed. In this case RAP does not form an internal checksum of the program when it is entered. Alternatively the subroutine could be made into a common program. See P6

4) Parameters

N : the address of the location containing the number of items

A : the address of the first item

A + C(N) $\leqslant$ the number of locations in the computer's store.

These parameters should be placed, in the sequence shown, in the two locations immediately following the entry instructions.

TIME

Time taken depends in a complicated fashion on N and on the number of ones in the binary expression of N. The following times are a guide.

1023	numbers	.75 secs
2047	numbers	1.78 secs
3999	numbers	4.17 secs

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

A sift sort with varying interval of comparison and exchange. The method is described in "A High-speed Sorting Procedure" by D.L. Shell in Communications of the A.C.M., Vol. 2, No. 7 of July, 1959.

November, 1963.

CHAPTER 14 : GENERAL SORT
(Subroutine)

CODE gensor S

FUNCTION

To sort into ascending sequence of keys, any number of items located consecutively anywhere in the store. Each item may be of any fixed length, its key may be of any fixed number of consecutive words, and the first word of the key may be located anywhere within the item.

A separate entry checks that the sequence is correct.

The data is sorted within the area it occupies.

STORE USED

69 locations.

METHOD OF USE

1) Global Declarations

(i) Block names and global labels: gensort (sort, check, A2)

(ii) Global data : S*gensort

2) Entry

To sort : SUBR, sort*gensort followed by one indirect parameter.

To check : SUBR, check*gensort followed by one indirect parameter.

3) Exit

If the check finds an item out of sequence the accumulator will be negative on exit, and the address of the first word of the lowest offending item is found by adding the content of location S*gensort to the address part of the second instruction in location A2*gensort.

4) Restrictions

The numerical difference between any two words of the keys which are to be compared must be less than one. This is assured if all the words of every key are non-negative. If this restriction is not observed an error will arise, and the overflow indicator will be set on exit.

A program using gensort as a subroutine cannot be entered more than once because an order in gensort is overwritten during the running of the program. Therefore the program using this subroutine should be assembled with key 35 depressed. In this case RAP does not form an internal checksum of the program when it is entered. Alternatively the subroutine could be made into a common program.

5) Parameters

The word which follows the entry instructions must be the address of the location holding N. The five parameters listed below must be placed in any five consecutive locations, in the sequence given.

N : the number of items

A : the address of the first word of the first item

I : the number of words in each item

K : the number of consecutive words in the key

F : the position of the first word of the key within the item, using the convention that F = 0 indicates that the first word of the key coincides with the first word of the item.

The following inequalities define the ranges of the parameters :

$A + I.N \leqslant$ the number of locations in the computer's store

$1 \leqslant K \leqslant I$.

$0 \leqslant F \leqslant I-K$

TIME

The time taken depends in a complicated fashion on I, N and on the number of ones in the binary expansion of N. The following times are a guide.

N	300	600	900	1200	1500	1800
I	sec.	sec.	sec.	sec.	sec.	sec.
2	.314	.714	1.55	1.87	2.29	3.37
4	.428	1.05	2.10	–	–	–
6	.65	1.37	–	–	–	–

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

A sift sort with varying interval of comparison and exchange. The method is described in "A High Speed Sorting Procedure" by D.L. Shell in Communications of the A.C.M., Vol. 2, No. 7 of July, 1959.

November, 1963.

CHAPTER 15 : SINGLE WORD, MULTIPLE KEY FIELD SORT
(Subroutine)

CODE multke S

FUNCTION

To sort into ascending or descending order of keys, any number of single word items located sequentially anywhere in the store. The key field may be any group of consecutive bits in the word, (see NOTE below).

Those consecutive sets of the above items which were not differentiated by the previous sort may now be further sorted on some additional key field. This process may be repeated for any number of key fields.

STORE USED

54 locations.

METHOD OF USE

1) Global Declarations

(i) Block name and global labels : multkey (M)

(ii) Global data : ws*multkey, ws*multkey +6

2) Entry and Exit

SUBR, multkey with 2K + 2 parameter words following the entry instruction, (K being the number of key fields). The content of the accumulator is destroyed.

Exit is to the first location following those holding the parameters. The content of this location must not be zero, nor -1×2^{-38}.

On exit the accumulator contains the contents of location (2K + 3) + 1.

3) Restriction

A program using multkey as a subroutine cannot be entered more than once because an order in multkey is overwritten during the running of the program. Therefore the program using this subroutine should be assembled with key 35 depressed. In this case RAP does not form an internal checksum of the program when it is entered. Alternatively the subroutine could be made into a common program. see P 8

4) Parameters

The following parameters must be placed in the 2K + 2 locations following the entry instructions, which are assumed to be in location E. Thus:

In location	place the parameter	which specifies
E + 1	n	address of first item
E + 2	N	address of last item
E + 3	d_1	direction of first sort
E + 4	K_1	collating constant for first key field
E + 5	d_2	direction of second sort
E + 6	K_2	collating constant for second key field
.	.	..
.	.	.
E + 1 + 2i	d_i	direction of ith sort
.	.	.

where n, N, K_1, K_2, ... are all integers and

$d_i = 0$ if items are to be sorted in ascending sequence for this key field.

$d_i = -1 \times 2^{-38}$ if items are to be sorted in descending sequence for this key field.

If d_i takes neither of the above values, the routine will exit to the word containing d_i, i.e. address E + 1 + 2i.

5) Note

The sign digit may be one of the digits of a key field if desired. The consecutive bits forming this key field will nevertheless be treated as a positive binary integer.

Thus, for example, one may sort on the F1 functions of a series of instructions.

6) Modification

To sort on a new key field within categories already differentiated by some other key field(s), set a collating constant for the previous key field(s) in location ws*multkey, before entry, and modify multkey in the following way:

M*multkey 20 ws+6*multkey : 00 0

(The contents of location M*multkey on the library tape are

20 ws + 6*multkey : 26 ws* multkey)

The constant set in ws*multkey, is disturbed on completion of that sort.

TIME

Time taken is approximately 143 x I x b milliseconds where I = number of items to be sorted and b = total number of bits in all the key fields.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

On any one scan through the items, only one bit is examined, to determine whether an interchange is required. Previously examined bits are also tested so that the results of earlier scans are not disturbed.

J. W. J. Williams *November, 1963*

CHAPTER 16 : POLYNOMIAL INTERPOLATION AND EXTRAPOLATION
(Subroutine)

CODE polint S

FUNCTION

To obtain an approximation to the value of a function f(x) for any given x by fitting a polynomial of specified degree to values of $f(x_i)$ tabulated at equal intervals of the argument. The values of x_i used are chosen to be as nearly as possible symmetrical about x except that when this would imply the use of values outside the range of the table, values at the end of the table are used.

STORE USED

80 + A(n) locations.

METHOD OF USE

1) Global Declarations

(i) block name : polint

(ii) global data : address * polint, A(n)
where A(n) is a block of n locations allocated to hold the table of given values.

2) Entry and Exit

Entry: Place the address of the table, A, in location address * polint and place the standard floating–point number x in the accumulator, and enter by SUBR, polint. The contents of locations address * polint and A(n) are not disturbed and need not be set again before subsequent entries.

Exit: On exit f(x) is in the accumulator in standard floating–point form.

3) Table

The table of $f(x_i)$ may be placed anywhere in the store.
The table must be of the following form:

Location	Contents	
A	n	integers
A + 1	N	integers
A + 2	x_o	Standard floating–point numbers
A + 3	h	Standard floating–point numbers
A + 4	$f(x_o)$	Standard floating–point numbers
A + 5	$f(x_1)$	Standard floating–point numbers
. . . . .	. . . . .	Standard floating–point numbers
A + 3 + N	$f(x_{N-1})$	Standard floating–point numbers

where n is a positive integer or zero such that n + 1 is the degree of the interpolating polynomial and n+ 2 is the number of values of the function used.

N is an integer specifying the number of values of $f(x_i)$ in the table.

x_o is the first argument at which the function is tabulated.

h is the interval in x_i, i.e. $h = x_{i+1} - x_i$

4) Error Indication

Two new lines are output and

polint error

is displayed if n is negative (attempt to fit polynomial of negative or zero degree)

or if $N < n + 2$ (attempt to use more values of the function than are in the table)

and control is transferred to RAP to await a message.

ACCURACY

In general a polynomial of degree m gives a good approximation to a tabulated function only if all difference of order m + 1 are small. See PROCESS USED.

Otherwise the method is exact within the limits imposed by the rounding errors inherent in floating–point arithmetic.

TIME

The maximum time taken is $.75+.5\,n + .12\,n^2$ milliseconds.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

Neville's iteration method. See "Numerical Analysis" by Z. Kopal for a description of the method and a discussion of the errors which arise when higher differences are not small.

r is defined as the integral part of $\frac{x - x_o}{h} - \frac{n}{2}$ except that if this quantity is negative, r is set equal to zero, and if it exceeds N–n–2, r is set equal to N–n–2. The n + 2 values of $f(x_i)$ with $i = r, r + 1, \ldots\ldots\ldots r + n + 1$ are used to obtain f(x).

The basic iteration formula is

$$f_{j+1}(x_i) = \begin{vmatrix} f_j(x_i) & x_i - x \\ f_j(x_{i+1}) & x_{i+j+1} - x \end{vmatrix} \div (x_{i+j+1} - x_i)$$

where $i = r, r + 1, \ldots\ldots\ldots, r + n + 1 - j$ and $f_o(x_i) = f(x_i)$.

Then $f(x) = f_{n+1}(x_R)$.

The above formula can be written in the form

$$f_{j+1}(x_i) = f_j(x_i) + \{ f_j(x_{i+1}) - f_j(x_i) \} \{ x - x_i \} \div \{ x_{i+j+1} - x_i \}$$

$$= f_j(x_i) + \{ f_j(x_{i+1}) - f_j(x_i) \} \{ x - x_i \} \div (j + 1)\, h$$

and it is this last relation which is used in the subroutine.

Dr. A.M. Murray.
University of Aberdeen.

November, 1963.

CHAPTER 17 : INTEGRATION OF A SET OF SIMULTANEOUS FIRST ORDER. DIFFERENTIAL EQUATIONS
(Subroutine)

CODE intdif S

FUNCTION

To perform one step of the step–by–step integration of the n simultaneous first order differential equations.

$$\frac{dy_i}{dx} = F(x, y_1, y_2, \ldots, y_n), \ (i = 1, 2, \ldots, n),$$

using the Runge – Kutta – Gill method.

$n \geqslant 1$, so intdiff may be employed to integrate a single equation. The upper bound of n is determined by store size.

STORE USED

53 + Y(n) + K(n) + G(n) + aux locations.

METHOD OF USE

1) Global Declarations

(i) block name : intdiff, aux

(ii) global data : par * intdiff, Y(n), K(n), G(n)

2) Notation

n The number of equations to be integrated. (See paragraph 3)

h The integration step

x The independent variable

i A suffix

y_i The dependent variables

k_i hF_i

g_i An intermediate result produced in one step and used in the next step (if there is a next step). Initially the values of all g_i must be zero.

Y_i The address of the location holding y_i

K_i The address of the location holding k_i

G_i The address of the location holding g_i

h, x, y_i, k_i and g_i are standard floating-point numbers, while n, i, Y_i, K_i and G_i are fixed-point integers.

3) Auxiliary Subroutine

(i) In order to define the function he wishes to be integrated, the user must write an auxiliary subroutine which will calculate, on each entry,

$$k_i = hF_i(x, y_1, \ldots, y_n), (i = 1, 2, \ldots, n),$$

and place the results k_i in locations K_i.

Enter the subroutine by the instruction SUBR, aux (where intdiff assumes that the name given to the subroutine is aux).

(ii) This auxiliary subroutine is entered four times from intdiff each time intdiff is entered from the main program.

(iii) If x occurs explicitly in any F_i then the values of x to be used by the auxiliary subroutine on the four occasions on which it is entered are x_o, $(x_o + \frac{1}{2}h)$, $(x_o + \frac{1}{2}h)$, $(x_o + h)$, respectively, where x_o is the initial value of x. This can be achieved by arranging that the value of x shall be increased by $\frac{1}{2}h$ immediately before exit from the auxiliary subroutine for the 1st, 3rd, 5th, ... time.

(iv) An alternative way to do this is to set n one greater than the number of dependent variables and make the additional variable, y_n so formed, satisfy $y_n = x$, and then use y_n instead of x when calculating F_i. The increments to the value of y_n are made by intdiff automatically.

To make $y_n = x$, set its initial value, then set $k_n = h$ by the main program before entering intdiff and let the auxiliary subroutine evaluate

$$K_i = hF_i\,(y_n, y_1\, y_2, \ldots, y_{n-1})$$

for $(i = 1, 2, \ldots, n-1)$ only.

If this is done and the value of h is altered, the value of k_n must also be altered by the main program.

(v) The auxiliary subroutine forms one of the blocks of the main program and its name must be included in the block declaration.

4) Entry and Exit

Entry for the first step of an integration:

(i) Place the parameter $-(n-1)$ in location par*intdiff.

(ii) If using the procedure of paragraph 3(iv), place k_n in location K + n.

(iii) Set the initial values of y_i in locations Y + i and clear all locations G + i.

(iv) Enter intdiff by the instruction SUBR, intdiff.

Exit after any step:

(v) On exit, the incremented values of y_i will be in location Y_i and the error quantities g_i will be in locations G_i, using the same notation as in 'PROCESS USED'

$$C\,(Y_i) = y_{4i}$$

$$C\,(G_i) = g_{4i}$$

(vi) The contents of locations Y_i and locations G_i must not be altered if it is intended to re-enter for another step, but the contents of locations K_i are of no further use and these locations may be over-written if desired.

Re–entry for a further step of an integration:

(vii) If necessary, adjust the settings mentioned in paragraphs 4(i) and 4(ii).

(viii) Make a standard entry to intdiff.

5) Restriction

A program using intiff as a subroutine cannot be entered more than once because an order in intdiff is overwritten during the running of the program. Therefore the program using this subroutine should be assembled with key 35 depressed. In this case RAP does not form an internal checksum of the program when it is entered. Alternatively the subroutine could be made into a common program.

TIME

If A is the time taken (in milliseconds) by the auxiliary subroutine, the total time taken on each entry is :–

$$0.3 + 4A + 1.4n \text{ milliseconds}$$

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

The theory of the process is discussed by S. Gill (Proc. Cambridge Phil. Soc. Vol. 47, p. 96, 1951)..

In the following description, first suffices denote the stage of the calculation required for one step of an integration. That is to say y_{oi} is the initial value of y_i, while g_{4i} is the fourth and final calculated value of g_i. It should be noted that g_{oi} of the first step of an integration is zero, while g_{oi} and y_{oi} of a subsequent step are identical with g_{4i} and y_{4i} of the preceding step.

Stage 1 $\quad k_{1i} = hF_i\ (x, y_{o1}, \ldots, y_{on})\ \ (i = 1, \ldots, n)$

$$r_{1i} = \tfrac{1}{2}\,(k_{1i} - g_{oi})$$

$$y_{1i} = y_{oi} + r_{1i}$$

$$g_{1i} = g_{oi} + 3r_{1i} - \tfrac{1}{2}k_{1i}$$

Stage 2 $\quad k_{2i} = hF_i\ (x + \tfrac{1}{2}h, y_{11}, \ldots, y_{1n})$

$$r_{2i} = (1 - \sqrt{\tfrac{1}{2}})\ (k_{2i} - g_{1i})$$

$$y_{2i} = y_{1i} + r_{2i}$$

$$g_{2i} = g_{1i} + 3r_{2i} - (-\sqrt{\tfrac{1}{2}})\,k_{2i}$$

Stage 3

$$k_{3i} = hF_i\,(x + \tfrac{1}{2}h,\ y_{21},\ \ldots,\ y_{2n})$$

$$r_{3i} = (1 + \sqrt{\tfrac{1}{2}})\,(k_{3i} - g_{2i})$$

$$y_{3i} = y_{2i} + r_{3i}$$

$$g_{3i} = g_{2i} + 3r_{3i} - (1 + \sqrt{\tfrac{1}{2}})\,k_{3i}$$

Stage 4

$$k_{4i} = hF_i\,(x + h,\ y_{31},\ \ldots,\ y_{3n})$$

$$r_{4i} = \tfrac{1}{3}(\tfrac{1}{2}k_{4i} - g_{3i})$$

$$y_{4i} = y_{3i} + r_{4i}$$

$$g_{4i} = g_{3i} + 3r_{4i} - \tfrac{1}{2}k_{4i}$$

S. E. Gilbert

November, 1963.

CHAPTER 18 : BESSEL FUNCTIONS J0, J1, Y0, Y1
(Subroutine)

CODE bessel S (Issue 2)

FUNCTION

To calculate $J_0(x)$, $J_1(x)$, $Y_0(x)$, $Y_1(x)$ where x is the standard floating–point number in the accumulator.

STORE USED

About 312 locations. This includes the subroutines log, sqrt and trig.

METHOD OF USE

This subroutine consists of two blocks, bessel and polyn, and also requires log, sqrt and trig.

1) **Global Declarations**

(i) Block names and global labels: bessel (J_0, J_1, Y_0, Y_1), log, sqrt, trig (sin, cos, tan), polyn.

(ii) Global data: cos 1 * trig, Jn * bessel, t2 * bessel, ws * polyn.

The block polyn is punched on the same tape as bessel, and is used as a subroutine by bessel. log, sqrt and trig must be assembled with bessel which uses them as subroutines.

2) **Entry**

SUBR, J_0 * bessel	to find $J_0(x)$
SUBR, J_1 * bessel	to find $J_1(x)$
SUBR, Y_0 * bessel	to find $Y_0(x)$
SUBR, Y_1 * bessel	to find $Y_1(x)$

3) **Exit**

The required result is in the accumulator on **exit.**

Where the required result is $Y_0(x)$ or $Y_1(x)$ and $x \leqslant 4$, $J_0(x)$ or $J_1(x)$ is to be found in location Jn * bessel.

4) Error Indication

'bessel error' is output on a new line on the output writer if $x<0$ on entry to find $Y_0(x)$ or $Y_1(x)$, and control is transferred to RAP to await a message.

5) Restriction

A program using bessel and polyn as subroutines cannot be entered more than once because an order in polyn is overwritten during the running of the program. Therefore the program using this subroutine should be assembled with key 35 depressed. In this case RAP does not form an internal checksum of the program when it is entered. Alternatively, the subroutine could be made into a common program consisting of two blocks, bessel and polyn.

TIME

To calculate $J_0(x)$ and $J_1(x)$, where $x \leqslant 4$, approximately 1.04 milliseconds.

To calculate $Y_0(x)$ and $Y_1(x)$, where $0 \leqslant x \leqslant 4$, approximately 3.0 milliseconds.

To calculate $J_0(x)$, $J_1(x)$, $Y_0(x)$ and $Y_1(x)$, where $x>4$, approximately 3.5 milliseconds.

TAPES

The library tape is punched for input by SAP. A tape is provided.

PROCESS USED

Summation of series.

Refer to Mathematical Tables – Aids to Computation Vol. 11, 1957, pages 86–88.

April, 1965

CHAPTER 19 : PTSREAD
(Common Program)

CODE PTSREA S

FUNCTION

To cause one or more numbers to be read from a specified input device.

1. To input a number to the accumulator, the mode of the number and the input device being specified by a parameter word; alternatively, to input a string of alphanumeric information and pack it, five characters to a word in a location specified by a parameter word.

2. To input a number or alphanumeric string on the same device as on the previous entry, and using the same parameters.

STORE USED

341 locations.

METHOD OF USE

This routine is based on the 503 Algol input procedures, (see PROCESS USED), and accepts numbers in any format which would be accepted by the Elliott Algol system (see 2.1.3.). In addition, it can be used to input fixed–point fractions.

Entry Points

Method 1. COMP, PTSREAD

W (W is the parameter word)

On exit from numeric read, the number will be in the accumulator.

On exit from alphanumeric read, the address of the first location after the packed words will be in the accumulator.

Method 2.

a) Floating Point	COMP, PTSREAD, 2
b) Fixed Point	COMP, PTSREAD, 3
c) Alphanumeric	COMP, PTSREAD, 4

In all cases the contents of the accumulator will be as in Method 1.

Parameters

The parameter word W

F1 N1 : 00 2048d

controls the input as follows:

F1 = 00 Floating–point input; N1 is not used; input on device d.

F1 = 20 Fixed-point input; if the number contains a decimal point it is assembled as a fixed-point fraction, otherwise as an integer. Fixed-point fractions are divided by $10^{(N1)}$. Input is on device d.

F1 = 40 Alphanumeric input on device d.

N1 is the address of the location which holds the address where the first packed word is to be stored. On exit, this location will be pointing to the first word after the packed information.

d = 0 The device used is reader 1.

d = 1 The device used is reader 2.

d = 2 The device used is the typewriter.

Error Indications

On detecting an impermissible combination of characters, READ ERROR will be displayed and the program will wait. When the sign of the word generator is changed the routine will exit.

The following conditions will cause an error:–

Two decimal points in a number.

Two $_{10}$'s in a number.

A decimal point following $_{10}$.

No digits after $_{10}$.

No digits after a decimal point.

No digits between signs.

Fixed–point number out of range.

£ read before + − .$_{10}$ or digit during input.

Digit read before £ during string input.

£ read in inner string.

TAPES

The program is written in SAP. A tape is provided. This should be translated as a common program (see 2.1.2.4.).

PROCESS USED

1) Numbers

Numbers dealt with by PTSREAD conform to the ALGOL definition of number; each number must be followed by some character other than a digit, i.e. a decimal point or a suffix $_{10}$.

2) Inner Strings

Interpreted layout characters may be inserted in inner strings, i.e. between an additional £ and ?

The meanings are	l	newline
	s	space
	r	runout (i.e. blank)
	t	tab
	h	halt

Each of these may be followed by an integer indicating the number of such characters required.

For example,

££16s10??

is a string consisting of six new lines followed by ten spaces.

3) Packing of Strings

Alphanumeric characters are packed five to a word, with the leading character at the more significant end. The sign bit is 1 for the last word of any string and 0 for all other words. If a string is not an exact multiple of 5, the final characters are left justified.

D. Pullin *January, 1964.*

CHAPTER 20 : PTSPRINT
(Common Program)

CODE PTSPRI S

FUNCTION

To cause one or more numbers to be printed on a specified output device.

1. To output the number from the accumulator in a mode and format specified by two parameter words, which also specify the output device used; alternatively, to output the string of packed characters, of which the address of the first word is in the accumulator.

2. To output the number in the accumulator, or the string whose starting address is in the accumulator, in the same mode as the last number of the same type, and on the same output device and in the same format as the last number of any type. Standard formats are provided and can be set up by a single entry to the package.

This routine is based on the 503 Algol output procedures and provides all the facilities of format and page layout that are available in the Elliott Algol system. In addition, it can be used to output fixed–point fractions.

The expected form of the information for output is as defined in PTSREA S (2.2.3.19.).

STORE USED

478 locations.

METHOD OF USE

Entry Points

Method 1. COMP, PTSPRINT
W 1
W 2

with the number or the starting address of the string in the accumulator.

Method 2.

a) Floating–point COMP, PTSPRINT, 2

b) Integer	COMP, PTSPRINT, 3
c) Fixed–point fraction	COMP, PTSPRINT, 4
d) String	COMP, PTSPRINT, 5
e) Presumed settings	COMP, PTSPRINT, 6

a), b) and c) will output the number from the accumulator; d) will output the string whose starting location is adjacent to the last location of the previously output string, and e) performs no output but sets up the following formats for subsequent parameterless entries:

Punch 1, newline, leadzero "space", grouping (0)

For floating–point	freepoint (8)
For integer	digits (8)
For fixed–point fractions	12 digits

PARAMETERS

The first parameter word, W1,

F1 N1 : 00 N2

controls the mode of printing as follows:–

F1	Mode
40	Scaled
20	Aligned
10	Freepoint
04	Fixed–point fraction
02	String
00	Integer

N1 = number of digits before the point.

N2 = total number of digits.

Note. When freepoint is used, N1 and N2 should be set equal.

For integers and fixed–point fractions, N1 is ignored..

The second parameter word, W2,

F1 N1 : F2 n+2048d

specifies the format of the output and the device to be used, as follows:–

a)

F1	Meaning	
00	Sign of number dealt with in usual way	
40	Special (1)	see 2. 1. 3. 2. Chapter 2. 5. 3.
20	Special (2)	
10	Special (3)	

b) F2 the six bits are treated as an integer (i) and the digits are grouped according to the rules for "grouping (i)". (see 2. 1. 3. 2. Chapter 2. 5. 1.

c) N1 is the address of the first word of the string that is to be prefixed to the number.

If N1 = 0 the prefix is "newline".

If N1 = 1 there is no prefix (i.e. "sameline").

d) The seven least significant bits of the character n are the binary equivalent of the character used to replace leading zeros. If n = 0 the leadzero character is a space.

e) Output occurs on device d where

d = 0 for tape punch 1

d = 1 for tape punch 2

d = 2 for the output writer.

Emergency Printout

If the number to be printed is too large for the requested format, an emergency routine is entered which will preserve the page layout. Provided that more than six characters have been requested the number will be printed in scaled (n–6) format; if not, a halt symbol and an H will be output.

If a real number is not in standard form, PRINT ERROR will be displayed.

If an impermissible character appears in an inner string, STRING ERROR will be displayed.

TAPES

The program is written in SAP. A tape is provided. This should be translated as a common program (see 2. 1. 2. 4.).

CHAPTER 21 : EDITALL

CODE EDITAL S

FUNCTION

To produce a modified copy of a five– or eight–hole tape (the input tape) by means of detection, insertion and replacement of strings of characters. The alterations are specified by means of an edit tape .

STORE USED

Program 318 locations

Workspace 201 locations

If the one reader mode is used (see METHOD OF USE) then the edit tape is packed with five 7–bit characters to a word and is stored in the free store between the program and its workspace.

METHOD OF USE

There are two methods of use, depending on whether one or two paper tape readers are available.

In the two reader mode, the input tape is in reader 1 and the edit tape in reader 2 during the editing process; in the one reader mode the edit tape is first read into the computer and stored, and is then 'read' from the store as required during the editing process. (See Note 2).

RUNNING INSTRUCTIONS

(n denotes the number of channels on the tape).

1. To translate EDIT ALL. depress KEY 35 and type SAP.
2. To enter EDIT ALL type EDITAL; 1.
3. Clear the keyboard.

One Reader Mode

4. Set up on the keyboard — 00 1 : 00 n

5. Load the edit tape in the reader and set 40 on the F2 keys. The edit tape is read in.

6. Load the input tape in the reader and change the sign of the word generator.

   The edited tape is output on punch 1. If the edit tape ends with an RE command (see Commands), the editing must be stopped by depressing the MANUAL button then pressing RESET.

7. To check the edited tape for punching errors — Type EDITALL; 2.

   The message ERRSUM indicates an error (see Checking the Edited Tape).

Two Reader Mode

4. Set up on the keyboard — 00 2 : 00 n

5. Load the input tape in reader 1, and the edit tape in reader 2.

   Change the sign of the word generator.

   The edited tape is output on punch 1. If the edit tape ends with an RE command (see Commands), the editing must be stopped by depressing the MANUAL button and then pressing RESET.

6. To check the edited tape for punching errors — Type EDITALL; 2.

   The message ERRSUM indicates an error (see Checking the Edited Tape).

Commands

Each command occupies a single line of a print–up of the edit tape. The first two non–ignorable characters on the line specify the function; the remaining characters up to, but not including, the next "new line", form the edit string . Lines containing no function characters are ignorable. (See Note 1).

FL (Find Line)

The edit string is read. Then the input tape is copied until a line beginning with this string is found. The last character copied is the last character of the edit string.

DL (Delete to Line)

The edit string is read. Then the input tape is skipped until a line beginning with this string is found. The last character skipped is the last character of the edit string.

FC (Find Characters successively)

If the characters of the edit string are C_1 C_2 C_3 . . . . C_n, then the input tape is copied until C_1 has been copied, and then further until C_2 has been copied, and so on until C_n has been copied.

DC (Delete to Characters successively)

If the characters of the edit string are C_1 C_2 C_3 . . . C_n, then the input tape is skipped until C_1 has been skipped, and then further until C_2 has been skipped, and so on until C_n has been skipped.

FE (Find End of Line)

The input tape is copied up to, but not including, the next "new line". The "new line" character is read and stored in a buffer, and is held while insertions, if any, are made from the edit tape. If the next command after the insertions is a "find" command, then the buffer character is output; if it is a "delete" command, the buffer character is ignored.

The edit string is ignored, and may be used for comments.

DE (Delete to End of line)

The input tape is skipped up to, but not including, the next "new line". The treatment of this "new line" and of the edit string is the same as for FE.

IS (Insert on Same line)

The edit string is copied. (See note 1).

IL (Insert on new Line)

A "new line" is output, and then the edit string is copied.

CO (Comment)

The edit string is ignored.

RE (Remainder)

The remainder of the input tape is copied. The copying must be stopped manually. (See Checking the Edited Tape).

ST (Stop)

The program enters a keyboard loop. When the keyboard sign digit is changed, the program continues as if CO had been read.

The two following commands are effective only when five–hole tape is being edited. For eight–hole tape they are equivalent to ST:–

SL (Set Letter)

Punches a letter shift on the output tape, and sets the input and output shift markers to "letter shift".

SF (Set Figure)

Punches a figure shift on the output tape, and sets the input and output shift shift markers to "figure shift".

Ignorable and Compound Characters

1. Eight–hole tape

The characters erase and blank, and also space and tab when not underlined, are ignorable, in the sense that they can be removed from or inserted into a string of characters without changing the value of the string, (e.g. "if a=0" and "if a = 0" are equivalent strings). Thus these characters cannot be used as "targets" for FC and DC commands. They are however copied if they occur in the edit string of an IL or IS command. (See Notes 1 and 2).

A character preceded by an underline or a vertical bar (or both) is regarded as a single character. Thus, for example, FC b will not find the b in begin. In combination with either of the non–escaping characters, space and tab are not ignorable.

2. Five–hole tape

Blank and space are ignorable in the sense given above.

The character "carriage return" is ignored under all circumstances, and the character "line feed" is treated as "new line" (i.e. carriage return, line feed). Thus it is not possible to edit tapes on which carriage return occurs unaccompanied by a line feed.

The Edit Tape for the One Reader Mode

When only one tape reader is being used the edit tape is held in the store during editing. In this case, the edit tape must be terminated by a halt character (' : ' for 5–hole tape, 'Stop Code' (H) for 8–hole tape) occurring at the start of a line. If the halt character occurs anywhere on the edit tape other than as the first character on a line, it is packed normally.

If the program tries to read beyond the end of the stored edit tape, EDIT OFLO is displayed. In the two reader mode, the edit tape would shoot out of the reader.

Checking the Edited Tape

During editing, a checksum is formed of all the characters sent to the output punch.

During checking, the characters are read and subtracted successively from the checksum. When the checksum becomes zero, the next ten characters are read from the tape. If these are all blank, it is assumed that the end of the tape has been reached, and the check is successful. END is displayed. If not, or if the checksum ever becomes negative, the check has failed and ERRSUM is displayed.

The check has also failed, if the tape shoots through the reader while being checked.

The checksum is preserved, so that the check can be repeated after a failure, in case the error is caused by the reader.

If the editing is stopped manually, then this is normally done when some of the blanks at the end of the input tape have been copied. If, however, editing is stopped while non–blank characters are being output, this must be done by depressing the MANUAL button, then the RESET button, so that it is not possible for a character to be output and yet not added into the checksum.

Notes

1. To make a print up of the edit tape more readable, a space may be punched between the function letters and the edit string. Therefore, if the first character after the function letters is a space, it is ignored. This means that to insert a space on the output tape one must give the command IS followed by two spaces.

2. In the one reader mode the length of the edit tape is limited by the amount of free store available. The tape is packed five characters per word, blank characters being ignored completely. This means that in the one reader mode, IL and IS commands cannot be used to insert blanks on the output tape. If the edit tape is too long, NOROOM is displayed.

3. The edit string can occupy only one line. If several consecutive lines are to be inserted, one can precede each by the command IL; alternatively one can use a ST command to halt the editing, and change the input tape in the reader.

4. In the five hole mode, the program assumes that the input and edit tapes have started on figure shift. Redundant shift characters are ignored during editing, except when an RE command is being obeyed. Also, if a single RE command is used to edit (i.e. copy) a five hole tape, then a figure shift character is output before copying starts.

5. In treating FL and DL commands, space is allowed for 150 non–ignorable characters in the edit string. This is more than can occur on a single line of the widest teleprinter or flexowriter. If the edit string of an FL or DL command exceeds 150 characters, the effect of the command is undefined.

6. If the first line of a program is to be the target for an FL or DL command, the program tape must begin with a "new line".

7. Since the EDITALL program modifies itself during running, it is essential that key 35 be depressed during translation of the SAP program tape.

CONFIGURATION

The basic 503 computer with either one or two paper tape readers.

TAPES

The program is written in SAP. A tape is provided.

PROCESS USED

The editing takes place in steps; in each step a command is read from the edit tape, and then an appropriate amount of the input tape is processed. The commands of the edit tape must therefore be written in the same sequence as they are to be obeyed.

A short example of an edit tape and the input and modified output is given overleaf.

J. S. Hillmore
J. W. J. Williams

January, 1964.

EXAMPLE

Input

```
procedure  equals (2); value x; integer x;
comment    equals assigns the matrix on top of the list to the name x;
begin      integer P;
P:=down;
if store [NA+x]  ≠ 0 then delete (x);
if tour  (P, store [P]) then;
end equals;
```

Output

```
procedure   equals (x); value  x; integer x;
comment     equals assigns the matrix on top of the list to the name x;
begin
if  store  [NA+x] ≠  0  then  delete  (x);
if  tour   (down, NA+x)  then;
end equals;
```

Edit Tape

```
FL    pro
FC    (
DC    2
IS    x
FL    begin
DL    P
DE    complete line has been deleted
FC    ; (
CO    last command equivalent to : FL if tour (
DC    ]
IS    down, NA+x
RE
```

CHAPTER 22: 503 LINE PRINTER OUTPUT
(Common Program)

CODE LPRINT S

FUNCTION

A. To convert one 503 tape code character to a code suitable for printing on the line printer and store it in a buffer (120 characters); printing one line on the line printer when the converted character makes the buffer full or is one which has been converted from a tape code character 'new line' (decimal 2). For this function LPRINT is used as a S.A.C. common program (see 2.1.2.4).

B. To read a tape punched in the 503 paper tape code and print it on the line printer until either the paper tape is exhausted or a halt code character (decimal 76) is read.

STORE USED

About 300 locations, including data.

METHOD OF USE

The tape LPRINT (Issue 2) cannot be translated using SAP Mk.1 ISSUE 1 because it uses the RAP word facility included in SAP ISSUE 2.

Entry Points

For A. A standard S.A.C. common program entry is made, with the character to be printed in the accumulator, either to the first or second trigger point.

COMP, LPRINT, 1 for the first entry to LPRINT. (This clears the buffer and sets pointers so that the character accepted is output at the beginning of a line.)

COMP, LPRINT, 2 for all subsequent entries.

A standard exit is made.

For B. Entry is made to the third trigger point by typing LPRINT; 3.

Operating Instructions (for function B only)

(i) Enter LPRINT at the third trigger point by typing LPRINT; 3.
(ii) Clear the keyboard.
(iii) Set the required value of n on the N2 keys (see FORMAT OF OUTPUT), then, after loading the tape to be printed in reader 1, change the sign of the word generator.
(iv) To print the last line of a tape which does not end with a new line, type LPRINT; 4.

Error Indications

The message:—

LINE PRINTER NOT AVAILABLE

is displayed if the line printer becomes unavailable for any of the following reasons:—

(i) The line printer is in the manual state or is switched off.
(ii) The paper supply is exhausted.
(iii) The paper is torn or jammed in the feed mechanism.
(iv) The throat (cover door) is open.
(v) The paper runs away, i.e. a character is called which is not on the control loop (see 1.4.3.).
(vi) The hammer drive fuse is blown.

If the error state can be cleared the run will continue automatically.

FORMAT OF OUTPUT

The standard number of characters to a line on the line printer is 120; for function A this is assumed. For function B, however, the number of characters to a line, say n, may be set at any desired value provided $n \leq 120$. If, however, N_2 is zero, the standard size line of 120 characters is used.

If more than n characters are accepted between two new line characters (decimal 2), then the remaining characters will be printed on subsequent lines. The last overflow line will be shifted as far to the right-hand side of the page as possible, to indicate that it is an overflow of characters from the previous line.

CONFIGURATION

The basic 503 computer with a line printer.

TAPE

The program is in S.A.C., suitable for translation by SAP. A tape is provided to installations with a line printer.

PROCESS USED

A number of characters are available in the 503 tape code which are not in the line printer code. The list below shows how the 503 tape code characters are represented in the line printer code if other than normal.

503 tape code	line printer code
R	S
P	Δ
B	Δ
H	Δ
~	Δ
E	ignored
a-z	A-Z
A-Z	A-Z * | |
L	causes the content of the buffer to be output on the line printer.
T	tabs are assumed to be set at 6-space intervals.
U	minus sign on the second line. *

* Two lines are printed on the line printer for each line of characters accepted. In each of the 120 positions of the 2nd line either nothing or one of the following characters will be printed:—

| (vertical bar)	the character above is an upper case letter.
– (minus)	the character above is underlined.
+ (plus)	the character above is an underlined, upper case letter.

N.B.—For function B, the 503 tape code character R is ignored and H causes the word 'wait' to be displayed on a new line and reading and printing to stop. To continue, change the sign digit. If on continuation the B digit is 1 then the last line is printed and control is returned to RAP.

L. J. Hamilton.
P. Howell.

February, 1964.

CHAPTER 23: ROOTS OF A POLYNOMIAL

CODE BAIRST A (Issue 2)

FUNCTION

To calculate the roots of an equation

$$a_0x^n + a_1x^{n-1} + \ldots\ldots\ldots\ldots + a_{n-1}x + a_n = 0$$

STORE USED

488 locations.

CONFIGURATION

The requirements are as for 503 ALGOL.

TAPE

A mnemonic tape is provided.

TIME

The time taken depends on the degree of the polynomial and the number of iterations taken to find each pair of roots. For a well-conditioned, 6th degree polynomial, the time taken is about 1 second.

PROCESS USED

The procedure is based on the Bairstow-Hitchcock iterative method of determining quadratic factors of a polynomial. If the

degree of the polynomial is odd, a zero root completes one pair of real roots. The procedure is not reliable for a polynomial with several non-zero equal roots, and when such a problem is attempted the values of K, eps 1, eps 2 and eps 3 may well have to be larger than usual (see next paragraph).

SUMMARY OF PROCEDURE

```
procedure  BAIRSTOW(n,a,eps0,eps1,eps2,eps3,K,m,x,y,nat,ex);
value      eps0,eps1,eps2,eps3,n,K;
array      a, x, y;
integer array  nat, ex;
real       eps0, eps1, eps2, eps3;
integer    n, m, K;
```

n	is the degree of the polynomial whose coefficients are held in a one-dimensional array a.
a[0]	holds the coefficient of x^n.
a[i]	holds the coefficient of x^{n-i}
a[n]	holds the constant term
eps 0	is an accuracy parameter (see below)
eps 1	is an accuracy parameter (see below)
eps 2	is an accuracy parameter (see below)
eps 3	is an accuracy parameter (see below)
K	is the maximum number of iterations to be performed in calculating each quadratic factor.

On exit from the procedure

m holds the number of pairs of roots found

x[i], y[i] are the roots of the polynomial, where i = 1, 2, ... m.

nat[i] indicates the nature of the i th pair of roots.
If nat[i] = 1, x[i], y[i] are a pair of real roots.
If nat[i] = -1, x[i], y[i] are real and imaginary parts, respectively, of a complex pair of roots.

ex[i] indicates which of the following conditions caused exit from the iterative loop in finding the i th pair of roots:-

ex[i] = 1 The absolute values of the remainders r0, r1 became less than eps 1

= 2 The absolute values of the corrections incrp, incrq became less than eps 2. This gives an indication of the accuracy of the roots.

= 3 The absolute values of the ratios incrp/p, incrq/q became less than eps 3. This gives an indication of the accuracy relative to the size of the root.

= 4 The number of iterations reaches K. The pair of roots formed are not reliable and no further effort to find additional roots is made.

For i > m the values of x[i], y[i], nat[i] and ex[i] are undefined.

eps 0 is used as a lower bound for the denominator in the expressions from which incrp and incrq are found. The recommended

value of K is 50, and this will suffice for a well-conditioned polynomial.

E.K. Ney
L.J. Hamilton
P. Simmons

April, 1964.
June, 1965.

CHAPTER 24 : 503 ALGOL LINE PRINTER PROCEDURES

CODE LPRALG A

FUNCTION

To allow the use of the line printer for output in a 503 ALGOL program.

Note: Since these procedures modify the standard ALGOL system, care must be taken when using them (see Note 3 below).

STORE USED

Total storage	less than 550 locations
of which the arrays buffer and tab	take 141 locations

METHOD OF USE

There is one main procedure 'lineprinter', and four supplementary ones. The procedure, 'lineprinter' is a device setting procedure and is used as below.

lineprinter

A call of 'lineprinter' sets the line printer as the current output device. For example, the instructions:

```
punch(1);
print £TOM £SL??, punch (2), £DICK?,
      lineprinter, £HARRY £L??;
print £WHO £L??;
```

will produce the words "TOM WHO" on punch 1,
"DICK" on punch 2, and
"HARRY" on the line printer

Thus, the line printer can be used by an ALGOL program as a serial output device.

However, actual printing only takes place when a new line character is output, or when the line printer buffer is full of characters.

Tabulating Facilities

If a tab character is output, the next character is printed in a position governed by the tab settings. The effect is exactly the same as on any typewriter.

There are 120 columns on the line printer page. These are numbered 1 to 120. There is an array called "tab" which is 20 elements long, containing integers which indicate columns. In the event of a tab character being output, spaces are placed in the line printer buffer up to the next column specified.

The elements of the array tab must be in ascending order; any element which is less than an earlier element is ignored. Thus the lists 6,12,10,24...; 6,12,12,24...; and 6,12,24...; have identical effects.

It is not necessary to set every element of the array. The number of elements set will be the number of tabs required.

The presumed tab settings are six spaces apart in columns 6, 12, 18-114, 120 and will remain so unless altered by the program. i.e. tab [1] = 6, tab [2] = 12,.... tab [20] = 120

e.g. Under the presumed settings if the last character was output in column 7, tab will place spaces in columns 8-11 and the next character will be placed in column 12.

Example:

```
tab [1] := 8;

        print lineprinter, £these £t? words £t? are £t? tabbed £L?
                           So £t? are £t? these?;
```

will output:

```
THESE**WORDS*****ARE***TABBED
```

Where * indicates 'space'

```
SO*****ARE*THESE
```

(The other tabs remain at 12, 18, 24 etc.)

The tab settings are automatically restored to 6, 12, 18, etc. at the beginning of a run of a program.

When outputting, if there are no tabs left on a line and a tab character has just been produced, then the line printer will print the subsequent information on the next line.

When using the other line printer procedures mentioned below, remember that printing and other operations are not performed on the line printer until a new line character is encountered or until the buffer is full.

topofform

This procedure causes the line printer to search for a hole in the top of form channel of the line printer vertical format control loop and print on that line. It is the responsibility of the programmer or operator in charge to see that the control loop is correctly placed before the program is run.

find (M)

This causes the line printer to search for a character of value M on the control loop and print on the corresponding line. M must take values from 0 to 30. If M is negative or greater than 30 then the words "Errcall LP" are displayed on the output typewriter. The line printer executes a top of form and the program is allowed to continue.

lines (M)

This causes the line printer to throw M lines. The call 'lines (1)' has no effect. The same range and error action apply as with 'find (M)'.

overprint

This will cause the line printer to overprint as soon as a new line character is encountered or the buffer is full. In this case the effect of a new line character is to

print on the same line all the characters formed since the last new line character. It is the responsibility of the programmer to determine proper page layout.

Notes

1. All lower case letters are converted to upper case. No account is taken of U and V.

2. If 'lineprinter' is cancelled by means of a 'punch(n)' statement, the characters in the buffer are retained so that when 'lineprinter' is called again, the next characters are placed in the buffer as if no 'punch' statement had been made.

Any characters left in the buffer at the end of a program using the line printer will be output, even if 'lineprinter' is cancelled and not reset. This is done by detecting the words "End of program" or "Space oflo" or "Int oflo" or "Subscr oflo" on the output writer; the remaining characters will then be output immediately afterwards. Thus, no characters are lost.

3. If the program is stopped other than by output of an error message or "End of program", then the next message to be typed in must be CANCEL. or RESET. or one of the standard 503 ALGOL messages.

This calls for special care before input of a precompiled program (which is input by typing the RAP message, IN.). Proceed as follows:

(i) Press the tape reader MANUAL button.

(ii) Type ALGOL.

(iii) When the HOLD-UP lamp lights, press the MESSAGE button.

(iv) Read in the precompiled tape by typing IN.

Error Print outs

All error prints are produced on the output typewriter.

1. "LP Manl" — is printed if the line printer is switched off or in the Manual state. A wait ensues until the condition is rectified.

2. "LP err 1" — is printed if the yoke is open or paper nearly out. A wait ensues until the fault is rectified.

3. "LP err 2" — is printed if the paper in the line printer is torn or has run out or a line printer error has occurred. A wait ensues until the fault is rectified.

4. "Errcall LP" — is printed if M in a call of one of the procedures 'find (M)' or 'lines (M)' is outside the range of 0 to 30. The program continues.

TAPE

A mnemonic tape is provided to installations with a line printer. This ends

Ⓗ lineprinter; Ⓗ precompile;

It can be used to produce a precompiled copy. Whichever version is used, the tape is read in before the ALGOL program and the ALGOL program must terminate with an extra end to correspond to the begin at the start of the procedures tape.

Reading the tape up to

(1) the first Ⓗ, gives no special effects

(2) the second Ⓗ, makes the lineprinter the presumed output device

(3) the very end, makes the lineprinter the presumed output device and gives a precompiled copy

(4) the first Ⓗ and then moving the tape along to just beyond the second H , gives a precompiled copy which leaves punch 1 as the presumed output device.

J. W. D. Stubbs *November, 1964.*

CHAPTER 25: PROGRAM BATCHES ON MAGNETIC TAPE

CODE BATCH S

FUNCTION

The programs comprising this package (DUMP, BRING and LOAD, issue 2) enable the user to create batches of programs on magnetic tape (DUMP) and subsequently to retrieve the programs, a batch at a time, and re-store in their original positions (BRING and LOAD). A batch is defined as the contents of the main store (apart from the program DUMP and the Reserved Area) or the contents of the main store and the core backing store; this option may be taken at the time the batch is created. There is an optional facility in issue 2 which allows the user to write LOAD on to the magnetic tape with the batch.

Programs are dumped in exactly the same form in which they are stored at that time.

These programs provide an alternative to binary output of compiled programs to paper tape.

N.B. Batches prepared using DUMP Issue 1 may not be brought to store using BRING & LOAD Issue 2. Batches which must be preserved should be brought down using BRING & LOAD Issue 1 and re-dumped using DUMP Issue 2.

STORE USED

DUMP occupies 350 locations
BRING occupies 95 locations
LOAD occupies 127 locations (+512 data locations)

METHOD OF USE

1. Naming Batches

Each batch is assigned a name at the time the batch is created. If the name "BATCH 1" is given to the first batch on a reel, all subsequent batches on that reel will follow in order after the first, thereby creating a multi-batch reel. If the first batch has any other name all subsequent batches for that reel will overwrite the previous batch, creating a single-batch reel.

2. Loading

All the programs to comprise one batch must be in store (usually they will have been assembled) before the program LOAD issue 2 (optional) and DUMP issue 2 are loaded. These programs are coded in binary for input by RAP.

(i) If the user wishes to have LOAD issue 2 written onto magnetic tape with the batch, then he must input it immediately before reading in DUMP issue 2.

(ii) If the free store area is not large enough to contain LOAD issue 2 and DUMP issue 2 then the 512 data locations allocated to LOAD may be deleted by entering LOAD at its 2nd entry point before reading in DUMP. This will not affect the running of LOAD.

The reel of magnetic tape to contain the batch must always be loaded on Handler 1.

3. Entry Points

There are five entry points to DUMP, which have the following effects:

DUMP;1.BATCHNAME.	To dump the contents of main store and backing store. The first action taken by dump is to remove itself from the RAP program list (so that it will not itself be dumped). The main store is then dumped in two sections: the area up to the RAP FF pointer and the area between the LF pointer and the Reserved Area. The core backing store is then dumped.
DUMP;2.BATCHNAME.	As entry 1 except that the core backing store is not dumped.
DUMP;3.BATCHNAME. and DUMP;4.BATCHNAME.	are reserved for future extensions to the program. At present they have the same effect as entry 2.
DUMP;5.	Prepares the reel mounted on Handler 1 for dumping by writing a dummy label at the beginning of the reel.

Example

The following typewriter output would be obtained from reading in programs A and B and writing them as a batch called "TRIAL" (all messages to the computer are underlined):

```
RESET.
IN.
A
IN.
B
IN.
LOAD
IN.
DUMP
DUMP;5.              (to prepare the reel)
END
DUMP;2.TRIAL         (dump main store including LOAD)
END
LIST.
7881                 (store now empty)
```

4. Retrieval of Programs

To retrieve the programs batched, as in the above example, and restore them to their original position in store, the programs BRING and LOAD are used. The typewriter output would be:

```
RESET.
IN.
BRING TRIAL.          (half the binary tape is read in. BRING
                      is self-triggering. LOAD is read from
                      magnetic tape and entered and the batch
                      is now brought into main store. The rest
                      of the binary tape is ignored.
END
LIST.
A
B
:
:
```

If LOAD has not been written onto magnetic tape the typewriter output would be:

```
RESET.
IN.                   (the 1st half of binary tape is read in)
BRING TRIAL.LOAD      (the rest of the binary tape is read in
                      automatically and the batch is brought
                      into main store.)
END
LIST.
A
B
:
:
```

N.B. LOAD issue 2 possesses the additional feature (required for the STAR system) that after loading the batch into store it searches for a program called STAR. If it does not find one then it transfers to RAP to display END and wait for a message. If it does find a program called STAR then either STAR is in SAP1 binary and must be stored from location 5 onwards, or STAR is a SAP2 program and starts at a higher address. In either case, LOAD enters STAR at its 3rd entry point.

5. Error Messages

H1MNL	-	Handler 1 set to "LOCAL".
NO WRT PERMIT	-	File protected when an attempt made to dump a batch. Continue, once the situation has been remedied, by typing CONT.
PTYERR	-	Parity error has occurred on reading the reel label. This could happen if the wrong density has been selected on the handler. Continue, once the situation has been remedied, by typing CONT.
CANT WRITE	-	Unable to write a block after 5 attempts. No continuation possible.
X REEL	-	Wrong reel loaded by mistake when attempting to bring a batch to store.
CANNOT READ	-	Unable to read a block after 5 attempts. No continuation possible.
TP	-	Output each time a parity error occurs when writing. This gives an indication of the reliability of the reel and/or handler.
RP,P	-	The same as TP but output when reading down a batch.

CONFIGURATION

One magnetic tape deck is required, in addition to the basic 503.

TAPES

The tapes are coded in binary for input by RAP. Two tapes are provided, one containing DUMP and one containing BRING and LOAD. The programs on the latter tape are self-triggering. The tapes may also be supplied as three separate tapes, DUMP, BRING and LOAD; in this case, BRING and LOAD are not self-triggering.

C.F.DEAL
C.M.STONES

February, 1966

CHAPTER 26: ALGOL (MK 1) PLOTTER PACKAGE

CODE PLOT A

INTRODUCTION

The package is a set of procedures to facilitate the use of the digital plotter in an Elliott 503 Algol Mk 1 program. Contained in the package are procedures for moving the pen in a raised or lowered state, drawing axes and the writing of strings and numbers.

It is desirable when using the plotter to specify distances in the units of the particular problem being programmed rather than in plotter steps. Therefore the procedures have been written with their parameters in units and two scaling factors 'scabscissa' and 'scordinate' are used within the procedures to transform the units to plotter steps. 'scabscissa' and 'scordinate' are real variables which take the form of the number of plotter steps or fraction of a plotter step corresponding to one unit.

The position of the pen is noted by three variables 'abscissa', 'ordinate' and 'penmarker' which are updated in the procedures. 'abscissa' and 'ordinate' are integer variables which record the co-ordinates of the pen position in plotter steps relative to a preset origin and 'penmarker' notes whether the pen is raised or lowered. If the pen is raised then 'penmarker' is set to zero else 'penmarker' is set to greater than zero. To set the directions of 'abscissa' and 'ordinate' a variable 'page' has been introduced. If page=1 then 'abscissa' and 'scabscissa' refer to the plotter's east-west direction and 'ordinate' and 'scordinate' to the plotter's north-south direction. If page=3, this has the effect of rotating all directions through ninety degrees in an anti-clockwise direction, 'abscissa' now referring to the plotter's north-south direction and 'ordinate' and 'scordinate' to the plotter's east-west direction.

The above global control variables may be set independently or by using the procedure 'setorigin'.

FUNCTIONS

```
procedure setorigin (e,scaa,scao,way);
value  e,scaa,scao,way;
integer  e,way;
real  scaa,scao;
```

The pen is moved to a position 'e' steps from the left margin of the plotter. 'page' is set equal to the parameter 'way' which must be 1 or 3.

If 'way' is not set to 1 or 3 then 'page' will be set equal to 1. 'scabscissa' is set to 'scaa' and 'scordinate' to 'scao'. 'abscissa' and 'ordinate' are set to zero at this point. The scaling factors are stored as floating point numbers which may be either integers or mixed numbers. It is therefore necessary to choose the scaling factors, so that when calling procedures with parameters in units, the distance obtained by multiplying the number of units by the scaling factor, when rounded to an integral number of plotter steps, is actually the distance required. A plotter step may be either 1/100th inch or 1/200 inch depending on the particular installations.

The procedure aligns the setting of the pen, the pen position marker in the plotter logic and the variable 'penmarker' in the pen-up position.

```
procedure penraise;
```

If the pen is down then it is raised else nothing happens.

```
procedure penlower;
```

If the pen is up then it is lowered else nothing happens.

[It is possible in certain circumstances when the plotter is first switched on, for the pen position marker in the plotter logic to differ from the actual state of the pen. This can be corrected in the case of 'penraise' by first giving a 'penlower' and in the case of 'penlower' by first giving a 'penraise'.]

```
procedure line (c,d);
value c,d;
real c,d;
```

This is a procedure which causes the pen to be moved in a straight line from its present position indicated by 'abscissa' and 'ordinate' to a point(c,d)

where 'c' and 'd' are in units and are the co-ordinates of the point relative to the origin. 'c' and 'd' may be positive or negative. 'abscissa' and 'ordinate' are updated during the pen movement. This procedure is used in 'drawline' and 'movepen'.

```
procedure drawline (e,f);
value e,f;
real e,f;
```

A line is drawn from the present pen position given by 'abscissa' and 'ordinate' to a point (e,f) where 'e' and 'f' are in units and are the co-ordinates of the point relative to the origin.

'e' and 'f' may be positive or negative. 'abscissa', 'ordinate' and 'penmarker' are updated during the pen movement. The pen is in the pen down position on exit from the procedure. This procedure uses the procedure 'line'.

```
procedure movepen (e,f);
value e,f;
real e,f;
```

The pen is moved in the pen up position from its present position indicated by 'abscissa' and 'ordinate' to a point (e,f) where 'e' and 'f' are in units and are the co-ordinates of the point relative to the origin. 'e' and 'f' may be positive and negative. 'abscissa', 'ordinate' and 'penmarker' are updated during the pen movement and the pen is in the pen-up position on exit from the procedure. This procedure uses the procedure 'line'.

```
procedure axes (xwidth,ywidth,px,nx,py,ny);
value xwidth,ywidth,px,nx,py,ny ;
real xwidth,ywidth;
integer px,nx,py,ny;
```

Axes are drawn with the pen commencing and finishing at the origin of the axes. 'abscissa' and 'ordinate' are not affected. 'px' and 'nx' are the number of divisions required on the positive and negative axes of the x-axis, each division being 'xwidth' units in length. Similarly, 'py' and 'ny' are the number of divisions on the positive and negative arms of the y-axis, each division being 'ywidth' units in length.

```
procedure cencharacter (number);
value number;
integer number;
```

To facilitate the plotting of points on graphs, an extra procedure has been included for drawing centred symbols about a point. The following symbols may be drawn, by calling the procedure with the parameter 'number' equal to the corresponding code number.

1 - +
2 - ◊
3 - X
4 - ⋈
5 - Y
6 - λ
7 - ←
8 - ↓
9 - →
10 - ↑

The characters are drawn starting from the ending at the centre of the character or in the case of arrows, at the tip of the arrow. 'abscissa' and 'ordinate' are not affected during the procedure. Each character is ten plotter steps in width.

```
procedure plotter (width,dir);
value width,dir;
integer width,dir;
```

By calling this procedure it is possible to use the plotter as an additional output device for printing strings and numbers. It is only necessary to call punch(1), punch(2), punch(3), lineprinter or plotter (width,dir). plotter (width,dir) may be called globally or locally in a print statement whichever is desired.

In the case of the lineprinter, output is in the same form as from the lineprinter procedures LPRALG A. Contained in the package are the procedures 'top of form' find (on)', lines (on)' and 'overprint' to be used on the lineprinter.

plotter(width,dir), sets plotter as the current output device by setting a marker for output on theplotter and by calling a procedure 'output' which modifies the Algol dynamic routines to allow theplotter and the line-printer to be used as additional output devices, the two parameters set the character size and the direction of writing.

If page=1 then by setting 'dir' equal to 1,3,5 or 7 writing is drawn in the plotter's W-E, S,N, E-W or N-S direction. If page=3 this has the effect of rotating all directions through ninety degrees and by setting 'dir' equal to 1,3,5 or 7 writing is output in the plotter's S-N, E-W, N-S or W-E direction. If the parameter is given any other value than these then the value is reduced by one and writing is output in that direction e.g. if 'dir' is set to 6 then writing is in the 5 direction.

Characters are constructed on a 6x5 grid. The size of the character is specified by the parameter 'width' which is the width of the base of the character in plotter steps and must be a multiple of five. If 'width' is given any value other than a multiple of five then the size will be rounded down to the nearest number which is a multiple of five. The smallest size of character permitted is therefore five plotter steps in width. The legibility of characters depends on the type of pen used and the length of the plotter step of the particular installation. In the case of a plotter with a 1/100 inch step using a ball-point pen, the smallest characters may be drawn but in most other cases it is advisable to use a larger width. Each character is started and finished at points 2/5 width from the base of the character. 'abscissa', 'ordinate' and 'penmarker' are updated during the procedure 'scabscissa' and 'scordinate' have no effect on the size or shape of the characters.

All telecode characters may be drawn except 10, 11, underline, and vertical bar. It is also not possible to implement newline, paper thrown, tab or backspace. If any attempt is made to output these characters then they are ignored by the procedure. If a newline, tab, backspace or paper throw is required then it is necessary to use the procedure 'movepen', Lower case letters are written in the form of cursive script to eliminate the raising and lowering of the pen between each character.

It must be remembered that output on the plotter will be in the same form as output on a punch and therefore it is necessary to set the format of numbers using the Algol format setting procedures.

The procedure 'output' modifies the standard Algol dynamic routines, which are reset on reaching 'end of program' or on the output of an error message.

If the program is stopped other than by an error message or 'end of program' then the next message to be typed must be CANCEL, or RESET or one of the standard 503 Algol messages. If a precompiled tape is to be read in, then the following procedure must be followed:

a. Press the tape reader manual button.

b. Type ALGOL.

c. When the hold-up lamp lights, press the MESSAGE button.

d. Read in the pre-compiled tape by typing IN.

EXAMPLE OF THE USE OF 'plotter'

```
begin
real a,b,c;
integer i;
comment  The program reads in 15 readings of temperature against time and
draws a table of values on the plotter.  The square and square root of the
temperature are output on punch (1) for each set and the square and the
square root of the time output on lineprinter.

setorigin (300,100,100,1);

comment  The origin is set at a distance of 300 plotter steps from the left
margin of the plotter.  The two scaling factors are set to 100 steps/unit.
'page' is made equal to 1.
```

The main title of the table is drawn by making the plotter the current output device and outputting the title as a string. The pen is then moved to the beginning of the next line, and the second line of the title is written. The punch and lineprinter titles are output in the usual way;

```
movepen (1,10);
print plotter(20,1),£TABLE1?;
movepen (0,9,5);
print plotter(15,1),£Temp£s15?Time?;
print punch (1),££1?TABLE2(Temp)£1?Square Squareroot?;
print lineprinter,££1?TABLE3(Time)£1?Square Cuberoot?;

comment The values of temperature and time are now read in and output on
the appropriate device;

for i:=1 step 1 until 15 do
begin
read a,b;
c:=9 - (i*0.5);
movepen (0,c);
plotter (10,1);
print aligned (3,1), a;
movepen (2.85,c);
print aligned (3,1), b;
print punch(1),aligned(8,1),a*a,sameline,sqrt(a);
print lineprinter,aligned(8,1),b*b,sameline,sqrt(b);

end;
end;
```

end the extra end is required since the plotter package which precedes the program begins with a begin;

EXAMPLE OF THE USE OF 'axes'

```
begin
integer i;
real temp;

comment A graph is to be drawn of temperature against time. The temperature
range is from 0°C to 100°C and readings are plotted every 25 secs. for 300 secs;

setorigin (200,2,5,1);

comment The origin is set 200 plotter steps from the left margin. The
scaling factors are set at 20 steps/sec and 5 steps/degree. Labelled axes
are now drawn with divisions every 50 secs and 10 degrees;
```

```
axes (50,10,6,0,10,0);
plotter (10,1);
for i:= 50 step 50 until 300 do
begin
movepen (i-10,0);
print digits (3),i;
end;

movepen (100,-10);
print plotter(15,1),£time(secs)?;

movepen (-75,20);
print plotter (15,3),£temp(deg.cent.)?;

for i:=10 step 10 until 100 do
begin movepen(-20,i);
      print digits(3),i;

end;

comment  Readings of temperature are read in and the points plotted on  the
graph;

for i:=25 step 25 until 300 do
begin
read temp;
movepen (i,temp);
cencharacter(1);

end;

end;
end;
```

RESTRICTIONS

(i) No precautions are taken to prevent the pen moving off the edge of the paper.

(ii) If the user's program is to output check print then precautions must be taken to ensure that the current output device at the time of the check print is a punch and not the plotter.

ERROR INDICATIONS

(i) If the plotter is in the MANUAL state, the plotter instructions will be held up by the 'busy' state. If the plotter is switched off ERRINT 4 will occur when the first instruction is issued to the plotter.

(ii) The normal Algol print errors will be displayed in the event of an error occurring.

(iii) Lineprinter errors will be the same as those described for the lineprinter procedures LPRALG A.

TIME

During all pen movements the pen moves at maximum speed.

SUMMARY OF PROCEDURES

setorigin (e,scaa,scao,way) - sets the origin 'e' steps from the left margin, 'scaa' and 'scao' set the scaling factors and 'way' sets the variable 'page'.

penraise - the pen is raised

penlower - the pen is lowered

movepen (e,f) - the pen is moved to the point (e,f)

drawline (e,f) - a line is drawn to the point (e,f).

axes (xwidth,ywidth,px,nx,py,ny) - axes are drawn with 'px' and 'nx' divisions on each arm of the x-axis, each division being 'x-width' units in length. Similarly for the 'py', 'ny' and 'y-width' on the y-axis.

cencharacter(number) - to draw any one of the ten centred characters about a point , the character being specified by the parameter 'number'.

plotter (width,dir) - allows the use of the plotter as an additional output device to draw characters of size 'width' steps in width and in a direction given by 'dir'.

M.Atkinson September, 1965.

CHAPTER 26 ALGOL (MK 1) PLOTTER PACKAGE

CODE PLOT A

INTRODUCTION

The package is a set of procedures to facilitate the use of the digital plotter in an Elliott 503 Algol Mk 1 program. Contained in the package are procedures for moving the pen in a raised or lowered state, drawing axes and the writing of strings and numbers.

It is desirable when using the plotter to specify distances in the units of the particular problem being programmed rather than in plotter steps. Therefore the procedures have been written with their parameters in units and two scaling factors 'scabscissa' and 'scordinate' are used within the procedures to transform the units to plotter steps. 'scabscissa' and 'scordinate' are real variables which take the form of the number of plotter steps or fraction of a plotter step corresponding to one unit.

The position of the pen is noted by three variables 'abscissa', 'ordinate' and 'penmarker' which are updated in the procedures. 'abscissa' and 'ordinate' are integer variables which record the co-ordinates of the pen position in plotter steps relative to a preset origin and 'penmarker' notes whether the pen is raised or lowered. If the pen is raised then 'penmarker' is set to zero else 'penmarker' is set to greater than zero. To set the directions of 'abscissa' and 'ordinate' a variable 'page' has been introduced. If page=1 then 'abscissa' and 'scabscissa' refer to the plotter's east-west direction and 'ordinate' and 'scordinate' to the plotter's north-south direction. If page = 3, 5 or 7, this has the effect of rotating all directions through 90^{o}, 180^{o} or 270^{o} in an anti-clockwise direction.

The above global control variables may be set independently or by using the procedure 'setorigin'.

FUNCTIONS

```
procedure setorigin (e, scaa, scao, way);
value e, scaa, scao, way;
integer e, way;
real scaa, scao;
```

The pen is moved to a position 'e' steps from the left margin of the plotter. 'page' is set equal to the parameter 'way' which must be 1, 3, 5 or 7.

If 'way' is not set to 1,3, 5 or 7 then 'page' will be set equal to 1. 'scabscissa' is set to 'scaa' and 'scordinate' to 'scao'. 'abscissa' and 'ordinate' are set to zero at this point. The scaling factors are stored as floating point numbers which may be either integers or mixed numbers. It is therefore necessary to choose the scaling factors, so that when calling procedures with parameters in units, the distance obtained by multiplying the number of units by the scaling factor, when rounded to an integral number of plotter steps, is actually the distance required. A plotter step may be either 1/100th inch, 1/200 inch or 0.1 mm, depending on the particular installations.

The procedure aligns the setting of the pen, the pen, the pen position marker in the plotter logic and the variable 'penmarker' in the pen-up position.

```
procedure penraise;
```

If the pen is down then it is raised else nothing happens.

```
procedure penlower;
```

If the pen is up then it is lowered else nothing happens.

[It is possible in certain circumstances when the plotter is first switched on, for the pen position marker in the plotter logic to differ from the actual state of the pen. This can be corrected in the case of 'penraise' by first giving a 'penlower' and in the case of 'penlower' by first giving a 'penraise'.]

```
procedure line (c, d);
value c, d;
real c, d;
```

This is a procedure which causes the pen to be moved in a straight line from it's present position indicated by 'abscissa' and 'ordinate' to a point (c, d) where 'c' and 'd' are in units and are the co-ordinates of the point relative to the origin; 'c' and 'd' may be positive or negative. 'abscissa' and 'ordinate' are updated during the pen movement. This procedure is used in 'drawline' and 'movepen'.

```
procedure drawline (e, f);
value e, f;
real e, f;
```

A line is drawn from the present pen position given by 'abscissa' and 'ordinate' to a point (e, f) where 'e' and 'f' are in units and are the co-ordinates of the point relative to the origin.

'e' and 'f' may be positive or negative. 'abscissa', 'ordinate' and 'penmarker' are updated during the pen movement. The pen is in the pen down position on exit from the procedure. This procedure uses the procedure 'line'.

```
procedure movepen (e, f);
value e, f;
real e, f;
```

The pen is moved in the pen up position from its present position indicated by 'abscissa' and 'ordinate' to a point (e, f) where. 'e' and 'f' are in units and are the co-ordinates of the point relative to the origin. 'e' and 'f' may be positive and negative. 'abscissa', 'ordinate' and 'penmarker' are updated during the pen movement and the pen is in the pen-up position on exit from the procedure. This procedure uses the procedure 'line'.

```
procedure axes (xwidth, ywidth, px, nx, py, ny);
value xwidth, ywidth, px, nx, py, ny;
real xwidth, ywidth;
integer px, nx, py, ny;
```

Axes are drawn with the pen commencing and finishing at the origin of the axes. 'abscissa' and 'ordinate' are not affected. 'px' and 'nx' are the number of divisions required on the positive and negative axes of the x-axis, each division being 'xwidth' units in length. Similarly, 'py' and 'ny' are the number of divisions on the positive and negative arms of the y-axis, each division being 'ywidth' units in length.

```
procedure plotchar (n, size, angle);
value n, size, angle;
integer n, size:
real angle;
```

All characters to be plotted are output by the procedure. Characters are formed by stating the co-ordinates which are to be joined, by the procedure line, in the pen up or down position as specified. The characters are constructed on a grid, the maximum size being 15 x 15. If a user wishes to alter or replace any of the standard characters, they can construct the character as required, then substitute the new 'elliott' orders in the required places. If there are more or fewer 'elliott' orders then the lookup table should be altered accordingly.

The lookup table is in the form 'elliott (0, 0, 0, 0, 0, 0, n);' where 'n' is the address of the character -1.

The characters are in the form:

m p(x, y) p(x, y) p(x, y) p(x, y)

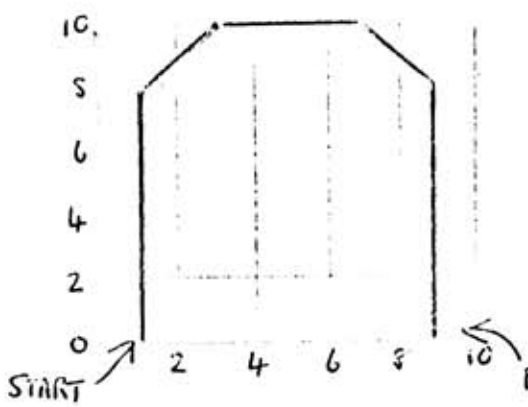
m = 0 or 1	If m = 0 then the next 'elliott' order is to be plotted but if m = 1 then this is the last elliott order.
p = 0 or 1	If p = 0 then raise pen else if p = 1 then lower pen.
$0 \leq x \leq 15$	This is the x co-ordinate that the pen must be moved to
$0 \leq y \leq 15$	This is the y co-ordinate that the pen must be moved to.

The character in 'elliott' form will look like:

Bit 39 Bit 1

1	1	1111	1111	1	1111	1111	1	1111	1111	1	1111	1111	00
M	P	x	y	P	x	y	P	x	y	P	x	y	

Example of Producing Characters



The co-ordinates for the character would be as follows:-

0 0(1, 0) 1(1, 8) 1(3, 10) 1(7, 10)

0 1(9, 8) 1(9, 0) 1(9, 5) 1(1, 5)

1 0(10, 0) 0(10, 0) 0(10, 0) 0(10, 0)

These co-ordinates can then be converted to 'elliott' orders by the program CONVERT CO-ORDINATES TO ELLIOTT ORDERS (Appendix 2) which is written in ALGOL.

A test program 'PLOT CO-ORDINATES' (Appendix 3) is also available which reads the co-ordinates in the above form and plots them. This enables the user to test various characters without editing the package every time.

```
procedure cencharacter (number);
value number;
integer number;
```

To facilitate the plotting of points on graphs, an extra procedure has been included for drawing centered symbols about a point. The following symbols may be drawn, by calling the procedure with the parameter 'number' equal to the corresponding code number.

1 - +
2 - ◇
3 - X
4 - ⋈
5 - Y
6 - ⟩
7 - ↙
8 - ↓
9 - →
10- ↑

The characters are drawn starting from and ending at the centre of the character or in the case of arrows, at the tip of the arrow. These characters are drawn on an 8 x 8 grid, and may be varied in size by altering the global variable 'censize'. A standard setting of 10, is made by the plotter package, which then constructs characters on a grid of size 10 x 10 plotter steps.

```
procedure plotter (height, dir);
value height, dir;
integer height, dir;
```

By calling this procedure it is possible to use the plotter as an additional output device for printing strings and numbers. This

procedure may be called globally or locally in a print statement whichever is desired.

'plotter (height, dir)', sets the plotter as the current output device by calling the procedure 'output' which modifies the ALGOL dynamic routines to allow the plotter to be used as an additional output device. The two parameters set the character size and the direction of writing.

If 'dir' is set equal to one then this has the effect of writing from W-E by increasing dir by 1 then this has the effect of rotating the plotter through 45 degrees in an anti-clockwise direction. This increase may be done for $1 \leqslant dir \leqslant 8$.

Characters are constructed on a 10 x 10 grid, starting at point (0, 0) and ending at point (0, 10). The size of the character is specified by the parameter 'height' which is 4/5 the height of the character in plotter steps. The legibility of characters depends on the type of pen used and the length of the plotter step of the particular installation. 'abscissa', 'ordinate' and 'penmarker' are updated during the procedure. 'scabscissa' and 'scordinate' have no effect on the size or shape of the characters.

All telecode characters may be drawn except 10, 11, underline, and vertical bar. It is also not possible to implement newline, paper thrown, tab or backspace. If any of these characters are given to the 'output' procedure then a space is left. Lower case letters are converted into upper case, since it is assumed to be satisfactory for most users. There is however an edit provided (Appendix 1) which will insert lower case letters, which are constructed on an 8 x 15 grid. If a user wishes to continue to use the character set from issue 2 of the plotter package, then a tape containing the co-ordinates for these characters is available on request.

It must be remembered that output on the plotter will be in the same form as output on a punch and therefore it is necessary to set the format of numbers using the ALGOL format setting procedures.

The procedure 'output' modifies the standard Algol dynamic routines provided MOD 3 is in store; if it is absent then the result is undefined.

EXAMPLE OF THE USE OF 'plotter'

```
begin
real a, b, c:
integer i;
comment The program reads in 15 readings of temperature against time
and draws a table of values on the plotter. The square and square root of
the temperature are output on punch (1) for each set and the square and
the square root of the time output on lineprinter.

setorigin (300, 100, 100, 1);

comment The origin is set at a distance of 300 plotter steps from the left
margin of the plotter. The two scaling factors are set to 100 steps/unit.
'page' is made equal to 1.

The main title of the table is drawn by making the plotter the current
output device and outputting the title as a string. The pen is then moved
to the beginning of the next line, and the second line of the title is written.
The punch and lineprinter titles are output in the usual way;

movepen (1, 10);
print  plotter (20, 1), £TABLE1?:
movepen (0, 9, 5):
print  plotter (15, 1), £Temp£s15?Time?:
print  punch (1), ££1?TABLE2(Temp)£1?Square Squareroot?;
print  lineprinter, ££1?TABLE3(Time)£1?Square Cuberoot?:

comment The values of temperature and time are now read in and output
on the appropriate device;
```

```
for i:=1 step 1 until 15 do
begin
read a, b:
c:=9 - (i *0.5):
movepen (0, c);
plotter (10, 1);
print aligned (3, 1), a;
movepen (2.85, c);
print aligned (3, 1), b;
print punch (1), aligned (8, 1), a*a, sameline, sqrt(a);
print lineprinter, aligned(8, 1), b*b, sameline, sqrt(b);

end;
end:
```

end the extra end is required since the plotter package which precedes the program begins with a begin;

EXAMPLE OF THE USE OF 'axes'

```
begin
integer i:
real temp;

comment A graph is to be drawn of temperature against time. The
temperature range is from 0°C to 100°C and readings are plotted every
25 secs. for 300 secs;

setorigin (200, 2, 5, 1);

comment The origin is set 200 plotter steps from the left margin. The
scaling factors are set at steps/sec and 5 steps/degree. Labelled axes
are now drawn with divisions every 50 secs and 10 degrees;

axes (50, 10, 6, 0, 10, 0);
plotter (10, 1);
for i:=50 step 50 until 300 do
begin
movepen (i-10, 0);
print digits (3), i:
end;

movepen (100, -10);
print plotter (15, 1), £time (secs)?;
```

```
movepen (-75, 20);
print plotter (15, 3), £temp(deg. cent.)?;

for i:=10 step 1- until 100 do
begin movepen (-20, i);
      print digits (3), i;

end;

comment Readings of temperature are read in and the points plotted on the
graph;

for i:=25 step 25 until 300 do
begin
read temp;
movepen (i, temp);
cencharacter (1);

end;
end;
end;
```

RESTRICTIONS

(i) No precautions are taken to prevent the pen moving off the edge of the paper.

(ii) If the user's program is to output check print then precautions must be taken to ensure that the current output device at the time of the check print is a punch and not the plotter.

(iii) MOD3 must be in store when using the plotter package. (see Appendix 4).

(iv) Procedure 'output' must not be segmented.

ERROR INDICATIONS

(i) If the plotter is in the MANUAL state, the message 'PL man' is displayed on the typewriter and the plotter instruction will then be held up by the 'busy' state. If the plotter is switched off ERRINT 4 will occur when the first instruction is issued to the plotter.

(ii) The normal Algol print errors will be displayed in the event of an error occurring.

TIME

During all pen movements the pen moves at maximum speed.

General Purpose Software Group.
(with acknowledgements to the G.P.O. for their development work in connection with Issue 3)

APPENDIX 1

NAME

EDIT TO PLOTTER PACKAGE
FOR LOWER CASE CHARACTERS.

FUNCTION

To edit the PLOTTER PACKAGE (either the ALGOL1 or ALGOL3 version) in order to produce a version for users who require lower-case cursive script.

METHOD OF USE

A new PLOTTER PACKAGE tape should be produced using the program EDIT8 (see description in section 2.2.3.27 of the MANUAL).

APPENDIX 2

NAME — CONVERT CO-ORDINATES TO ELLIOTT ORDERS.

FUNCTION — To modify the character set of the plotter package.

METHOD OF USE — This program will convert the character format, as specified in the plotter package description, into the correct 'elliott' orders.

There is also a print-up on the lineprinter of the co-ordinates together with the produced 'elliott' orders.

OPERATING

(i) The program is compiled under ALGOL.
(ii) Load the data tape i.e. co-ordinate tape, in reader 1 and change the dwait key (key 19).
(iii) Output occurs to punch (1) and the lineprinter. The data tape should be terminated with a halt code (decimal 76) which will cause a 'dwait' at the end.

TAPE — No tape is issued but the text of the program is as follows:

```
CONVERT CO-ORDINATES TO ELLIOTT ORDERS:
begin
integer F11,F12,N1,BLINE,F21,F22,N2,i,p,x,y,m;
switch ss:=convert;
sameline:
convert: read m;
digits(2):
print punch(4), £££1??, m;
for i:=1 step 1 until 4 do
begin
read p,x,y:
print punch(4), ££s2??, p, £(?, x, £, ?, y, £)?;
elliott (3,0,m,0,5,5,1):
elliott (0,4,p,0,5,5,4):
elliott (0,4,x,0,5,5,4):
elliott (0,4,y,0,2,0,m):
```

```
end;
p:=7:
elliott (3, 0, m, 0, 5, 5, 2):
elliott (5, 0, 36, 0, 0, 3, p):
elliott (1, 6, F11, 0, 5, 4, 3):
elliott (1, 6, F12, 0, 5, 4, 13):
elliott (1, 6, N1, 0, 5, 4, 1):
elliott (1, 6, BLINE, 0, 5, 4, 3):
elliott (1, 6, F21, 0, 5, 4, 3):
elliott (1, 6, F22, 0, 5, 4, 13);
elliott (1, 6, N2, 0, 0, 0, 0):
elliott (7, 3, p, 1, 4, 3, 1);

digits(4):
print £ £ℓ?elliott (?, F11, £, ?, F12, £, ?, N1, £, ?, BLINE, £, ?, F21, £, ?,
F22, £, ?, N2, £);?;
print punch(4), £ £t?elliott (?, F11, £, ?, F12, £, ?, N1, £, ?, BLINE, F, ?,
F21, £, ?, F22, £, ?, N2, £):?:
if F11=3 then print £ £ℓ? ?, punch(4), £ £ℓ? ?:
goto convert:
end:
```

APPENDIX 3

NAME — PLOT CO-ORDINATES

FUNCTION — To plot characters direct from a data tape.

METHOD OF USE — This program uses the plotter package and plots the co-ordinates read from paper tape. This is done by the procedures 'movepen' and 'drawline'. The size of characters may be altered by the setting of the origin which is typed in every time the program is run.

OPERATING

(i) Compile the plotter package under ALGOL.
(ii) Load the program tape in reader 1 and change the sign bit.
(iii) Load the data tape in reader 1 and change the dwait key.
(iv) 'setorigin' is output to the typewriter.
(v) Type the values for the parameter for procedure setorigin as follows

e (s) scaa (s) scao (s) way (s)

This will set the origin (see procedure 'setorigin' in the plotter package description) and character will then be read and plotted. By changing the values of 'scaa' and 'scao' the size of character can be varied.

TAPE — No tape is issued but the text of the program is as follows:-

```
punch(3):
print ££ℓ?PLOT CO-ORDINATES?;
begin
integer m,p,x,y,i,xstart;
switch ss:=repeat,start;

print ££ℓ?setorigin ?;
read reader(3),m,x,y,p;
setorigin(m,x,y,p);
start:xstart:=abscissa/scabscissa:
repeat:=read m;
for i:=1 step 1 until 4 do
```

PLOT A

```
begin
read p,x,y:
if p=0 then movepen(x+xstart,y) else drawline(x+xstart,y);
end;
if m=0 then goto repeat else goto start;
end:
end.
```

APPENDIX 4

TRANSLATION TO OWNCODE ON PAPER TAPE

If this ALGOL 1 facility is being used with the PLOTTER PACKAGE a special PLOTTER MOD is required (the tape is issued to basic installations). Operating instructions are as follows:

a) TO PRODUCE OWNCODE

Step	Action	Message Output	Tape in reader
1	Type RESET.		
2	Type IN.	ALGOL1	ALGOL (Tape 1)
3	Change Sense Key 39	ALGOL2 swait	ALGOL (Tape 2)
4	Press message button		
5	Type OWNOUT.		
6	Change key 39	swait	PLOTTER PACKAGE
7	Change key 39	swait	Users program
8	Owncode tape is output on punch 1.		

b) TO INPUT OWNCODE AND RUN THE PROGRAM

9	Type RESET.		
10	Type IN.	OWNIN ALGOL2	ALGOL (Tape 2)
11	Press message button		
12	Type IN.	ALGOL2 UN-CHEK	PLOTTERMOD (first half)
13	Type IN.	OWNIN UN-CHEK	PLOTTERMOD (second half)
14	Type OWNIN.		Owncode tape
15		PROGRAM NAME FREE STORE a-b Dwait	
16	Change key 19		Data tape

CHAPTER 27: EDIT8

CODE EDIT8 S

FUNCTION

To produce a modified copy of an eight-hole tape (the input tape) by means of deletion, insertion and replacement of strings of characters.

The alterations are specified by means of an "edit tape".

STORE USED

Program 208 locations
Workspace 56 locations

METHOD OF USE

The program has 6 entry points.

Entry 1

This entry is the normal entry point to the program and requires the input tape to be loaded in reader 1 and the edit tape to be loaded in reader 2.

The edited tape is output on punch 1.

If the edit tape ends with an RE command (see COMMANDS), the editing must be stopped by pressing the MANUAL button and then pressing RESET.

Entry 2 Type:- EDIT8;2.

This entry is for checking the edited tape for punching errors and requires the tape to be loaded in reader 1.

The message ERRSUM indicates an error.

Entry 3 Type:- EDIT8;3.

This entry performs in the same way as entry 1 except that no edited tape is produced. The edit can therefore be tested without using computer time producing an edited tape which may be incorrect. If the edit tape ends with an RE command, the input tape will shoot through the reader.

Entry 4 (Reserved for future use for typing in edit)

Entries 5 and 6

These are used to enable EDIT8 to be used as a common program. Entry 5 is used to set up the system to act in this way and Entry 6 causes the characters to be supplied to the calling program rather than be output to paper tape.

COMMANDS

Each command occupies a single line of a print-up of the edit tape (with the exception of command IB). The first two non-ignorable characters on the line specify the function; the remaining characters up to, but not including the next "new line" form the "edit string". If a line contains no function characters or an invalid command the remainder of the line is copied onto the output writer (See Note 1).

FL (Find Line)

The edit string is read. The input tape is then copied until a line beginning with this string is found. The last character copied is the last character of the edit string. (See notes 3 and 4).

DL (Delete to Line)

The edit string is read. The input tape is then skipped until a line beginning with this string is found. The last character skipped is the last character of the edit string. (See notes 3 and 4).

FC (Find Characters successively)

If the characters of the edit string are C1 C2 C3...Cn, then the input tape is copied until C1 has been copied, then further until C2 has been copied and so on until Cn has been copied.

DC (Delete to Characters successively)

If the characters of the edit string are C1 C2 C3 ... Cn, then the input tape is skipped until C1 has been skipped, then further until C2 has been skipped, and so on until Cn has been skipped.

FE (Find End of Line)

The input tape is copied up to, but not including, the next "new line". The "new line" character is read and stored in a buffer, and is held whilst insertions, if any,are made from the edit tape. If the next command after the insertions is a "find" command, then the buffer character is output; if it is a "delete" command, the buffer character is ignored. The edit string is ignored.

DE (Delete to End of Line)

The input tape is skipped up to but not including the next "new line". The treatment of this "new line" and of the edit string is the same as for FE.

IS (Insert on Same Line)

The edit string is copied (See Note 1).

IL (Insert on New Line)

A "new line" is output, and then the edit string is copied.

IR (Insert Runout)

4" of blank tape is output.

IB (Insert Block)

The first line of the edit string is read and stored. The edit tape is then copied until a line beginning with this stored string is found. The last character copied is the last character of the stored string.

e.g. To insert:-

```
begin  MAIN;
       SUBR,INOUT
       COMP,TAX
       40   start
end;
```

on the output tape, it is necessary to punch:-

```
IB  end;
begin  MAIN;
       SUBR,INOUT
       COMP,TAX
       40   start
end;
```

On the edit tape.

CO (Comment)

The edit string is copied to the output writer, starting with the first character after CO. The comment appears on a new line and the terminating "new line" character is ignored.

RE (Remainder)

The remainder of the input tape is copied. The copying must be stopped manually.

ST (Stop)

The message E8Wait is displayed on new line and the program then enters a number generator loop. When the sign digit (Key 39) of the number generator is changed, the program continues as if CO had been read.

Ignorable and Compound Characters

The characters erase and blank, and also space and tab, when not underlined are ignorable, in the sense that they can be removed from or inserted into a string of characters without changing the value of the string, (e.g. "4OREPEAT*MAIN" and "4O REPEAT * MAIN" are equivalent strings). Thus these characters cannot be used as "targets" for FC and DC commands. They are copied, however, if they occur in the edit string of an IL or IS command (See Note 1).

A character preceded by an underline or a vertical bar (or both) is regarded as a single character. Thus, for example, FCb will not find the b in begin. In combination with either of the non-escapable characters, space and tab are not ignorable.

Checking the Edited Tape

During editing, a checksum is formed of all the characters sent to the output punch. During checking, the characters are read and subtracted successively from the checksum. When the checksum becomes zero, the next ten characters are read from tape. If these are all blank, it is assumed that the edit has been successful. END is displayed. If not, or if the checksum ever becomes negative, the check has failed and ERRSUM is displayed.

The check has also failed if the tape shoots through the reader whilst being checked.

The checksum is preserved, so that the check can be repeated after a failure, in case the error is caused by the reader.

If the editing is stopped manually, then this is normally done when some of the blanks at the end of the input tape have been copied. If, however, editing is stopped whilst non-blank characters are being output this must be done by depressing the MANUAL button, then the RESET button, so that it is not possible for a character to be output and yet not be added into the checksum.

Notes

1. To make the print up of the edit tape more readable, a space may be punched between the function letters and the edit string. Therefore, if the first character after the function letters is a space, it is ignored. This means that to insert a space on the output tape one must give the command IS followed by two spaces.

2. The edit string can occupy one line only, with the exception of the IB command. If several consecutive lines are to be inserted one can use the IB command or precede each line by the command IL; alternatively, one can use a ST command to halt the editing and change the input tape in the reader.

3. In treating FL and DL commands, space is allowed for 30 non-ignorable characters in the edit string. If the edit string of a FL or DL command exceed 30 characters, the effect of the command is undefined.

4. If the first line of a program is to be the 'target' for a FL or DL command, the program tape must begin with a 'new line'.

5. Since the EDIT8 program modifies itself during running, it is essential that key 35 be depressed during translation of the SAP program tape.

CONFIGURATION

The basic 503 computer with two paper tape readers.

TAPES

The program is written in SAP.

PROCESS USED

The editing takes place in steps; in each step a command is read from the edit tape, and then an appropriate amount of the input tape is processed. The commands of the edit tape must therefore be written in the same sequence as they are to be obeyed.

CHAPTER 28: GENERAL NUMBER PRINT PROGRAM

CODE NPRINT S

1. FUNCTION

To output on punch 1, punch 2, directly connected typewriter or line printer, in 8-channel (or line-printer) code, the number in the accumulator, interpreting it either as a number in standard automatic floating point form, or as an integer.

2. STORE USED

About 500 locations, including data.
Auxiliary register - affected.
Overflow indicator - affected.
Accumulator contents random on exit.

3. METHOD OF USE

Normal format

(a) Every number is preceded by + or - and is followed by S S

(b) Non-significant leading zeros are suppressed and spaces are output in their place. The sign is printed immediately before the first digit or the point.

2.2.3.28

NPRINT S

ENTRY

A standard S.A.C. common program entry is made to one of 5 trigger points. The entry is followed by a parameter word determining mode of output. The number to be printed must be in the accumulator. This program must be used with SAP Issue 2 as it uses facilities not available in Issue 1.

```
COMP,   NPRINT,     P
     a0   m    B     p     n
```

A standard exit is made to 2 words after entry (i.e. directly following the parameter word).

PARAMETER

For output on PUNCH 1	set a = 0	p = 00
For output on PUNCH 2	a = 1	p = 00
For output on DIRECTLY CONNECTED TYPEWRITER	a = 2	p = 00
For output on LINE PRINTER	a = 3	

p = 6 bit number (written in octal) to be stored in the first word of the line printer buffer whenever a new line is started. p controls vertical format (see 503 Manual 1.4.3, page 2).

Octal p	Decimal Value	Action
$00 \leq p \leq 36$	$0 \leq p \leq 30$	Move (p + 1) lines and print
p = 37 or p = 40	p = 31 or p = 32	Overprint (no line feed)
$41 \leq p \leq 76$	$33 \leq p \leq 62$	Look for configuration (p - 32) on paper tape control
p = 77	p = 63	"Top of form"

B - DIGIT If B = 0 the number is punched on the same line as the preceding number (or stored further up in the same buffer in the case of the line printer). The buffer has 120 locations. If the buffer is full it is printed and the buffer is stored at the beginning of the new buffer.

If B = 1 the program will punch L before the number. In the case of the line printer, if B = 1 the buffer will be printed as a single line and p from the parameter word will be set in the first location of the new buffer. The new number is then stored at the beginning of the buffer.

ENTRY 1

```
COMP,     NPRINT,     1
a0    0     B      p      n    (parameter word)
```

To print a signed floating point number as a decimal fraction of n digits followed by $_{10}$ and a decimal exponent consisting of a sign and 2 digits. Altogether (n + 8) characters appear on the line.

e.g. number + 123.456 with parameter word 00 0 : 00 4
produces output on punch 1 in the form + . 1235 $_{10}$ + 0 3 S S

Error Action

If n is zero, output will be as for n = 9.

ENTRY 2

```
COMP,     NPRINT,     2
a0    m     B    p     n     (parameter word)
```

To print a signed floating point number with m digits before and n digits after the decimal point. Altogether (m + n + 4) characters appear on a line. If n = 0 the decimal point is replaced by a space.

e.g. number + 123.456 with parameter word 10 4 : 00 2

produces output on punch 2 in the form S + 123.46 S S

Error Action

If the number A is such that $|A| \geq 10^{m}$ or $|A| \geq 10^{11}$ then L ? is output followed by A as printed by entry 1 with n = 9.

ENTRY 3

```
COMP,       NPRINT,       3
a0     0      B     p       n
```

To print a floating point number of n digits with a decimal point in the appropriate position. Altogether (n + 4) characters appear on the line. If the number is such that it contains exactly n significant decimal digits before the point, the decimal point is replaced by a space.

e.g. number 123.456 with parameter word 00 0 : 00 4
produces output on punch 1 in the form +123.5 S S
but if parameter word is 00 0 : 00 3 output is
+123 S S S

Error Action

If a number A is such that $|A| \geq 10^n$ or $|A| \geq 10^{11}$, then L ? is output followed by A as printed by entry 1 with n = 9. If n = 0 is set the format of entry 1 is used with n = 9.

ENTRY 4

```
COMP,     NPRINT,     4
a0    0   B     p     n
```

To print an integer with n digits. A total of (n + 3) characters appear on the line.

e.g. integer 1234 with parameter word 30 0 / 10 6

produces output on the line printer. The buffer is printed (because B = 1). After printing the paper is moved 9 lines (p = 8) and S S + 1234 S S is stored at the beginning of the buffer.

Error Action

If the integer I is such that $|I| \geq 10^n$, then L ? is output followed by I with n = 12. If n = 0 is set then n = 12 is used.

LINE PRINTER

Care must be taken when the line printer is being used, to ensure that the final buffer is printed. To do this, after the last number has been stored in the buffer in the usual way, clear the accumulator and enter NPRINT with the instruction.

```
COMP,     NPRINT,     5
30  0   /    00    0
```

Error Indication

The message:= LINE PRINTER NOT AVAILABLE is displayed if the line printer becomes unavailable for any of the following reasons:-

(i) The line printer is in the manual state or switched off.

(ii) The paper supply is exhausted.

(iii) The paper is torn or jammed in the feed mechanism.

(iv) The throat (cover door) is open.

(v) The paper runs away i.e. a character is called which is not on the control loop (see 1.4.3).

(vi) The hammer drive fuse is blown.

If the error state can be cleared the run will continue automatically.

FORMAT

Normal format for output is

(a) Every number is preceded by + or - and is followed by S S.

(b) Non-significant leading zeros are suppressed and spaces are output in their place. The sign is printed immediately before the first digit or the point.

The format can be changed from normal in several ways. The modified format will be used in all subsequent outputs using NPRINT (N.B. These modifications corrupt NPRINT so that it no longer sum checks. To preserve the sum check the format must be set back to normal by entering:

```
COMP,   NPRINT,   6
```

No parameter is needed and the
to the instruction following its entry. The format can be set back
to normal by the instruction COMP, NPRINT, 6.)

MODIFICATIONS

1. To print S instead of +

(a) before integers : Enter COMP, NPRINT, 7
No parameter. Exit to instruction following entry.

(b) before floating point numbers : Enter COMP, NPRINT, 8
No parameter. Exit to instruction following entry.

(c) before exponent in form produced by entry 1:
Enter COMP, NPRINT, 9
No parameter. Exit to instruction following entry.

2. To suppress sign altogether.

(a) before all numbers: Enter COMP, NPRINT, 10
No parameter. Exit to instruction following entry.

(b) before exponent in the form produced by entry 1:
Enter COMP, NPRINT, 11
No parameter. Exit to instruction following entry.

3. To print character of value K instead of ..

Enter COMP, NPRINT, 12
Parameter +K. Exit to instruction following parameter.

4. To print characters of value K, L instead of S S at the end of the number:

Enter COMP, NPRINT, 13
Parameter 1) +K
Parameter 2) +L

Exit to instruction following parameter 2.

5. To suppress S S at end of number: Enter COMP, NPRINT, 14
No parameter. Exit to instruction following entry.

M.PADMORE,
COMPUTER RESEARCH
LABORATORY.

NOVEMBER, 1965.

CHAPTER 29: THE FORTRAN TO ALGOL TRANSLATOR

CODE FEAT A (Core Backing Store Version)
MTFEAT A (Magnetic Tape Version)

CONTENTS

CHAPTER 29: FORTRAN TO ALGOL TRANSLATOR

1. FUNCTION OF THE TRANSLATOR

The function of the FORTRAN to ALGOL Translator is to translate a working FORTRAN program into its ALGOL equivalent in a form which is acceptable to the Elliott ALGOL Compiler. FEAT (or MTFEAT) translates into ALGOL 60 and FORTRAN programs translated in this way may, therefore, be run on non-Elliott machines, since input and output is effected by procedures.

FEAT (or MTFEAT) will handle the majority of routines written in FORTRAN II or FORTRAN IV with a few restrictions (see Restrictions and Notes) and tapes are provided to cater for both.

It is assumed that the reader is familiar with both the ALGOL and FORTRAN languages.

2. GENERAL DESCRIPTION

There are two versions of the translator program:-

(i) FEAT which uses core-backing store to store the FORTRAN programs.
(ii) MTFEAT which uses magnetic tape to store the FORTRAN programs.

All following references to FEAT or MTFEAT apply specifically to that particular version of the translator.

The translator program is written in ALGOL and is input and translated accordingly by ALGOL 3 or ALGOL 1. When FEAT (or MTFEAT) has been translated, the program reads in the Standard functions and Decode Tables as data. The FORTRAN sub-programs are then input, written to backing store (core-backing store in the case of FEAT and magnetic tape handler 4 in the case of MTFEAT), processed and output as an equivalent series of ALGOL statements. Certain other information is also output (see 7).

The mnemonic tapes which comprise a translated FORTRAN program must be input in a specified order, followed by the requisite data (see Appendix I). The FORTRAN program when translated is an ALGOL program and is, therefore, read in and translated by the ALGOL compiler.

Note MTFEAT has been segmented to allow larger arrays for dictionaries. If ALGOL 1 is used to translate MTFEAT, MOD 8 or MOD 9 must, therefore, be in store. A magnetic tape routine MTSTORE is also required.

3. CONFIGURATION

3.1 FEAT

The use of FEAT requires storage comprising the 503 Main Store plus one unit of Core Backing Store. To translate FEAT itself the ALGOL tapes must be in store. These will occupy part of the main store and locations 0 to 8192 of Core Backing Store. FEAT is read into the main store occupying approximately 6000 locations. The remaining area of free main store is used by FEAT as space for dictionaries, leaving locations 8192-16384 of Backing Store for the FORTRAN program. FORTRAN sub-programs are read in singly and this therefore allows a FORTRAN sub-program of approximately 48,000 characters to be translated without disturbing FEAT or the ALGOL compiler and to be subsequently translated and run without disturbing the ALGOL compiler. A FORTRAN program or sub-program is stored automatically from location 8192 upwards unless a different location is specified (see 6.1)

3.2 MTFEAT

MTFEAT requires one magnetic tape handler on which to store the FORTRAN programs. In addition, since MTFEAT is segmented, it is necessary to have either a second tape handler or one unit of core-backing store to hold the segments.

4. THE FORTRAN PROGRAM

4.1 Requirements of a FORTRAN program

It is essential that FEAT (or MTFEAT) should be used only to translate working FORTRAN programs, since in general the translator does not perform checking functions. It is possible, therefore, that a FORTRAN program will translate despite the fact that it contains an error, and produce a corrupt ALGOL version.

Before translation the FORTRAN sub-programs must, if necessary, be reordered to ensure that all Functions and Subroutines precede the main sub-program and that no sub-program is referred to by any other before it has been translated.

It is not possible to turn certain FORTRAN operations into correct ALGOL and a few restrictions must be observed (see Restrictions and Notes).

4.2 The FORTRAN Tape

With the exception of NO FORMAT, errors relating to mistakes in the FORTRAN listing are not detected, and it is advisable, therefore, to run a program punched from the FORTRAN listing through a FORTRAN precompiler, before it is translated. It is important that the version punched is an exact copy of the FORTRAN listing, including the correct number of initial spaces and spaces in the middle of a line. A group of six spaces may, however, be replaced by a tab. This facility is particularly useful at the beginning of a line, whenever the first six columns are blank in the FORTRAN listing.

Blanks occurring after the final non-blank character may be omitted unless they are part of a Hollerith string. This comprises an integer 'n' followed by an H, e.g. 36H, and n other characters. The n characters must in this case be punched even if they are all blank, unless the next line is a continuation statement when the trailing blanks need not be punched.

5. TAPES (Supplied only on request)

5.1 FEAT

The following tapes are required:

(a) FEAT
(b) Standard Procedures Tape 1
(c) Standard Procedures Tape 1c
(d) Standard Procedures Tape 2
(e) FORTRAN II Standard functions and Decode Tables
(f) FORTRAN IV Standard functions and Decode Tables

5.2 MTFEAT

The following tapes are required:

(a) MTFEAT
(b) Standard Procedures Tape 1
(c) Standard Procedures Tape 1c
(d) Standard Procedures Tape 2
(e) FORTRAN II Standard functions and Decode tables
(f) FORTRAN IV Standard functions and Decode tables
(g) MTSTORE

5.3 Standard Functions and Decode Tables

There are two tapes of Standard Functions and Decode Tables which provide for the translation of FORTRAN II and FORTRAN IV functions into their ALGOL equivalents. They are data tapes for FEAT and the appropriate tape must be input after FEAT has been translated (see 6.1). When FEAT is translating a FORTRAN sub-program the functions included on a Standard Functions and Decode tape are accepted as the following standard ALGOL functions:

FORTRAN II	FORTRAN IV	ALGOL Functions
ABSF	ABS	abs
SQRTF	SQRT	sqrt
SINF	SIN	sin
COSF	COS	cos
ATANF	ATAN	arctan
LOGF	LOG	ln
EXPF	EXP	exp
FLOATF	FLOAT	-

The function used to convert integers into floating point form in FORTRAN is not necessary in ALGOL, since it is possible to write a mixed expression. There is no equivalent ALGOL function to FLOATF and FLOAT, and FLOAT and FLOATF are therefore ignored.

If any additional FORTRAN functions are required they must be included on a FORTRAN Standard Procedures tape as type procedure declarations and input when the mnemonic ALGOL version of the original FORTRAN program is being translated (see Appendix I).

5.4 Standard Procedures tapes

5.4.1 GENERAL DESCRIPTION

(a) Tape 1 and Tape 1c

The procedures are written in ALGOL and are used, by the translated FORTRAN program, to input and output data in standard FORTRAN format. Tape 1 reads data from tape reader one and tape 1c reads data from an Elliott Card Reader punched in the format specified by the original FORTRAN program. The basic difference between the two tapes is that a different procedure "fill buffer" is used. If input is required from a different device, then it is necessary to replace the procedure "fill buffer".

(b) Tape 2

The tape has on it the following functions:

XINTF, INTF, TANHF, XABSF and TANH

It will be required, therefore, whenever one or more of these functions is used.

In addition to the above functions, the tape also caters for the FORTRAN IF(SENSESWITCH n) which will be replaced by if ng (n) then

5.4.2 FORTRAN FIELD DESCRIPTORS

The descriptors E, F, I & H are allowed and operate as described by the ECMA Standard on Fortran.

Other descriptors which are allowed are as follows:

A This allows alphanumeric characters to be input or output to or from a specified location. The characters are packed 6 or less to a word. If more than 6 characters are to be read then the first 6 are packed in the location and the remainder ignored.

O This causes input or output of a number in octal form.

X This causes, on input, n characters to be skipped and on output, n spaces to be punched.

/ This causes a new card to be read or a new line to be output.

5.4.3 OPERATING

See appendix I.

5.4.4 ERROR MESSAGES

Message output to typewriter	Reason
card oflow	An attempt has been made to access the 81st character from the input buffer.
line oflow (this message is only displayed by tape 1)	On filling the input buffer, from the paper tape reader, there were more than 80 characters on one line.
INCORRECT FORMAT	A field descriptor, not listed under section 3, has been used in a format statement.
data error 'n'	An error has occurred in the input of

data under descriptors A, E, F, I and O. Where n is the position of the next character to be accessed from the input buffer.

5.4.5 RESTRICTIONS AND NOTES

1. Tape 1, data is read into an 80 word buffer until a new line is read. If the data begins with a new line then this is treated as if a blank card had been read. After the new line is read then the remainder of the buffer is cleared. The characters blank and erase are ignored.
2. Logical field descriptors are not allowed and will give INCORRECT FORMAT error.
3. Field descriptor A packs a maximum of 6 characters starting from bit 36 down to bit 1.
4. If on output the number before the field descriptor H is a 1 then a new line is output and the next character is omitted.

6. TO TRANSLATE A FORTRAN PROGRAM

6.1 Method of Operating with FEAT

FEAT is an ALGOL program and may be input and translated by the usual methods (see 2.1.3 of the Manual). Operation will normally be as follows:

1. Type RESET.
2. Read in ALGOL tapes 1 and 2 from core backing store (read in the leader tape by typing IN).
3. Translate FEAT by changing the sign digit (to clear the Swait) or typing ALGOL.
4. (a) Set key 3 of the word generator if the lowest location of core backing store to be used is to be specified. Clear the Dwait (by changing the leftmost F2 digit) and type the lowest location of backing store to be used.

   or (b) Key 3 will normally be left clear - FEAT will then use locations 8192 upwards of core backing store automatically.
5. (a) if the FORTRAN program is to be read from cards clear key 2 of the word generator.

   or (b) If the FORTRAN program (sub-program) is to be read in from paper tape, set key 2 of the word generator.

6. Change the leftmost F2 digit to clear the Dwait and read in the STANDARD FUNCTIONS AND DECODE TABLES tape for either FORTRAN II or IV as appropriate.

7.(a) Clear key 1 of the word generator if a complete program is to be translated.

or (b) Set key 1 of the word generator if a single sub-program is to be translated.

8. Load the FORTRAN program in the card reader or tape reader (as specified at step 5) and clear the Dwait. The program is processed sub-program by sub-program and output as its ALGOL equivalent.

9. To translate a further program or sub-program:

(a) Clear the Dwait and repeat from step 7.

or (b) If a different setting of keys 2 or 3 is required, type REPEAT.

Operation will be repeated from step 4.

Note If ALGOL 3 is used for translation (see 2.1.5 of the MANUAL) it will probably be necessary to segment FEAT and allocate arrays to core-backing store.

6.2 Method of Operating with MTFEAT

Operation with ALGOL 1 will normally be as follows:

1.		type RESET.
2.	Read MTSTORE	type IN.
3.	Read ALGOL 1 tape 1 issue 1	type IN.
4.	Read ALGOL 1 tape 2 issue 1	change sign.
5.	Read either MOD 8 or MOD 9	type IN.
6.	Translate MTFEAT	type ALGOL.
7.	a) If the Fortran program is to be read from cards clear key 2. b) If the Fortran program is to be read from paper tape set key 2.	
8.	Read in Standard functions and decode tables tape	change dwait.
9.	a) If a complete program is to be translated clear key 1. b) If a single sub-program is to be translated set key 1.	

10. Load Fortran program (as specified at step 7). change dwait.
11. To translate a further program or sub-program
    a) repeat from step 9.
    b) interrupt and type REPEAT. to start from step 7.

At the end of step 5, a magnetic tape batch can be made which can be retrieved, when ever necessary, using RAPMT (see DUMP2M and RAPMT descriptions).

Note If ALGOL 3 is used for translation, core-backing store should be available to hold the CBS arrays.

Operation with ALGOL 3 will normally be as follows:

1. input an ALGOL 3 batch from magnetic tape
2. type ALGOLM.C. or ALGOLB.C. (depending on the type of ALGOL 3 batch - see 2.1.5 of the MANUAL) to translate MTFEAT
3. when "dwait" is displayed, interrupt, load MTSTORE in reader 1 and type IN.
4. (a) If the FORTRAN program is to be read from cards clear key 2.
   (b) If the FORTRAN program is to be read from paper tape set key 2.
5. Read in Standard functions and decode tables tape type ALGOLM.R. or ALGOLB.R. (see step 2)
6. a) If a complete program is to be translated clear key 1.
   b) If a single sub-program is to be translated set key 1.
7. Load Fortran program (as specified at step 4.) change dwait.
8. To translate a further program or sub-program
   a) repeat from step 6.
   b) interrupt and type ALGOLM.R. or ALGOLB.R. to start from step 4.

6.3. Output

The local declarations and ALGOL statements corresponding to the original FORTRAN are output on Punch 1. FORMAT statements are not translated but are copied and output as encountered between string quotes on Punch 2.

After the last sub-program has been translated a length of blank tape isoutput on both punches. This is followed by the output of a format count on Punch 1, the global declarations and a routine to read in the strings on Punch 2. These four tapes must be input for translation in the order specified in Appendix 1.

7. PROCESS USED TO TRANSLATE A FORTRAN PROGRAM

The FORTRAN program is input, one sub-program at a time. SUBROUTINES and FUNCTIONS precede the main sub-program. During input, lists are compiled of COMMON, EQUIVALENCE, DIMENSION, GOTO and DO statements, and of any variables which are declared as REAL, INTEGER or LOGICAL. FORMAT statements are not stored but immediately output between string quotes.

A list of all identifiers used is compiled for the local declarations. (Identifiers appearing in COMMON are naturally declared globally). The declarations of local variables are then output followed by the labels in the form of a switch declaration. Arithmetic Statement Functions are searched for, translated and output. The sub-program is output order by order after the standard substitutions have been made. Thus for example goto replaces GOTO, and

```
A:=C*sin(D);
```

replaces

```
A=C*SINF(D)
```

When the final instruction has been output, the next sub-program is input and the process repeated. When the final (main) sub-program has been processed, a declaration of the COMMON variables is output followed by a set of orders to reinput the FORMAT string.

8. ERROR MESSAGES

Certain operations which are permissible in FORTRAN cannot be translated into ALGOL. When such an operation is encountered, the translator (FEAT or MTFEAT) takes alternative action and outputs a warning message on the output writer:

2.2.3.29.
FEAT A

FORTRAN operation	Action taken	Message output
1. Two COMMON lists contain entries of diffcrent type or dimensions in the same part.	The translator allocates separate storage locations.	INCONSISTENT COMMON A, B Where A and B are the names of the two entries
	When the warning message has been output the translator searches the list of items which have been rejected from previous COMMON lists. If it finds another item which had been in the same part of a list, of the same type and dimensions as the newly rejected one it places them in the same location and outputs:	COMMONED WITH C
2. A formal parameter which also appears in a COMMON list.	The translator does not common the parameter with the other variables in the COMMON list.	FORMAL PARAMETER IN COMMON COMMON FACILITY IGNORED
3. Reference is made to a FORMAT statement whose label cannot be found.	The translator stops.	NO FORMAT

Note 1. Error 3 might occur for example as a result of a label being punched inaccurately and appearing in the continuation column on the FORTRAN card.

2. If an array is called with the wrong number of dimensions this will not be detected until the ALGOL version of the translated FORTRAN program is being run (see Appendix I).
Error No. 27 will be displayed.

9. RESTRICTIONS AND NOTES

When FEAT or MTFEAT is used to translate a FORTRAN program the following points must be observed:

1. A label which appears in more than one COMPUTED GOTO in FORTRAN will be doubly declared in the ALGOL block and cause an error when the ALGOL version of a FORTRAN program is being translated by the Elliott ALGOL compiler. The error is most easily eliminated by attaching a second label to the ALGOL statement and using the new label in the switch declaration.
2. Integer division is rounded not truncated.
3. The FORTRAN statements REWIND and ENDFILE are ignored by the translator.
4. The FORTRAN statement BACKSPACE is output as a procedure call.
5. READ TAPE n, LIST and WRITE TAPE n, LIST are translated as a call of setB and a call of Bout for each element in the list. Procedures for Bout and setB are not provided in FEAT or MTFEAT. A global variable boole is set to false or true according to which statement was used.
6. When FEAT is used IF (A.EQ.B) is not translated into correct ALGOL and will give an error on translation. This will occur when any Boolean IF is used. With MTFEAT, however, logical IF is translated correctly.
7. Boolean constants are not recognised.
8. EQUIVALENCE statements are ignored.
9. Named common lists are not correctly translated, therefore, all common data should be unnamed.

APPENDIX 1

TO TRANSLATE AND RUN THE ALGOL VERSION OF A TRANSLATED FORTRAN PROGRAM

The FORTRAN program is translated by FEAT or MTFEAT into standard ALGOL 60. Its ALGOL equivalent may, therefore, be translated by the normal operating procedure for an ALGOL program, except that the ALGOL program and its data are punched in parts which must be input in the order specified:

(I) To translate the ALGOL version of the FORTRAN program

1. The global declarations and initialisation statements which comprise the second part of the output on Punch 2 during translation of the FORTRAN program.
2. The requisite FORTRAN standard procedures tape(s). Any additional procedures that are required must also be input at this point.
3. The translated FORTRAN program which forms the first part of the output from Punch 1.

(II) To run the translated ALGOL program, data should be input in the following order:

1. The format count which formed the second part of the output on Punch 1.
2. The format statements which were output as strings forming the first part of the output from Punch 2.
3. Any data required by the FORTRAN program.

APPENDIX II

SUBSCRIPT OVERFLOW

During the translation of a FORTRAN program into ALGOL it is possible that subscript overflow may occur as a result of declaring arrays of inadequate size in FEAT.

The mnemonic FEAT tape comprises its title followed by array declarations separate from the rest of the tape by a few inches of blanks. If the set of declarations on the tape does not meet the requirements of the FORTRAN program to be translated, subscript overflow may occur during translation into ALGOL. If this is so a new set of declarations must be typed or the existing set altered.

The number of statements of each type allowed on the standard master tape is given in the table below. For each additional statement of any type that is required, the upper bound of the appropriate FEAT array declaration must be increased by the number indicated in the final column of the table, except for the statements FUNCTION and SUBROUTINE where the lower bound of the array should be decreased.

	Statement	array	standard	increases
1.	DIMENSION	arrd	2	1
2.	COMMON	arrc	2	1
3.	EQUIVALENCE	arre	2	1
4.	GOTO	ggot	11	1
5.	all identifiers	id	147	2
6.	FORMAT	fortab	48	2
7.	DO	ddo	11	1
8.	commoned identifiers	table 1	10	3
9.	FUNCTION) SUBROUTINE)	func	14	decrease lower bound by 2
10.	arithmetic statement functions	(arrd (arrc	2 2	1 1
11.	for each label in a computed GOTO	trad	30	1
12.	DO	trad	10	3

Whenever one of the error conditions listed above is detected, the message SUBSCR OFLO will be displayed.

2.2.3.29
FEAT A

During input of a FORTRAN sub-program an indication of subscript overflow can occur due to error conditions 1-7 and 9. Errors of types 5 and 9 only may be caused by sub-programs previously input, since in Error 5, the number of identifiers is the total for a whole program and Error 9 is caused by the total number of sub-programs in any one program.

If overflow occurs during input of the FORTRAN statements, the subscript overflow statement will have been caused by the last statement read. Examination of this will, therefore, give the error type.

After a sub-program has been read in, SUBSCR OFLO can then be caused by error type 8. begin is then output on Punch 1, and SUBSCR OFLO can then be caused by 'Error' type 10. After begin is output on punch 1, SUBSCR OFLO can be caused by error types 10 and 8.

Declarations of the standard identifiers are then output on Punch 1 and SUBSCR OFLO can then be occasioned by 'Error' 11.

When the program or sub-program is being output on Punch 1, the message SUBSCR OFLO will be due to 'Error' 12.

Note SUBSCRIPT OVERFLOW should not occur in the case of MTFEAT since the arrays are considerably larger.

CHAPTER 30: POST MORTEM LISTING ON LINEPRINTER

CODE PML S Issue 2

FUNCTION

A.a) To output the contents of a group of consecutive main store or Core Backing Store locations via the lineprinter as pseudo-instruction pairs, integers or octal numbers.

b) To display on the typewriter the address held by the Sequence Control Register at the time of the last Manual or Error Interrupt (with 1st/2nd half indication).

B. To provide the facilities of Aa) as a Common Program.

STORE USED

353 locations.

METHOD OF USE

FUNCTION A: Operating Instructions

(i) Enter PML by typing PML.

(ii) Either (a) Specify the style of output by typing S; where S = 0, 1, or 2 [pseudo-instruction, integer and Octal respectively].

or (b) Type MAN. or ERR. to obtain the address held by the S.C.R. at the time of the last Manual or Error Interrupt. If the interrupt occurred in the 2nd half of a location a colon ":" is placed

infront of the address (which is a 4-figure integer with no suppression of leading zeros). Go to step (iv).

(iii) Set the area of the store to be output by typing

F - L.

where F & L represent the addresses of the first and last locations of the area [see STORE ADDRESSES].

(iv) Once the required area or S.C.R. addr[illegible] has been printed a newline is output on the typewriter and the process may be repeated from step (ii). If a full-stop is now typed control will return to RAP.

Messages

LP MAN — is displayed on a newline on the typewriter and the program waits if the lineprinter is in a manual state when printing is about to start, or if the "paper low" warning is given during the run. The program will continue automatically when the lineprinter is taken out of the manual state, or the paper reloaded.

NOPROG — is displayed if a non-existent main store address is typed. It is then necessary to re-enter the program.

CBS — is printed on the lineprinter page whenever backing store locations are to be output.

Store Addresses

F and L represent the addresses of the first and last locations of the output area.

a) Main Store: F and L are the usual main store addresses.

b) Core Backing Store: F and L are each of the form

"x + A" ["+" separates x & A]
where x is an integer $0 \leq x \leq 15$
representing the number of complete blocks of 8192 backing store locations before locations F and L.
A is the remainder $0 \leq A \leq 8191$.

e.g. (i) Location 0 of backing store is expressed as 0 + 0

(ii) Location 8192 as 1 + 0

(iii) If the required area is 5197 to 17980 then
F = 0 + 5197 and L = 2+1596

Note: "+" is the only separator allowed between x and A.

Style Parameter S (Function A)

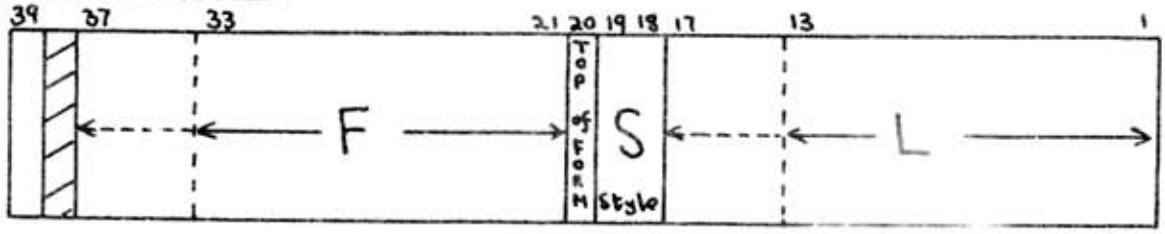
STYLE	S
Pseudo-instruction pair	0
Integer	1
Octal	2

FUNCTION B: Common Program

Entry write COMP, PML in the calling program, and set the accumulator as below.

Exit a standard exit is made with the accumulator clear.

Accumulator on entry:-



bit 39: Backing Store Indicator — If set F and L are interpreted as backing store addresses.

bits 37-21, 17-1: F and L — the first and last addresses of the output area. Backing store addresses can extend over a maximum of 17 bits.

bits 19-18: Style

19	18	
0	0	(i.e. <:00 0>) Pseudo-instruction
0	1	(i.e. <:20 0>) Integer
1	0	(i.e. <:40 0>) Octal.

bit 20 (B-digit): Top of Form Marker — If the B-digit is set the lineprinter top of form will be called on entry [see below].

Note: If F ≥ L only location F will be printed.

Top of Form and Alignment of Printing

Top of form will be called on entry if:

a) the B-digit is set in the accumulator,

or b) more than 50 locations are to be printed,

or c) if there is insufficient room to contain the area in the space left on the page.

Thus, if the B-digit is set on the first entry only, then on subsequent entries the program will automatically align the printing to avoid the page perforations. In the absence of a top of form call a 4-line gap is left between consecutive entries.

Examples of Common Program Entries:-

(i) To print locations 700 - 1500 of main store in Octal style enter with the accumulator set as:-

<00 700 : 40 1500>

(ii) locations 8000 - 9000 of backing store as pseudo-instructions:-

<40 8000 : 01 808>

(iii) locations 30 to 40 of backing store (first entry) as integers:-

<40 30/20 40>

FORMAT OF OUTPUT

200 consecutive locations are printed per lineprinter page in 4 columns of 50. A 2-line gap is left after every tenth location to form 4 groups in each column. The first location in each group has its address attached; a backing store address will be of the form "x + A" [see note below].

The lineprinter "top of form" is called at the start of each run (but see Function B) and at the end of each page.

Note: Octal Style — Since the Octal format on the lineprinter page leaves room for 4-figure addresses only, the "x+" portion of backing store addresses is suppressed during print-out.

RESERVED AREA

There is no need to lift the protection from locations 7936 - 8191 when printing their contents.

PROCESS USED

The backing store is accessed by single-word transfer. Any attempt to read from non-existent backing store addresses yields zero for the contents of the locations.

The S.C.R. values for MAN. and ERR. are obtained from locations 7909 and 7905 respectively.

CONFIGURATION

Basic 503 computer plus lineprinter (core backing store optional).

TAPE

A sum-checked relocatable-binary tape is provided which places the program from the first free location upwards.

TIME

2.5 minutes to print 8192 locations (at 1000 lines/min).

R. Gordon

December, 1965.

Chapter 31: CORE BACKING STORE ARRAY PROCEDURES

(Version 1)

CODE: CBSA A

GENERAL

On the tape are the following procedures:

- procedure bs array
- integer procedure takei
- integer procedure seti
- real procedure taker
- real procedure setr
- procedure clear
- procedure tobs

in the above order. The tape is intended to be read in before the main program. It should be noted that the title of the main program should be given in the form of a comment, except when using the precompiled version. One extra end must be included at the end of the program.

The tape ends with:

```
.........
procedures
.........
Ⓗ
print punch(3),££L?Type base?;
read reader(3),base;
Ⓗ
precompile;
Ⓗ
```

The integer "base" is the backing store address from which the backing store is intended to be used. In general, the value of base should be given in the main program, before the first procedures call, in which case the tape is not read between the first and second Halt-characters. If, however, it is required that the value of "base" is given though the typewriter, read the tape up to the second Halt-character. In this case, after the message "Type base" has been displayed, type the value of "base" (as an unsigned integer) and a semicolon, when the execution of the program continues. The instruction "precompile" enables a precompiled version of the tape to be made. In this case the tape is read to the end.

STORE USED

The whole procedure package occupies about 263 locations. If all the procedures are not used in the program, only the necessary parts need be copied from the tape. For this reason the procedures are separated by blanks.

PROCEDURE bs array

```
procedure bs array (B,low1,high1,low2,high2);
value low1,high1,low2,high2;
integer B,low1,high1,low2,high2;
```

FUNCTION

The procedure specifies a 2-dimensional backing store array, and reserves the necessary space for it from the backing store. The specification and space-reservation of every array must be performed separately by the procedure bs array.

PARAMETERS

B The backing store array ~~codeword~~ identifier, which must be declared as an integer variable in the main program before procedure calls.

low1 lower bound of the first subscript.
high1 upper bound of the first subscript.
low2 lower bound of the second subscript.
high2 upper bound of the second subscript.

After the procedure call, the integer B contains the following (underlined) integers. Restrictions:

(a) $-512 < \underline{low1} < 512$

(b) $0 < \underline{high2 - low2 + 1} < 16384$

(c) $0 < \underline{16384 + base - low2} < 32768$

PROCEDURES takei and taker

```
integer procedure takei (B,inx1,inx2);
value B,inx1,inx2;
integer B,inx1,inx2;
real ~~integer~~ taker (B,inx1,inx2);
value B,inx1,inx2;          procedure
integer B,inx1,inx2;
```

FUNCTION

These procedures are function procedures which take the value of the element with subscripts inx1 and inx2 of the backing store array B.

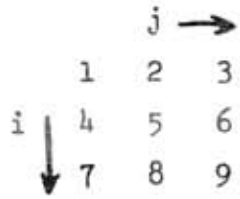
PARAMETERS

B	codeword of the backing store array
inx1	first subscript of the required element
inx2	second subscript of the required element

The time taken to get one element from backing store is about 620 micro-seconds.

EXAMPLE

The following integer array has been defined by the procedure call bs array (ABC,1,3,1,3) (i is the first subscript, j the second):



The instruction x:=takei (ABC,2,3);
puts the integer variable x equal to 6.

PROCEDURES seti and setr

```
integer procedure seti (B,inxl,inx2,R);
value B,inxl,inx2,R;
integer B,inxl,inx2,R;

real procedure setr (B,inxl,inx2,R);
value B,inxl,inx2,R;
integer B,inxl,inx2;
real R;
```

FUNCTION

These procedures are function procedures which place the value of the integer or real expression R in the element with subscripts inxl and inx2 of the backing store array B. At the same time the called procedure takes the value of R.

PARAMETERS

B	the ~~codeword~~ identifier of the backing store array
inxl	first subscript of the array element
inx2	second subscript of the array element
R	a number, variable or expression, to the value of which the element is set.

The procedures check the operation so that the element is placed in the backing store array at position [inx1,inx2], and is then immediately after this extracted, and if the two values differ,

CBS error x y R

is displayed, where x = 8,16 or 24.

The meanings of these values are:

8:	backing store is "busy"
16:	parity error
24:	both
y:	specifies the absolute address being used in the backing store.

After the message, the CBS parity error lamp goes out and the operation is tried again.

The time taken to store one element in backing store is about 750 micro-seconds.

EXAMPLE

The instruction:

m:=setr (XYZ,1,7,3.4);

places in the element with subscripts 1 and 7 of the backing store array XYZ, the value 3.4, and the real variable m is given the same value.

PROCEDURE clear

```
procedure clear (B,length,low2);
value B,length,low2;
integer B,length,low2;
```

FUNCTION

The aim of the procedure is to clear an array in the backing store, or in general to clear backing store, from the beginning of an array onwards.

PARAMETERS

B the ~~codeword~~ identifier of the array to be cleared (or the ~~codeword~~ identifier of the array from the beginning of which backing store is to be cleared).

length number of words to be cleared

low2 the lowerbound of the second subscript of array B (the same as in a call of bs array).

To clear more than one sequential array, give the parameter B of the array with the lowest required backing store address, its associated low2, and a sufficient value of "length".

EXAMPLE

If the integer variable base was given the value 0 at the beginning of the program, and the first backing store array specification was, for example

```
bs array (A,0,10,1,20);
```

then the procedure call

```
clear (A, 16384,1);
```

clears the whole one-unit backing store.

PROCEDURE tobs

```
procedure tobs(B,ARR,low2,DIRECT)
value B,low2,DIRECT;
array ARR;
integer B,low2;
boolean DIRECT;
```

FUNCTION

The procedure transfers an array from main store to backing store and vice versa.

PARAMETERS

B the ~~codeword~~ identifier of the backing store array

ARR the main store array

low2 the lower bound of the second subscript of array B (the same as in a call of bs array)

DIRECT true or false

If DIRECT has the value true, ARR is transferred from main store to backing store, if false the transfer is in the opposite direction.

The procedure tobs can transfer both integer and real arrays.

If at the time of transfer a parity error occurs, the procedure extinguishes the CBS parity error lamp, displays "BS parity" and tries to transfer again.

GENERAL NOTES

(a) The integer base is declared at the beginning of the procedure tape, and must not be declared in the main program again.

(b) The procedures are also suitable for use with one-dimensional arrays. In this case, at the places where low2, high1 or inx2 appear in the procedure calls, they are set to zero. (The parameters must not be omitted).

(c) When using the procedures, care must be taken that the subscript parameters of the procedure calls do not exceed the subscript bounds specified for the procedure bs array (because the ranges of the subscript are not checked in the procedures).

(d) Since the ALGOL compiler is placed in the beginning of backing store (about 8 k.), the integer variable base should be given the value 8192 unless the whole backing store is needed.

M. BEAT
FINNISH CABLE WORKS

NOVEMBER, 1965.

EXAMPLE PROGRAM

```
comment examples of using the procedures;
begin integer one,two,three,four,a,b,c,i,y;
      comment array codewords declared;
      real x;
      array AA [1:20];

      base:=8192;
      comment the procedure tape was only read as far as the
              first Halt-character;
      bs array(one,1,25,2,30);
      bs array(two,1,50,-25,25);
      bs array(three,1,20,0,0);
      bs array(four,1,100,0,0);
      comment backing store array bounds specified;
      clear(one,4000,2);
      comment backing store area cleared;
      x:=setr(one,7,29,sin(sqrt(347+45.31*0.03)));
      y:=(two,1,2,entier(ln(abs(x))));
      for i:=1 step 1 until 20 do
            begin read a;
                  b:=seti(three,i,0,a);
            end;
      c:=seti(two,1,3,takei(three,19,0)
            +takei(three,20,0));
      for i:=1 step 1 until 2 do
      AA[i] :=sqrt (i);
      lineprinter;
      print ££??,digits(8),y,b,c,freepoint(8),x,taker(four,20,0);
   end
end
end;
```

CHAPTER 32: ALGOL 1 CORE-BACKING STORE PROCEDURES: CBSB

(Version 2)

CODE CBSB A

GENERAL

The package contains the following procedures:

- procedure bsoflo
- procedure arrbs
- integer procedure takeil
- procedure setil

in the above order. The title of main program is given in the form of a comment, except when using the precompiled version. At the end of the program must be an extra end to go with the begin at the beginning of the procedure package.

The tape has the following form:

```
begin integer base;
-------
procedure
-------
(H)
print punch(3),ff1?Bs address?;
read reader(3),base;
(H)
precompile;
(H)
```

The integer "base" is the backing store address from which the backing store is intended to be used. In general, the value of base should be given in the main program, before the first procedure call, in which case the tape is not read between the first and second Halt-characters. If, however, it is required that the value of base is given through the typewriter, read the tape up to the second Halt-character. In this case, after the message "BS address" has been displayed, type the value of "base" (as an unsigned integer) and a semicolon, then the execution of the program continues. The instruction "precompile;" enables a precompiled version of the tape to be made, in which case the tape is read to the end.

STORE USED

The whole procedure package occupies about 101 locations.

PROCEDURES

1) `procedure bsoflo; print punch(3),££1?Bs oflo?,stop;`

A parameterless procedure which is used to display a subscript overflow message. After the procedure the program comes to a "stop".

2)
```
procedure arrbs(ARR,LOW,HI);
value LOW,HI;
integer ARR,LOW,HI;
```

Function

The procedure specifies a one-dimensional backing store integer array and reserves the necessary space for it from the backing store. The specification and space-reservation of every array must be performed separately by the procedure arrbs.

Parameters

ARR — The backing store array codeword, which must be declared as an integer variable in the main program before the procedure calls.

LOW — Lower subscript bound

HI — Upper subscript bound

"Bs oflo" is displayed if:

(a) base < 0
(b) LOW>HI
(c) base>16383

After the procedure call the integer ARR contains the following information:

(a) lower subscript bound
(b) upper subscript bound
(c) the absolute address of the first element of the backing store array.

3) integer procedure takeil(ARR,INX);
value ARR,INX;
integer ARR,INX;

Function

This is a function procedure which takes the value of the element with subscript INX of the backing store array ARR.

Parameters

ARR	The array codeword
INX	The element's subscript

"Bs oflo" is displayed if the subscript is out of the range specified in the procedure arrbs.

The time taken to get one element from backing store is about 580 µs.

4) procedure setil(ARR,INX,WORD);
value ARR,INX,WORD;
integer ARR,INX,WORD;

Function

The procedure places the value of the integer expression WORD in the element with subscript INX of the backing store array ARR.

Parameters

ARR	The array codeword
INX	The element's subscript
WORD	An integer constant, variable on expression whose value is put in the element.

"Bs oflo" is displayed if the subscript is out of the range specified in the procedure arrbs.

The time taken to store one element is about 550 μs.

Notes

(a) The integer "base" is declared at the beginning of the procedure tape, and must not be declared again in the main program.

(b) Since the ALGOL compiler is placed in the beginning of the backing store (about 8 k.), the integer variable base should be given the value 8192 unless the whole backing store is needed.

(c) These procedures are not to be confused with the procedure package CBS A.

Backing store array codewords defined using the procedure arrbs must never be used as parameters of the procedures takei, taker, seti or setr, and vice versa. If both packages are to be used in the same program, then a copy of the tapes must be made such that the integer base is only declared once.

M. KAISTI
FINNISH STATE,
COMPUTER CENTRE

NOVEMBER, 1965.

Example

```
comment examples of using procedures;
begin integer name1,name2,i;
      integer array aa [1:20], bb [1:10];
      comment declaration of the array codewords;
      arrbs(name1,1,20);
      arrbs(name2,1,10);
      comment specification of the subscript bounds;
      for i:= 1 step 1 until 20 do
      read aa[i];
      for i:= 1 step 1 until  20 do
      setil(name1,i,aa [i]);
      comment main store array aa stored in backing store;
      setil(name2,1,3);
      setil(name2,2,entier(sqrt(31.113*15.7.+3.33)));
      setil(name2,3,takeil(name1,3)*takeil(name1,7));
      lineprinter;
      digits(5);
      sameline;
      for i:=1 step 1 until 20 do
      print takeil(name1,i);
      for i:=1 step 1 until 3 do
      print takeil(name2,i);
   end
  end
end
```

The procedure tape was read up to the second Halt-character, so that the value of base is given through the typewriter;

CHAPTER 33: DOUBLE-LENGTH ADD AND SUBTRACT

(Subroutines)

CODE

DADD S
DSUB S

FUNCTION

To add or subtract two double length floating point numbers held in the manner described below in METHOD OF USE.

STORE USED

Program	111	locations
Workspace	5	locations

METHOD OF USE

i) The programmer must declare globally

1. Block names 'DADD' and 'DSUB' when using either 'DADD' or 'DSUB'.
2. data X1, X2, X, Y1, Y2, Y.

The data identifiers EX1, EX2, SIGN, DIFF have been declared globally by DADD and DSUB.

ii) Entry:

The numbers to be added or subtracted should be held in locations X1, Y1 and X2, Y2 upon entry. The respective exponents must be held in bits 1-9 of X1 and X2. The mantissa of X1, Y1 must be held in bits 39-10 of X1 and bits 38-1 of Y1 inclusive. The sign bit of Y1 must be clear upon entry. The numbers must be standardised.

The same holds for X2, Y2.

The instruction 'SUBR, DADD' causes entry to 'DADD'.

The instruction 'SUBR,DSUB' causes entry to DSUB which will subtract X2, Y2 from X1, Y1.

Exit:

On exit the required result is in location X,Y in the form described above.

PROCESS USED

DADD

The exponents are compared and if there is a difference of less than 67, the numbers are adjusted until their exponents are equal. The new mantissa are added together to form the mantissa of the result. The exponent is replaced and the number is standardised. If the difference in exponents is 67 or more, no significant addition can be performed and the larger number is taken as the result.

DSUB

The number in X2, Y2 is negated and DSUB uses DADD as a subroutine to add X1, Y1 and the new X2, Y2. Exit is made from DSUB with the result in locations X,Y.

Should the result of the addition or subtraction be too large to be represented in the manner described above in METHOD OF USE, a floating point overflow will be caused (See ERROR INDICATIONS). Should the number be too small to be represented or zero, X and Y will be cleared and exit made.

ACCURACY

The maximum error in the mantissa of the result is $\pm 2^{-67}$.

ERROR INDICATIONS

ERRINT 1 : Caused by floating point overflow. No continuation possible.

TAPES

The library tape is punched for input by SAP.

R. Gordon
H. Fingland

December, 1965.

CHAPTER 34: DOUBLE-LENGTH MULTIPLICATION

(Subroutine)

CODE

DMULT S

FUNCTION

To multiply two double-length floating point numbers held in the form described below in METHOD OF USE.

STORE USED

Program 69 locations
Workspace 5 locations.

METHOD OF USE

1) The programmer must declare globally

(i) Block name 'DMULT'
(ii) data X1, X2, X, Y1, Y2, Y

2) Entry: The numbers to be multiplied must be held in locations X1,Y1 and X2,Y2 upon entry. The respective exponents must be held in bits 1-9 of X1 and X2. The mantissa of X1, Y1 must be held in bits 39-10 of X1 and bits 38-1 of Y1. The sign bit of Y1 must be clear upon entry. The numbers must be standardised. The same holds for X2,Y2.

The instruction 'SUBR, DMULT' causes entry to DMULT.

Exit: On exit, the required result is in locations X,Y in the form described above.

PROCESS USED

The exponents of the numbers to be multiplied are added together and the mantissas contained in X1, Y1 and X2, Y2 are multiplied in 3 stages:

(i) X1 is multiplied by X2.
(ii) X1 is multiplied by Y2.
(iii) X2 is multiplied by Y1.

These solutions are added together in the appropriate way to form the mantissa of the result and the exponent is replaced. The number is standardised and exit is made.

Should the result of the multiplication be too large to be represented in the manner described above, a floating point error will be caused (See ERROR INDICATIONS below). Should the number be too small or be zero, X and Y will be cleared and exit made.

ACCURACY

The maximum error in the mantissa of the result is $\pm 2^{-67}$.

ERROR INDICATIONS

ERRINT 1 : Caused by floating point overflow. No continuation possible.

TAPES

The library tape is punched for input by SAP.

R. Gordon
H. Fingland
December, 1965.

CHAPTER 35: DOUBLE-LENGTH DIVIDE

(Subroutine)

CODE

DDIV S

FUNCTION

To divide two double-length floating-point numbers held in the form described below in METHOD OF USE.

STORE USED

Program	67	locations
Workspace	9	locations

METHOD OF USE

1) The programmer must declare globally

   (i) Block name 'DDIV'.

   (ii) data X1, X2, X, Y1, Y2, Y

2) Entry: The numbers to be divided must be held in locations X1, Y1 and X2, Y2 upon entry. X1, Y1 will be divided by X2, Y2. The respective exponents must be held in bits 1-9 of X1 and X2. The mantissa of X1,Y1 must be held in bits 39-10 of X1 and bits 38-1 of Y1 inclusive.
   The sign bit of Y1 must be clear upon entry. The numbers must be standardised. The same holds for X2, Y2.

   The instruction 'SUBR, DDIV' causes entry to DDIV.

Exit: On exit, the required result is in locations X,Y in the form described above, though Y holds only 9 bits after the sign bit - the rest being cleared.

PROCESS USED

The exponent of number X2, Y2 is subtracted from the exponent of number X1,Y1. The mantissas are modified to form rounded 39 bit mantissas in X1 and X2. X1 is then divided by X2 to give the 39 bit mantissa of the result. The exponent is replaced and the number is standardised.

Should the result be too large to be represented in the manner described in METHOD OF USE, a floating point error is caused (See ERROR INDICATIONS). If the result is too small to be represented or is zero, X and Y are cleared and exit is made.

ERROR INDICATIONS

ERRINT 1	Caused by floating-point overflow. No continuation possible.

ACCURACY

The maximum error in the mantissa of the result is $\pm 2^{-38}$.

TAPES

The library tape is punched for input by SAP.

ACKNOWLEDGEMENT

These subroutines were produced by converting Algol routines submitted by Mr. C. Plews of Reading University.

R. Gordon
H. Fingland

CHAPTER 36: ELLIOTT AUTOCODE TO ALGOL TRANSLATOR

CODE: AUTALG A

FUNCTION

To convert programs written in Elliott Autocode into ALGOL. The ALGOL programs produced may be run on the 503 or 803 with 8-channel paper tape equipment. After conversion to 4100 code, they may also run on 4100 ALGOL (issue 2), or, with manual editing, on issue 1.

PROGRAM AND MACHINE

The translator has been written in ALGOL and will run on the 503 ALGOL Mark I compiler. The computer must have 5 and 8 channel paper tape input and 8 channel paper tape output facilities.

SOURCE PROGRAM

The input text for the translator is an Elliott Autocode program which must be punched in 803 5-hole telecode.

FORMAT OF OBJECT PROGRAM

The output from the translator is ALGOL punched in 503 8-hole paper tape code. The layout of the translated program is:-

```
Title;
begin   (declarations)
        procedure program (ref); value ref; integer ref;
        comment ref holds the reference number at which
                program should be entered;
        begin switch ℓℓ := (list);
                  Ⓗ
              procedure READ;
              end  READ;
              (translated Autocode)
        end of procedures program;
        (initialisation)
end;
Ⓗ
switch lists of labels introduced during translation.
Ⓗ
```

N.B. The 503 and 803 ALGOL compilers require that all labels be declared in switch lists - see METHOD OF OPERATION, P.4.

(a) Some Autocode errors will be detected by the Translator - however many errors may be converted to incorrect ALGOL. Other errors might not be detected until the converted program is actually running.

(b) Certain machine dependent functions cannot be translated; in particular machine code blocks, MODIFY, VERIFY, FILM, INPUT or OUTPUT instructions. If any of these are detected a halt code (H) and blanks will be inserted in object tape and a message output on the typewriter.

(c) Integer arithmetic must not lead to results outside the range $-2^{23} \leq I \leq 2^{23}-1$, if the object program is to be run on the 4100 ALGOL.

(d) Care must be exercised if tests for equality of floating point numbers are used when running the object program on the 4100. Floating point numbers on 4100 are stored to 12 decimal places approximately, compared with $8\frac{1}{2}$ decimal places on the 803/503.

(e) A CYCLE Type 2 instruction (CYCLE A = p,q,r) is translated by an ALGOL for statement - hence it is not permitted to jump into the body of the loop.

There are no such restrictions on the other VARY and CYCLE Type 1 instructions.

(f) The SUBR instruction is translated by a recursive call on the procedure 'program'. For use on 4100 ALGOL (Issue 1) the statements used in the subroutine must be moved to the beginning of the program by editing manually and declared as a procedure. The recursive procedure call is then replaced by a normal procedure call. (This restriction will be relaxed with a later issue of 4100 ALGOL.)

(g) Triggers from data tapes and JUMP instructions leading to labels, specified at run time, are translated using switches. For use on 4100 ALGOL (Issue 1) the dynamic jumps (translated to "goto ll [i];") should be replaced by "ref:= i; goto trig;".

The switch following trig: "goto ll [ref];" should be replaced by the statement:-

```
if ref = 1 then goto l1
else if ref = 2 then goto l2
--------
(etc. for all reference numbers)
```

(h) Data tapes must be punched in the telecode of the object program and should obey the conventions of ALGOL. However all Autocode data tape facilities are available except the use of ? to cancel incorrectly punched numbers.

Labels on data tapes should be surrounded by = £ and ? instead of = and blank in Autocode.

(i) The translator produces an equivalent ALGOL object program. However it may be found that the object program can be improved by the programmer. For example, if none of the Autocode READ facilities except number input, then the procedure calls READ may be replaced by read statements.

METHOD OF OPERATION

The Translator is compiled in the usual way by the ALGOL system.

The source Autocode program is provided as data to the compiled program. The Autocode instructions are read in a line at a time and equivalent ALGOL statements output. Information for the programmer is output on the control typewriter during translation. After translation the computer will output switch lists containing any labels introduced during translation.

The object program may be compiled directly by the ALGOL system. During compilation of the declarations a systems wait will occur caused by a halt code (H) on the object tape. At this point any switch lists output after translation must be compiled before continuing the compilation of the program.

The object program is run as any other ALGOL program.

EXAMPLE

This example shows the Algol that is produced by translating a small Autocode program whose function is to read in a set of number pairs, A0 and A1, and print out a table of

$$\sqrt{A0^2 + A1^2}$$

Autocode

```
:: EXAMPLE PROGRAM
SETV   A(1)C
SETS   NI
SETF   SQRT
SETR   1
1)READ N
CYCLE  I=1:1:N
READ   A
READ   A1
A=A*A
A1=A1*A1
C=A+A1
C=SQRT C
LINE
PRINT  AO
PRINT  A1
PRINT  C
REPEAT X
STOP
START  1
```

Algol

```
EXAMPLE PROGRAM;
begin integer I,N;
      real    C;
      real array A [0:1];
                (STANDARD DECLARATIONS)
      procedure program (ref); value ref; integer ref;
      begin switch ℓℓ:=ℓ1, t0, exit, trig;
                (DECLARATION OF procedure READ;
      trig:     goto ℓℓ [ref];
      ℓ1:       READ(N,0,true,trig);
                I:=1;
      ℓ pl:     READ(0,A [0], false, trig);
                READ(0,A [1], false, trig);
                A [0]:= A [0] *A [0];
                A [1]:= A [1] *A [1];
                C:= A [0] +A [1];
                C:=sqrt (C);
      print £££??;
      print A [0], ££S2??;
      print A [1], ££S2??;
      print C;
      I:=I+1;
      if I ≠ N then goto ℓpl;
      stop;
      t0:exit: end of program;
      STANDARD INITIALISATION STATEMENTS
      read reader(3), start;
      if start = 0, then program(1)
                    else program(start);
end of translated Autocode program;
```

K.G. Robey
January, 1966

APPENDIX 1

Elliott Autocode and equivalent ALGOL

Note in examples below,

A,B,C,D	represent	floating point variables
I,J,K,L	represent	integer variables
m,n	represent	positive integer constants
p,q,r	represent	any integer constants
x,y,z	represent	floating point constants

Elliott Autocode	ALGOL	
Arithmetic		
A = B	A: = B;	
A = B + C	A: = B + C;	
A = B/C	A: = B/C;	
I = J * K	I: = J * K;	
Functions		
A = SIN B	A: = sin (B*pi);	
A = ARCTAN B	A: = arctan (B) / pi;	
A = LOG B	A: = ln (B);	
A = EXP B	A: = exp (B);	
A = SQRT B	A: = sqrt (B);	
A = INT B	A: = INT (B);	procedures INT
I = INT B	I: = INT (B);	and FRAC inserted
A = FRAC B	A: = FRAC (B);	by translator
A = MOD B	A: = abs (A);	
I = MOD J	I: = if J<0 then - j else J;	
A = STAND I	A: = I;	

Elliott Autocode	ALGOL
Setting and Start	
SETS I J (m)	integer I; integer array J [0:m];
SETV A B (m)	real A; real array B [0:m];
SETF INT FRAC	integer procedure INT(X); value X; real X; INT := sign(X) * entier (abs(X)); real procedure FRAC (X); value X; real X; FRAC := X- sign(X) * entier(abs(X));
SETF (other)	(ignored)
SETR n	procedure program (ref); value ref ; integer ref; begin switch ℓℓ:= ℓ1, ℓ2... ℓn, t0; goto ℓℓ [ref];

START n	t0: exit: end of procedure program;
	(initialisation statements)
	read reader(3), start;
	if start = 0, then program (n)
	else program(start);
	end of translated program;

Jumps	
JUMP @ n) JUMP n)	goto ℓn;
JUMP @ I) JUMP I)	goto ℓℓ[I];
JUMP IF A = B @ n) JUMPIF A = B @ n)	if A = B then goto ℓn;

Elliott Autocode	ALGOL
JUMP IF A ⅋ B @ I) JUMPIF A ⅋ B @ I)	if A > B then goto ℓℓ [I];
JUMP UNLESS A ⅋ B @ n) NO JUMP A ⅋ B @ n)	if not A < B then goto ℓn;

Other Controls

SUBR n	program (n);
EXIT	goto exit;
STOP	stop;
WAIT	wait;

Initial Values

SET A [m:n] = p,q...	goto ttm;
	tm: for pi:= p,q... do
	begin A [var] := pi;
	var:= var + 1;
	end;
	ttm:

VARY and CYCLE

VARY A = B:C:L	ctn := 0; A: = B; ℓpm:

REPEAT A	ctn: = ctn + 1;
	if ctn ≠ L then
	begin A: = A + C;
	goto ℓpm;
	end;

Elliott Autocode	ALGOL

CYCLE I = J:K:L	I: = J;
	±
	if I ≠ L then
	begin I: = I + K;
	goto ±
	end;

CYCLE A = B:C:D	A: = B;
	±
	if A ≠ D then
	begin A: = A + C;
	if abs (D+D-A-A) ≤ abs (C)
	then A: = D;
	goto ±
	end;

CYCLE A = x,y,z...	for A: = x,y,z... do begin
REPEAT A	end;

Input

READ A	READ (0, A, false, trig);
READ I	READ (I, 0, true, trig);
INPUT I	(not allowed)
READER I	reader (I); rdr: = I;

Elliott Autocode	ALGOL
Output	
PRINT A	print A, £ ?;
PRINT A,I	freepoint (I); print A, £ ?;
PRINT A,n/	scaled (n); print A, £ ?;
PRINT A,m:n	aligned (m,n); print A, £ ?;
PRINT I	print I, £ ?;
PRINT I, n	digits (n); print I, £ ?;
OUTPUT I	(not allowed)
LINE	print ££ℓ??;
LINES n	print ££ℓn??;
SPACES I	for sℓ:= 1 step 1 until I do print ££s??;
TITLE A£B?C bl.	print £A £q?B£u?C?;
PUNCH I	punch (I);
TELEPRINTER	punch (3);
Check	
CHECK A	A: = checkr (A);
CHECK I	I: = checki (I);

Chapter 37 : EDIT8 INTERFACE

CODE EDIT8I S

FUNCTION

To combine the functions of both EDIT8 and LPRINT so as to produce simultaneously an edited program tape and a listing of the new tape on the lineprinter. The alterations to the original tape are specified in exactly the same way as for EDIT8. LPRINT and EDIT8 are employed as common programs through the EDIT8 INTERFACE.

STORE USED

84 locations.

METHOD OF USE

The facilities provided are controlled by means of certain keys on the number generator. The keys are chosen so as to be adjacent to those used during translation of programs by the SAP assembler since EDIT8 may be used as a common program by SAP.

EDIT8 INTERFACE must be input after EDIT8. However, it need not be input immediately after EDIT8.

EDIT8 INTERFACE may be used in exactly the same way as EDIT8 inasmuch as entry points 1 and 2 have the same effect. There are, however, the following two differences when attempting normal operation:

1. Unless Key 32 is depressed, the edited tape is not produced. This enables the edit itself to be checked at high speed.

2. The edited tape is produced on punch 2 and not on punch 1 as is the case with EDIT8. This facilitates the parallel production of both binary and mnemonic copies of a tape using SAP (see COMMON PROGRAM USE).

The effects of the control keys are as follows:

Key 32 : Controls output to punch 2.
Output only occurs if the key is depressed.

Key 33 : Controls output to the lineprinter through LPRINT. Output only occurs if the key is depressed.

COMMON PROGRAM USE

Both the SAP and ALGOL compilers may be used in conjunction with the EDIT8 INTERFACE in the same way as EDIT8. With neither control key depressed the effect is exactly the same.

When either or both of the control keys are depressed the effect is as defined above.

SPECIAL NOTES

1. When EDIT8 INTERFACE is used as a common program, in order to obtain output of the last line of text on the lineprinter the operator must type:

   LPRINT ; 4 .

2. If Key 33 alone is depressed there must still be an edit tape ('RE') in reader 2 in order to obtain a lineprinter listing.

ERROR INDICATIONS

Message	Interpretation
ERRSUM	The tape produced has been mis-punched or has been mis-read when entry 2 is used to check the tape produced (see EDIT8 description). It is advisable to repeat the check read process to cater for the possibility of a mis-read rather than attempt to produce another tape immediately.
NoLPRINT	The common program LPRINT is not in store when Key 33 is depressed.

TAPES

The program tape is coded in a sum-checked binary form produced by SAP.

A. Woodcock (C.R.L.)

A.W. Porritt

May, 1966

CHAPTER 38 : MAGNETIC TAPE SORT (FIXED LENGTH)

CODE INTERN S

FUNCTION

To sort and update information written on a reel of magnetic tape into alphabetic or numeric order as determined by the user. This information must consist of standard length items packed into fixed length magnetic tape blocks. The magnetic tapes are handled using PCPP (see 2.5.3, Appendix 3).

CONFIGURATION

The basic 503 must be supplemented by a minimum of three magnetic tape handlers.

STORE USED

This varies with the code added by the user, but will be of the order of 1400 locations.

METHOD OF USE

(1) Entry Points

Entry to INTERN is a common program entry of the form:-

```
COMP, INTERN
:00 A
:00 B
:00 C
```

where A, B and C describe details of the layout of the tape (see (5) below for the general form of data on the tape). This entry initiates the sorting process and control will not be returned to the user until the process has been completed. The user is, however, required to write certain blocks for inclusion in INTERN itself (see below, TAPES) which will cater for the particular effects desired by the user. These SAP1 blocks are:-

```
COMPARE
ENDTEST
PREWRITE
FINAL
ENDFILE
```

and the first two are compulsory, while the others can be included as dummy blocks in many circumstances. The five blocks are all entered as subroutines at the appropriate stages of sorting, and their purpose and use will be described in (3) below.

It should be noted that an entry to INTERN is an indirect entry to PCPP, which is used in the transfer of buffers.

The procedure for running INTERN is to load PCPP, INTERN (complete), NAME (the user's program), and other programs required into main store, and type:-

PCPP. NAME; n.

where n defines the program entry point required. At this point handlers 1, 2 and 3 should be loaded with the original file and two scratch tapes, respectively. The original tape should be positioned immediately before the first physical block of data. At the end of the first pass, the message

END OF PASS 1

is output on the typewriter and a sign wait occurs. Handler 4 should be loaded before continuing: the original tape will not be used again. If only three magnetic tape handlers are available, then the number on the handler containing the original data should be changed to 4 at this point, and the reel changed if the original data is required. A message

SORTED DATA ON H 'N'

is displayed at the end of the sorting process: 'N' is the number of the handler onto which the sorted data has been finally output (i.e. N = 2,3 or 4). This integer N will be held in the accumulator on exit from INTERN.

(2) Parameters

The parameters following entry to INTERN have the following significance:-

A - Number of items per block on original unsorted reel.

B - Number of items per block on the sorted reel.

C - Number of words per item.

These figures are expected to remain valid throughout the length of a tape, and so must be constants.

(3) Writing the Supplement to INTERN

The existing blocks of INTERN deal with the mechanism of sorting, and not with the actual information content of items of data, which is presumed to be known only to the user. For this reason the user must supply the code for the blocks described, according to the following specifications:-

(a) Block "COMPARE".

Some part of each item contains the information forming the criterion for the item's position on the sorted tape (e.g. author, serial number, etc): this is known as the 'key' of an item. In order to function, INTERN must be told which of any two items should precede

the other. The addresses of the first words of the items to be compared will be in two locations with global identifiers "add1" and "add2" on entry to this subroutine. On exit, which should be made by the macro "EXIT,1", the accumulator should be set as follows:-

Positive	-	the item indicated by add1 is least.
Negative	-	the item indicated by add2 is least.
Zero	-	both have the same key.

This routine may reset the contents of any item but must not reduce the value of the key of any item.

(b) Block "ENDTEST."

INTERN incorporates a test for the end of the information written in the file, which should be indicated by a special item appearing at the end of the unsorted data. On entry to ENDTEST, the address of the first word of the item being considered is in the accumulator. The user's routine checks the item, setting the global variable "laststst" to +1 if the item is the last one on the file, then exits via the macro "EXIT,1".

(c) Block "PREWRITE."

Output of any item onto the sorted tape can be prevented during either pass (internal or merge, see below, PROCESS USED) by the block PREWRITE, which is entered each time an item is to be placed in an output buffer. As before, the address of the first word of the item is held in the accumulator on entry, so the user can check any relevant sections in the item (e.g. a marker written during the internal pass by COMPARE) in order to decide whether or not to output the item. The appropriate action is achieved by making one of two exits from PREWRITE : "EXIT,1" if the item is not to be written onto the sorted tape, or "EXIT,2" if it is to be written.

(d) Block "FINAL."

Before the last merge pass, FINAL is entered and may write a header block onto the new tape (whose handler number is in the accumulator on entry). If this facility is used, the block FINAL should include a declared vector as a buffer for the header block and output it using a PCPP macro instruction, before obeying "EXIT,1". On exit, the reel is assumed to be positioned immediately after the header block. If FINAL writes a block, it must increase the global variable "BLNO" by 1.

(e) Block "ENDFILE."

When INTERN reaches the end of the sorted data, this routine is entered and, if used, can ensure that nothing appears on the tape after the final item which may be confused with a genuine item, by writing end of file markers (e.g. -1's), either in the remaining locations of the last output buffer, or in an extra buffer should there be no such unused locations. The contents of the accumulator on entry will be 00 D:00 E, where D is the number of words required to fill the last buffer or part of buffer, and E is the address of the first location after the final item in the file (i.e. the first unused location of the buffer). The usual exit is made.

(4) Error Indications

If there is not sufficient room in main store for the necessary buffers, the message

?ROOM

is output on the typewriter. Any other errors occurring will be PCPP magnetic tape errors and are defined in 2.5.3 Appendix 3.

(5) Data Tapes

Four reels of magnetic tape are normally required during the operation of the program. Three of these are scratch tapes used to

store the contents of the file temporarily and of which one will be selected to hold the sorted information: the fourth is the original file.

The following points should be observed concerning the format of the data within a file:-

(a) The information is written in blocks of a standard length.

(b) Each block is composed of a number of complete items as specified by parameter A or parameter B (see METHOD OF USE above), with the exception only of the header block and the final block.

(c) Each item contains a standard number of complete words (parameter C).

(d) The tapes loaded on handlers 2, 3 and 4 should all have at least one block written on them, which could serve as a header block. Its content is ignored by both passes of the sort program.

TAPES

INTERN is supplied as a SAP1 mnemonic tape, including all the necessary global introductions, but excluding certain component sections as indicated by the stored entries in the following table.

Position on tape	Section	Description of Contents
1	INTERN	Declarations and main blocks
2	WAIT	Last block of INTERN's own routines
3	&	"Wait" marker for the assembler
4	&	"Wait" marker for the assembler
5	PREWRITE	Specialised routines used during the testing of INTERN. They should be repunched by the user.
6	COMPARE	
7	ENDTEST	
8	ENDFILE	
9	&	As for 3.
10*	FINAL	User's block.
11*	Trigger list	trigger MAIN;

Sections 5 to 8 inclusive are unlikely to be of general use: their effect is to output all items on the original file, to treat -1's as end of file markers, and to take the first word of each item as its key with bits 1 to 7 of the sixty-third word as a sub-key. (Note that an item is expected to contain at least sixty-three words.)

The user will probably need to substitute his own blocks in sections 5 to 8, as well as incorporating sections 10 and 11 on the tape, before assembling and running INTERN.

PROCESS USED

The original data is sorted into ordered strings by an internal pass, using handlers 1, 2 and 3, by a replacement-selection technique. After the keyboard wait, a merge pass is entered, using handlers 2, 3 and 4, which writes the sorted reel by a polyphase merge technique. (See the Communications of the Association of Computer Machinery, May 1963, for further details.)

CHAPTER 39: NUMERICAL INTEGRATION OF ORDINARY DIFFERENTIAL EQUATIONS

CODE NIODE A

FUNCTION

NIODE is an Algol procedure to perform one step of a step-by-step integration of a system of n simultaneous first order differential equations.

$$dy_i/dx = f_i\ (x, y_1, y_2 \ldots\ldots y_n)$$

$$i := 1,2, \ldots\ n$$

Any nth order differential equation can be written as a system of n first order differential equations - please see EXAMPLES OF USE at the end of this specification.

The method is a modification of a method based on Taylor Series to provide the best results possible with optimum stability (Taylor series itself is unstable) and applies to any system of differential equations with derivatives continuous or piecewise continuous with finite jumps in the range considered.

As own arrays cannot have variable bounds n, n has been restricted to $n \leq 20$. However, should the user require to solve N equations where $N > 20$, he has only to change the bounds of the own arrays in the setting instructions at the head of NIODE from 20 to the numerical value of N. The only bound on N is the available store size.

STORE USED

Approximately 1900 + 13*n locations

CONFIGURATION

The requirements are as for 503 ALGOL

TAPE

A mnemonic tape of the procedure is provided.

SUMMARY OF PROCEDURE

```
procedure     NIODE (n,X,Y,E,H,start);
      value   N,H, start;
      boolean start;  array Y;
      real    X,H;    integer n,E;
```

The parameters are:

n the number of equations

X the initial value of x=X for this step.

Y the array Y [1:n] which holds the initial values of y_i at x=X.

E the number of significant figures accuracy required.

H The maximum interval i.e. give initial values at x=X, then the procedure finds values at x=X+H.

start

start must be true on first entry to NIODE and then the starting process in NIODE sets up all initial values and also integrates one step from X to X+H. It is advisable to choose H such that there is no discontinuity in any of the derivatives in this interval H.
start must be false for subsequent entries and each entry integrates just one step from X to X+H.

On exit from the procedure:

X has been updated, and is now the initial value for the next entry

Y [i] holds the result y_i at this new value x=X

n,e,H, start

are unchanged in the procedure.

For subsequent entries, for ex, in tabulating results of one function, start must be false and there must be no discontinuity in the values of n,X or Y. H may vary for different entries, but if the sign of H is changed, the starting process must be re-entered i.e. start must be true. It is advisable to keep E constant for all entries, in fact, an increase in E may not be possible, since the values at the end of the first step are to the accuracy of E significant figures, and these form initial values for the next step.

METHOD OF USE

The user must declare globally

Y [1:n] on entry Y [i] holds the initial value of y_i

on exit Y [i] holds the result y_i at the new X

Yf[1:n]
Ff[1:n] These are used in procedure function
and
procedure function; relating the values of Ff [j] to Yf [i] for
j,i=1,2, ... n according to the differential equations

$$dy_i/dx = f_i\ (x, y_1, y_2 \ \ldots\ y_n)$$

For examples of how to write procedure function etc., please see EXAMPLES OF USE at end of this specification.

RESTRICTION ON DISCONTINUITIES

The method used applies to any system of ordinary differential equations with derivatives continuous or piecewise continuous with finite jumps in the range considered. There must, however, be no discontinuity in Ff [i] when Yf [i] = 0 in the range H, when start is true or false, since then the accuracy test has no real meaning. Also, it is advisable to choose H such that there is no discontinuity in the range H when start is true, since the starting process assumes that the functions are all well-behaved in this range.

ERROR MESSAGES

The following output is on channel 3, followed by a data wait.

(i) "Sign H changed".

This occurs if start is false, but the sign H has changed. On continuing, start is put to true in NIODE itself and the starting process is entered with this value of H.

(ii) "Discont. in n,X or Y"

This occurs if start = false, but the values of n,X or Y have been changed from those held at the end of the last entry to NIODE. On continuing, the new values are used, start is put to true in NIODE itself and the starting process is entered. This is equivalent to assuming that this is a different function.

(iii) "h tends to 0".

The elementary interval h used in increased or decreased during computation to provide the required accuracy in the minimum number of steps. Should h become very small compared with X, this warning is output, followed by a data wait. It is possible to continue, and such a case could occur around a discontinuity in one of the derivatives and by continuing the results should soon resume normal values and h increase again, but each time h is decreased such that $|h| < 5 * 10^{-8} * |X|$ the message is output.

(iv) Discont. F at Y = 0

This occurs if there is a discontinuity in one of the derivatives Ff [i] at Yf [i] = 0, in which case the accuracy test has no real meaning. On continuing, the message is output each time the accuracy test is entered until a value of h has been found to satisfy both tests. Results are not always possible, in which case "h tends to 0" will be output, or if results are possible, they may not be good.

ACCURACY

The user specifies the number of significant figures accuracy he requires. No instabilities are introduced by the method, but instabilities in the subject differential equations are reflected in the solution, and then the accuracy obtained is not necessarily the accuracy required. A discontinuity in any of the derivatives in the interval H when start is true or a discontinuity in Ff [i]when Yf [i] = 0 in the range H may lead to inaccurate results.

The 5th degree polynomial approximating the solution corresponds to a truncation error of the order of h^7, where h is the elementary interval size. The discontinuity error is bounded by

$$\tfrac{1}{2}|h\ (f(x_+) - f(x_-))|$$

The iteration error is of the order of h^8 and the roundoff error is minimised in the calculations.

TIME

The time taken depends on number of equations and also the equations themselves - see next paragraph.

PROCESS USED

The method is based on the article "On Numerical Integration of Ordinary Differential Equations" by Arnold Nordsieck, published in "Mathematics of Computation" January 1962. It is a "memory" method, operating with current values of the derivatives of a 5th degree polynomial approximating the solution. The method was obtained from a Taylor Series with the coefficients of the correction terms modified for the best accuracy possible with optimum stability. In actual fact, the method turns out to be a reformulation of Adam's method, because it uses effectively the same quadrature formula.

The method applies to any system of n first order differential equations with derivatives continuous or piecewise continuous with finite jumps in the range considered. It is thoroughly stable under all conditions, incorporates automatic starting and choice of elementary interval size. Any integration is started off given only the essential initial conditions i.e. values of X and Y [i]and the procedure function relating the y_i and f_i by the differential equations

$$dy_{i/dx} = f_i\ (x, y_1, y_2 \ldots . y_n)$$

$$i = 1,2,\ldots\ n$$

The starting process in NIODE (entered when start = true) chooses a slightly conservative value for the elementary interval size h, which is increased or decreased during computation to provide the specified accuracy of solution in the minimum number of steps. In fact, there are 2 tests in the procedure - one on the accuracy of that result just obtained and the other on the stability of the method as applied to the equations, and failure of either of these leads to a decrease in h to h/2.

EXAMPLES OF USE

1) An nth order differential equation to be written as n simultaneous first order equations.

Let $y_1 = y(x)$

$$y_2 = dy_{1/dx} = f_1$$

$$y_2 = dy_{2/dx} = f_2$$

$$\vdots$$

$$y_n = dy_{n-1/dx} = f_{n-1}$$

and then

$$dy_{n/dx} = f_n(x, y_1, y_2 \ldots y_n)$$

i.e.
$$dy^{n}_{/dx^n} = f_n(x, dy_{/dx}, d^2y_{/dx^2} \ldots d^{n-1}y_{/dx^{n-1}})$$

The $y_1 \ldots y_n$ are held in the array Yf [1:n] and the $f_1 \ldots f_n$ are held in the array Ff [1:n], and for this example, the procedure function is

```
procedure function;
    begin integer i, n1;
       n 1:=n-1;
       for i:=1 step 1 until n1 do Ff [i] := Yf [i+1];
       Ff [n]:= the differential equation itself;
    end;
```

2) The non-linear 2nd order differential equation

$$xy\ {dy^2}/{dx^2} + x\left({dy}/{dx}\right)^2 - 3y\ {dy}/{dx} = 0$$

is taken as an illustration of 1) above, and also to show a typical program using the procedure NIODE. Tabulate the values of y at x=1,1.5,2,2.5,3 to 5 significant figures given that at $x=1,\ y=2,\ {dy}/{dx} = 5.$ (The actual solution is $y^2 = 5x^4 - 1$, and from the initial conditions, we require the positive y values.)

Example of Use of NIODE;

```
begin real X; array Y, Ff,Yf [1:2];
      switch ss:=loop;
      procedure function;
      begin Ff [1]:=Yf [2];
            Ff [2]:=(3*Yf[1]*Yf[2]-X*Yf[2]*Yf[2])/(X*Yf[1]);
      end;
        (H)( feed in procedure NIODE here)
      sameline, freepoint(5);
      X:=1; Y[1]:=2; Y[2]:=5;
      print ££1?NIODE results £12s10? X Values £s11?
         Y Values £12s8??,X,££s9??,Y[1];
      NIODE (2,X,Y,5,.5,true);
      print ££1s8??,X,££s9??,Y[1];
loop : if (X-2.99999995) <0 then
      begin NIODE (2,X,Y,5, .5,false);
        print ££1s8??,X,££s9??,Y[1];
        goto loop;
end;
end of this example;
```

The results obtained were

NIODE results

X Values	Y values
1.0000	2.0000
1.5000	4.9308
2.0000	8.8882
2.5000	13.940
3.0000	20.100

and the time taken was about 6.3 secs, including punchtime.

Example 2) was repeated with the data x=1, y =-2,

$$dy/dx = -5$$ successfully.

3) The function with a discontinuous derivative (singular point),

$dy/dx = x$ for $x < 5$, $x > 5$ i.e. $y = x^2/2 + \text{constant}_1$

$dy/dx = 1-x$ for $x = 5$ i.e. $y = x - x^2/2 + \text{constant}_2$

was run successfully, with initial conditions x=3, y=4.5 and E=5 (i.e. 5 significant figures accuracy required), giving results

X Values	Y Values
3.0000	4.5000
4.0000	8.0000
5.0000	12.500
6.0000	18.000

The values of h, the elementary interval size taken, vary from a maximum permitted of 1.0 to approximately 10^{-4} at X=5, increasing again as X>5.

For this example, procedure function is

```
procedure function;
        if X=5 then Ff[1]:=1-X else Ff[1]:=X;
```

4) The function with a discontinuous derivative (step discontinuity),

$$dy/dx = x \qquad \text{for } x < 5 \qquad \text{i.e. } y = x^2/2 + \text{constant}_1$$

$$dy/dx = 1 - x \qquad \text{for } x \geq 5 \qquad \text{i.e. } y = x - x^2/2 + \text{constant}_2$$

was run successfully, with initial conditions x=3, y=4.5, E=5, giving results to 5 figure accuracy,

X Values	Y Values
3.0000	4.5000
4.0000	8.0000
5.0000	12.500
6.0000	8.0000

The values of h vary as in example 3) above, and for this example 4), procedure function is

```
procedure function;
    if X<5 then Ff[1]:=X else Ff[1]:=1-X;
```

5) N Simultaneous 1st order equations.

For example, the equations

$$dy_1/dx = y_1 - 4y_2$$

$$dy_2/dx = -y_1 - 2y_2$$

with initial conditions x=0, y_1=-2, y_2=3, were run successfully over the range x=0 to x=3. For this example, procedure function is

```
procedure function;
    begin Ff[1]:=Yf[1]-4*Yf[2];
          Ff[2]:=-Yf[1]-2*Yf[2];
    end;
```

The actual solutions are

$$y_1 = 2e^{-3x} - 4e^{2x}$$

$$y_2 = 2e^{-3x} + e^{2x}$$

P. Simmons
February, 1966.

CHAPTER 40: ALGOL MATRIX PACKAGE, MARK 2

FUNCTION

This package of procedures performs all the standard operations of matrix arithmetic.

COMPATIBILITY WITH MTX01A

This package is very similar in use to the 503 Libary Program MTX01A, but there are several differences namely:-

1) Procedures invmx and mxquot in MTX01A use Gaussian elimination with partial pivoting, whilst this package uses Crout reduction to reduce a matrix A to the form A = L (U+I), where L is a lower triangular matrix, the diagonal elements of L being the pivots of the reduction, and U is a strictly upper triangular matrix. This new method had, in some cases lead to more accurate results, but in most cases there is no difference in the results obtained. Using this new method introduces a few extra procedures, which are <u>not</u> recommended for use separately, but are called by invmx and mxquot etc, namely:-

crout	which performs the Crout reduction
permrows	necessitated by Crout
innerprod	performing the innerproduct of 2 vectors. (This is also called by mxprod).

2) solvmx is a new procedure which solves the equation AX=B, writing the result in B. A is also destroyed in the process. mxquot also solves the equation AX=B but it calls A by value, uses mxcopy to write B into X and then calls solvmx (A,X) working the result in X, thus both A and B are preserved at the expense of time and storage.

3) The determinant of the matrix being reduced by the Crout reduction can be calculated if the user wishes.

4) mxaux no longer exists, but is absorbed into procedures mxneq, mxcopy, mxsum and mxdiff.

5) The error messages given out are more explanatory.

6) This version is written in pure Algol and contains no optimization - MTX01A eliminates all actual array accesses by the use of Algol procedures address, size and location and hence is a lot quicker. However, this version does use Algol procedures lowbound and range and procedures formmx and innerprod use Jensen's Device.

For further details of the 5 main differences, please read the paragraph DETAILS OF PROCEDURES.

PARAMETERS

The parameters of the routines are, in general, arrays. The vectors and matrices which are used as actual parameters must have been declared in the main program as 2-dimensional arrays with appropriate subscript bounds. e.g. a column (or row) vector A should be declared as A [1:m, 1:1] (or A [1:1, 1:m]), and it is assumed that m > 1.

DETAILS OF PROCEDURES

Each procedure performs the tests for compatibility etc., which are necessary for the performance of the matrix operation intended. Error messages are displayed on punch (3) - the on-line typewriter.

The package has 2 global variables real det, innsum; innsum is used as workspace by procedure innerprod. det is used in procedure crout to calculate the determinant of the matrix being reduced by Crout's method. The calculation of det is optional - if det is zero on entry to the procedure crout, the calculation of the determinant is suppressed, but if det has any non-zero value on entry to crout the calculation is performed. det is set to zero by the package, so the calculation of the determinant is not performed unless the user sets det to any non-zero value in his program before calling invmx, solvmx or mxquot.

If the matrix being reduced is singular, a message to that effect is output on the on-line typewriter and the program enters a data wait. On continuing, the determinant is set to zero, and the actual calculation of it is not performed.

N.B. It is advisable when inverting large matrices with very large elements to leave det=0, and thus suppress the calculation of the determinant, because otherwise abs(det) could well be greater than 10^{77}, and cause floating-point overflow.

```
MATRIX PACKAGE:
begin real det,innsum;
procedure error; comment can be altered to suit users needs;
          stop;
procedure print1;
      print punch(3),£DIMENSIONS NOT COMPATIBLE£1?DIMENSIONS?;
procedure print2;
      print punch(3),£MATRICES NOT DISTINCT£1?DIMENSIONS?;
```

comment These 3 procedures are called when an error has been detected in the call of one of the procedures in the package.

for example 1

the call mxdiff (A,C,B) with the declarations
array A[0:2,0:2], B[1:3,2:4], C[1:4,0;2]

i.e. C has been declared of the wrong size for the matrix operation to be performed. This results in the following message output on the on-line typewriter.

MXDIFF ERROR DIMENSIONS NOT COMPATIBLE
DIMENSIONS 3*3, 4*3, 3*3

and the procedure error is entered.

The dimensions are output in the order of the parameters in the procedure call - i.e. in this example, in the order A,C,B.

N.B. mxquot calls mxcopy and solvmx, and thus the error messages are those of mxcopy or solvmx.

for example 2

the call mxquot (B,A,C)

where (1) B is a 3*4, A is a 3*4 and C is a 4*4 matrix

or

(2) B is corrected to be a 4*4 matrix like C, but A is still not square and hence has no inverse. The error messages given out will be

case (1) MXCOPY ERROR DIMENSIONS NOT COMPATIBLE
DIMENSIONS 3*4, 4*4

(these are the dimensions of B then C)

case (2) SOLVMX ERROR DIMENSIONS NOT COMPATIBLE
DIMENSIONS 3*4, 4*4

(these are the dimensions of A then B)

In both cases procedure error is called after the message has been output;

```
procedure mxdiff(a) becomes: (b) minus: (c);
        array a,b,c;
        comment   The result of subtracting array c from b is stored in a.
                  a may be the same as either b or c.  If the arrays are
                  incompatible "MXDIFF ERROR" is displayed, and the procedure
                  then calls procedure print 1, then prints the dimensions of
                  a,b,c in that order, and finally calls procedure error;

procedure mxsum(a) becomes: (b) plus: (c);
        array a,b,c;
        comment   The result of adding array b to array c is stored in a.
                  a may be the same as either b or c.  If the arrays are
                  incompatible "MXSUM ERROR" is displayed, and the procedure
                  then calls procedure print 1, then prints the dimensions
                  of a,b,c in that order, and finally calls procedure error;
```

```
procedure mxcopy(a) becomes: (b);
        array a,b;
        comment    This procedure copies a matrix b into a .  a may be the
                same as b.  If the matrices are incompatible "MXCOPY ERROR"
                is displayed, procedure print 1 is entered, the dimensions
                of a and b printed out in that order and finally procedure
                error is entered;

procedure mxneg(a) becomes minus: (b);
        array a,b;
        comment    This procedure negates a matrix b and writes the results
                in array a.  a may be the same as b.  If the matrices are
                incompatible "MXNEG ERROR" is displayed, print 1 is entered,
                the dimensions of a and b output in that order, and finally
                procedure error is entered;

real procedure innerprod(sk,tk,k,a,b);
        value a,b; real sk, tk; integer k,a,b;
        comment    This procedure calculates the sum over unit positive
                increments of k from a to b of sk*tk.  This procedure is
                used by mxprod, crout, invmx, solvmx and mxquot;

procedure mxprod(a) becomes: (b) times: (c);
        array a,b,c;
        comment    This procedure performs the matrix product of b and c,
                working the result in a, a must not be the same as either b
                or c.  If the matrices are incompatible, "MXPROD ERROR" is
                displayed, print 1 entered, the dimensions of a, b and c
                output, in that order, and finally the procedure error is
                entered.  Also if a is not distinct from both b and c then
                MXPROD ERROR is output, print 2 entered, the dimensions of
                a,b,c in that order output and then procedure error entered.
                The procedure calls innerprod;
```

```
procedure formmx(a) becomes: (X) with respect to: (I) and: (J);
      real X; array a; integer I,J;
      comment  This procedure forms a matrix a, where each element
               a[I,J] is a function X of I and J.  This forming is
               performed by rows.  This is done by use of Jensens device
               which allows a procedure to treat a formal parameter X as a
               function of the actual parameter corresponding to another
               formal parameter I.  (see E.W. Dijkstra's "A Primer of Algol
               Programming", Academic Press 1962 P. 57).
```

The actual parameters are:-

a — a 2-dimensional array

X — any real expression defining the value of the element a[I,J].

I,J — two integer parameters which specify resp. the row and column subscripts of the array element. The range of values of I and J are thus from lowbound (a,1) to range (a,1)-1+lowbound(a,1) and from lowbound(a,2) to range (a,2)-1+lowbound(a,2) respectively.

N.B. formmx (A,I+J,I,J) where A is declared as A[0:3,2:5] will result in a different matrix A from the same call of formmx where A is declared as [1:4,1:4] since in the 1st case I,J range from 0 to 3 and 2 to 5 resp. and in the 2nd case, I,J range from 1 to 4 and 1 to 4 resp;

```
procedure mxtrans(a) becomes transpose of: (b);
      array a,b;
      comment  This procedure forms the transpose of a matrix b and writes
               the result in a.  a may not be the same as b.  If the matrices
               are incompatible then MXTRANS ERROR is output, procedure print 1
               is entered, the dimensions of a  then be output and finally
               procedure error is entered.  If a is not distinct from b,
               MXTRANS ERROR is displayed, procedure print 2 entered, the
               dimensions of a then b output and finally procedure error entered;
```

```
procedure scprod(a) becomes a times the scalar: (X);
        value X, array a; real X;
        comment         This procedure multiplies a matrix by a scalar X in situ;

procedure permrows(a,r,lba1,lba2,n,m);
        value lba1,lba2,n,m;
        integer a; integer array r;
        comment     This procedure is necessitated by the procedure crout,
                and is called by invmx, solvmx and mxquot.  No description
                of it is given because it is not recommended for use separately;

procedure crout(a,entry,lba1,lba2,n,r);
        value lba1,lba2,n;
        array a; integer array r;
        string entry; integer lba1,lba2,n;
        comment     Performs as in situ reduction of a matrix into quasi-
                triangular matrices, U,L, by Crouts method with partial
                pivoting - the pivots being along the diagonal of  L - such
                that a=L(U+I).  No permutations of rows are performed and
                the pivotal row subscripts are recorded in the vector a
                [1:range(a,1)].  Because of this, this procedure is not
                recommended for use separately but it is called by invmx,
                solvmx, and mxquot.  (See P2 and 3 for remarks on det);

procedure invmx(a);
        array a;
        comment     This procedure inverts a matrix a in situ.  The method
                used is Crout reduction with partial pivoting. i.e. searching
                for pivots by columns only.  The procedure calls the
                procedures innerprod, permrows and crout.  If at any stage
                the ratio of the least pivot to date to the greatest pivot to
                date is less, in magnitude, than 10-6, the following error
                message is output.
```

"INVMX SINGULAR STAGE p SIZE n PIVRATIO q" and a data wait is entered. It is possible to continue after the data wait, but the results obtained are unlikly to contain any correct significant figures. (Also, floating-point overflow may possibly occur on continuing.) Failure at the 1st stage is specially distinguished. If a is not square, INVMX ERROR is displayed procedure print 1 is entered, the dimensions of a output, and procedure error entered. To use the facility to calculate the determinant of a, it is necessary to give any non-zero value to the real global variable det on entry. It is advisable in the case of large matrices with large elements to leave det=0, otherwise abs(det) could well become greater than 10^{77} and cause floating-point overflow;

```
procedure solvmx (a,b);
      array a,b;
      comment
```

Solves the equation ax=b working the result (x) in b, hence b is overwritten. a is destroyed too, being left in the crout reduction form. a may not be same as b. The procedure calls innerprod, permrows and crout and is itself called by mxquot. If a is not square, or if a and b are not compatible then SOLVMX ERROR is displayed, print 1 entered, the dimensions of a then b output and finally procedure error entered. Also, if a is not distinct from b, then SOLVMX ERROR is displayed, print 2 entered, and the dimensions of a displayed and finally procedure error entered. If at any stage, the ratio of the least pivot to date to the greatest pivot to date is less, in magnitude, than 10^{-6}, the following message is output and a data wait entered.

"SOLVMX SINGULAR STAGE p SIZE n PIVRATIO q." It is possible to continue but the results obtained are unlikely to contain any correct significant figures. (Floating-point overflow may possibly occur on continuing.) Failure at the 1st stage is specially distinguished. If the user wishes to have the determinant of a evaluated then he must set the global real variable det equal to any non-zero number before calling solvmx. However, it is advisable in the case of large matrices with large (positive or negative) elements to leave det=0 otherwise abs(det) could well become greater than 10^{77}, and cause floating-point overflow;

```
procedure mxquot (b,a,c);
        value a;
        array a,b,c;
        comment
```

This procedure solves equation ab=c. It calls a by value and uses mxcopy to write c into b thus a and c are preserved, and the result is in b. The procedure uses solvmx and thus crout, permrows and innerprod and also mxcopy. If b has not the same dimensions as c, then the message MXCOPY ERROR is output and print 1 entered, the dimensions of b then c output and finally procedure error is entered. If a is not square, or if a and b are not compatible then "SOLVMX ERROR" is output, print 1 is entered, the dimensions of a then b output, and finally procedure error is entered. If the user wishes to have the determinant of a calculated, then before calling mxquot it is necessary to give any non-zero value to the global real variable det. However, it is advisable in the case of large matrices with large (positive or negative) elements to leave det=0 otherwise abs(det) could well become greater than 10^{77} and cause floating-point overflow; Please read comment in solvmx on a being singular;

```
procedure readmx(a);
        array a;
        comment
```

This procedure reads any sequence of real numbers and places them in the array a. Successive elements are placed in the same row of a until that row has been filled, and then in following rows until array a has been filled. Data must be punched as described on P.6. Section 2.3.3.1 of the 503 Technical Manual. Reading takes place on the current device. The device setting may be altered before readmx is called:-

e.g.
```
reader(2)
readmx(a);
```

```
procedure printmx(a);
        array a;
        comment
```

This procedure prints an array a by rows. Each element is printed on the device and in the format current when the procedure is called. The format is specified before the procedure call.

e.g.
```
punch(2)
prefix(£,£s2??)
freepoint(4)
printmx(a)
```

Each rows is printed on a new line. No row or column numbers are printed,nor is there any facility included for printing large matrices. However, there are now some Additional Procedures on a separate tape to do this. Please see mxoutput etc.

METHOD OF USE

The package of procedures ends with the one instruction det:=0.0; and a H haltcode (telecode value 76) sign. The master program requires an extra end; and if it is fed in directly after the package of procedures, it must not have a title.

Any of these procedures may be replaced by others, or others included, as the user requires.

ACCURACY

Single length working is used.

TAPE

A mnemonic tape is provided.

STORE USED

If storage space is critical, unused procedures may be omitted, but care must be taken not to omit procedures which are called by the required procedures:-

All procedures call procedures error, print 1 and print 2.

mxquot	calls mxcopy, innerprod, permrows, crout, solvmx
solvmx	calls innerprod, permrows and crout
invmx	calls innerprod, permrows and crout
mxprod	calls innerprod.

PROCEDURE	APPROX STORE USED	INCLUDING
error)		
printl)	48	
print2)		
mxdiff	189	
mxsum	189	
mxcopy	142	
mxneg	141	
formmx	81	
mxtrans	227	
scprod	64	
mxprod	364	innerprod
invmx	1387	innerprod, permrows, crout
solvmx	935	innerprod, permrows, crout
mxquot	1079	mxcopy, solvmx (& thus innerprod, permrows, crout)
readmx	66	
printmx	73	

Any of these procedures may be replaced by others, or others included, as the user requires.

TIME TAKEN

This version is written in pure Algol and contains no omptimization; however, it uses Algol procedures lowbound and range, and procedures formmx and innerprod use Jensen's device.

	SIZE OF MATRICES		
PROCEDURE	5x5	10x10	20x20
formmx	Depends on the Function		
mxsum	.02 secs.	.11 secs	.46 secs
mxdiff	.02 secs	.11 secs	.46 secs
mxcopy	.02 secs	.08 secs	.32 secs
mxneg	.02 secs	.08 secs	.32 secs
mxprod	.14 secs	1.10 secs	8.30 secs
scprod	.02 secs	.08 secs	.32 secs
invmx	.26 secs	1.70 secs	11.25 secs
mxtrans	.02 secs	.08 secs	.32 secs
mxquot	Depends on the matrices		
solvmx	Depends on the matrices		
readmx	Full speed of reader		
printmx	Full speed of punch		

These times are of course, approximate.

ACKNOWLEDGEMENT

The procedures invmx, solvmx, crout and permrows are based on some procedures written by Mr. J. Boothroyd of the University of Tasmania, Hobart.

EXAMPLE OF USE

```
begin   array A[1:3,1:3],C,B[0:2,0:2];

        reader(1);
        readmx(A);
        reader(2);
        readmx(B);
        mxprod(C,A,B);
        det:=1.0;
        invmx(C);
        scaled(4);
        printmx(C);
        print ££1? det=?,sameline,det;
end;
end;
```

This small program is fed in immediately after the main package tape which stops on a H (haltcode) at the end of the tape. Output is on the current device.

ADDITIONAL PROCEDURES

In all cases where the number of columns is too great for the matrix to be printed on one level, either of the following output procedures may be used.

```
begin
procedure printcol(a);
        array a;
        comment   This procedure prints the matrix a by columns.  Each element
                  if a column is printed on a new line, then an extra 2 lines and
                  then the next column is printed, once again with each element on
                  a new line.  This procedure does not give such a good display
                  as mxoutput (below) but it is quicker and also takes up less
                  storage space.  Output is on the current device and in the
                  current format;
```

```
procedure spaces (t);
        value t;
        integer t;
        comment    This procedure prints t spaces.  It is used by mxoutput
                   below;

procedure mxoutput (A,m,n);
        value m,n;
        integer m,n;
        array A;
        comment    This procedure prints a matrix together with its row and
                   column numbers printed in the format digits(3).  It uses
                   procedure spaces above.  The procedure calculates from the
                   information given by parameters m,n the number of columns to
                   be printed across the page so that the array may be output
                   on 1 or more levels.
```

The parameters are:-

A	a real array
m	the number of characters available across the output sheet
n	the number of characters occupied by each element of the array.

The format for printing is set by the programmer.
From this the number of characters per element can be calculated. e.g.

freepoint (t)	t+2 characters are required per element
aligned(r,s)	r+s+2 characters are required of output formats. See 2.1.3.2 of the 503 Technical Manual, for further details of the character requirements of output formats;

METHOD OF USE

These procedures are on a separate tape from the main package, and a H (haltcode - telecode value 76) is punched at the end. When these procedures are required, this tape should be input immediately after the main package, and then there must be 2 extra end; on the master program.

STORE USED

PROCEDURE	APPROX. STORE USED	INCLUDING
printcol	78	
mxoutput	203	spaces

CONFIGURATION

Algol Matrix Package will fill the available store during translation, and hence facilities for storing the owncode version on Core Backing Store are desirable but not essential - depends on the size of the problem.

TAPE

A mnemonic tape is provided.

EXAMPLE OF USE

```
begin array A,B,C[1:24,1:24];
        reader(1);
        readmx(A);
        reader(2);
        readmx(B);
        mxprod(C,A,B);
        invmx(C);
        scaled(4);
        punch(2);
        mxoutput(C,90,10);
end;
end;
end;
```

First the main tape of the Matrix Package is fed in, then when this stops in a systems wait on the H (haltcode) at the end, the small tape containing these Addition Output Procedures is fed in. When this stops on the H (haltcode) at the end, the small program above is fed in.

P. Simmons
January, 1966

CHAPTER 41 : MAGNETIC TAPE ROUTINES (HANDLER 4)

CODE — MTSTOR S

FUNCTION — To write and read blocks to and from magnetic tape on handler 4. It is not possible to write further blocks once the reading back of blocks has occurred without destroying blocks previously written.

STORE USED — 200 locations including workspace.

METHOD OF USE — MTSTOR is written as a SAC common program. There are 3 entry points, and the standard SAC common program entry is made to all 3. Exit is always made by the instruction

```
EXITCP,1.
```

COMP, MTSTOR,1. This is the initialisation entry point of MTSTOR and should be entered before beginning to write blocks to magnetic tape. It is also necessary to make this entry before beginning to read back blocks which have been written to magnetic tape on a previous run.

COMP,MTSTOR,2. This causes entry to the second entry point of MTSTOR, and should be used when blocks are to be written to magnetic tape. It must be supplied with the block length and the address of the start of the block in main store. This is achieved by having an address (+S, say) in the accumulator upon entry, where location S and S + 1 hold:-

S) 00 0 : 00 start of block in main store

S + 1) 00 block length : 00 0

MTSTOR will write a block number into the more significant 19 bits of the 1st location of the block, so that it is not possible to store information in this portion of the location. MTSTOR numbers blocks consecutively, beginning at 1. The user should record the block number for himself as this is one of the parameters that has to be supplied when the block is to be read back.

COMP,MTSTOR,3. This entry isused to read blocks back. When entry is made, there should be an address (+S, say) in the accumulator , where location S and S + 1 hold
S) 00 0 :00 start of block in main store
S + 1) 00 block length : block number.

ERROR INDICATIONS

Message	Meaning and effect
H4 MAN	Handler 4 is in manual state. The program waits until the handler is released from manual and then continues.
H4 WNP	The tape on handler 4 does not have a write permit ring. The program waits until a write permit ring is inserted and then continues.
H4 EOT	The end of the tape has been reached. No continuation is possible.
BL TS	The length of a block to be written to magnetic tape is too small i.e. less than 4 locations. No continuation is possible
CANNOT WRITE	Displayed after several unsuccessful attempts to write a block to magnetic tape. No continuation possible.
BL NO TL	The block number of a block to be read is too large. No continuation is possible.
CANNOT READ	Displayed after several unsuccesful attempts to read a block from magnetic tape. No continuation is possible.
NOISE	A noise block has been encountered. The program continues.
LTH ER	A long or short block has been encountered. No continuation is possible.
S	Displayed when a short block which is not a noise block is encountered on reading. The program continues.

TAPES The program is coded in SAP1 and is not sum-checkable, so that when a binary version is produced, SAP1 should be entered with keys 35 and 36 of the word generator depressed.

General Purpose Software Group.

CHAPTER 42: ALGOL CARD READER PROCEDURES

GENERAL

The procedures are written for the ELLIOTT card reader. The following procedures are included:

- procedure card in
- integer procedure cdint1
- Boolean procedure descend
- integer procedure cdint2

in the given order.

At the end of the main program there must be an end; to go with the begin at the beginning of the procedure package.

STORE USED:

Approximately 268 locations.

PROCEDURES

1) procedure card in(S);
integer array S;

Function: The procedure reads a card, takes care of the buffering of the card reader and gives an error indication if the card reader is not available.

Parameters: S is an integer array, where the card will be placed. The subscript bounds of S are in general [1:80] (they can be wider, but the card is placed between these bounds). The actual parameter corresponding to S must be introduced in the program proper.

Global data: Boolean FSTCRD; is introduced at the beginning of the procedure package. It indicates whether the first card is to be read. At the end of the procedure package, immediately before the program proper, there is an instruction FSTCRD:=true; . The procedure gives it a value FSTCRD:=false; . The main program must not give it a value true thereafter.

Method used: The input is buffered. The card read into the own integer array Z [1:80] of the procedure during the previous call of the procedure is moved over to the array S immediately before reading a new card. The time to read in the card can thus be used for the calculation of the main program.

If the card reader is not available the procedure gives an error indication and waits until the operator has made the card reader available again. When the control word of the card reader indicates that the card reader is available again the program waits about 200 ms and then reads the next card into the array Z.

If the card to be read is the first card of the program, or if the control word of the card reader has indicated that the previous card has not been correctly read, then the information moved over into the array S will be discarded and the procedure will be started from the beginning again.

Error messages: The display (L) CRerrx means that the card reader is in an error state or is not available 10 ms after the last column of the previous card was input into the store.

x may have the following meanings:

x	bits of the control word	meaning	error
v	-	vacant	No cause for an error indication. The program is corrupted.
u	1	unavailable	The card reader is not on or is in manual state, there is no card in the reader, the stacker is full etc.
n	4	no error	The card reader has been busy more than 10 ms after the last column was read.
m	1,4	manual	Same as u and n together.
f	5	functional	A functional error of the card reader. May indicate that the card has not left the sensing platform. Otherwise like e (overleaf).

x	bits of the control word	meaning	error
e	1,5	error	The card last read in was not read in correctly. If the erroneous card has really gone through the reader, it must be put in the reader again together with the card on the sensing platform. If the card has jammed and not gone onto the sensing platform, it must be copied and the copy must be set into the reader. If there are no cards in the reader the next pack of cards must be set into the reader.
X	4,5	functional	Same as f and n together.
W	1,4,5	combined	Same as e and n together.

Note 1

x does not get the values X and W very often.

Note 2

The error indication may also be of the form (L) CRerrxX . Here the X means that the erroneous card has been discarded. The discarded card may be an empty "card" given by the card reader automatically in case the card remains on the sensing platform or the platform has become empty.

Such discarding error indications may also be caused by a card which has really gone through the reader, if the card reader has found any number of columns other than 80. The most usual of these error indications is L CRerreX . (See also the value e of x.)

2) integer procedure cdintl (S,A,B,error);
value A,B;
integer A,B;
integer array S;
label error;

Function: The procedure reads in integer from the card and gives this value to the main program.

Parameters: S is an array into which the card has been placed. A and B give the columns of the card between which (both the column A and the column B included) the integer is situated ($A \leq B$).

Allowing punchings: If there are one or more X-punchings (minus-punchings) in the columns A....B, the integer is negative. There may be a Y-punching (plus-punching) in any of the columns. The plus-punching is ignored. There must not be more than one numerical punching (0....9) in each column. If the procedure finds more than one numerical punching in any of the given columns, reading of the integer ends at once and the

procedure gives an error indication to the main program. Columns with no numerical punching are interpreted as columns with numerical punching 0. Thus the zero-punching may always be replaced by an empty column, X-punching with XO-punching etc. The units position of the integer must always be in the column B.

Error indications: The error indications mentioned before are given so that control is transferred immediately to the main program label corresponding to the formal parameter "error". If the absolute value of the integer to be read exceeds 274877906943 the ALGOL system may give the error indication Int oflo.

Time taken: Depends on the number of columns and the punchings of these columns. About 0.6 ms per column.

Note: The values of the elements of the array S are not destroyed.

3) Boolean procedure descend (SMALLER,GREATER);
value SMALLER,GREATER;
integer SMALLER,GREATER;

Function: The procedure is used by the procedure cdint2 to make a minimum-maximum-check. The procedure never causes integer overflow, and it clears the overflow indicator if set.

Parameters: SMALLER and GREATER are integers. The procedure gets the value true, if GREATER<SMALLER.

4) integer procedure cdint2 (S,A,B,MIN,MAX ,error);
value A,B,MIN,MAX;
integer A,B,MIN,MAX;
integer array S;
label error;

Function: The procedure reads an integer from the card, and gives this value to the main program.

Parameters: S is the array where the card has been put. A and B give the columns between and including which the integer will be read ($A \leq B$). MIN and MAX are the limits between which the integer should be. If the integer to be read is <MIN or >MAX, the procedure gives an error indication to the main program.

Error indications: The procedure uses the procedures cdintl and descend to make the maximum-minimum-check. The procedure cdintl gives to an integer C of the procedure cdint2 the value of the integer to be read. Then a jump is made to the label corresponding to the formal parameter "error" if descend (MIN,C) or descend (C,MAX) are true.

Note: The procedure cdint2 differs from the procedure cdintl only by having no maximum-minimum-check.

M. KAISTI,
FINNISH CABLE WORKS.

February, 1966.

EXAMPLE

The following example reads in a card and from it reads eight integers each occupying ten columns. It does this first by using the procedure cdint 1 and then by using cdint 2.

```
begin integer array  CD [1:80];
      integer COL1,COL2,min,max,I;
      switch s:= EL,ON;
      punch (1);
      goto ON;
EL:   print punch(3), £CARD ERROR?;
      stop;
ON:   cardin (CD);
      for COL1:=1 step 10 until 71 do
            begin COL2:=COL1+9;
                  I:=cdint1(CD,COL1,COL2,EL);
                  print digits (11),I;
            end;
      min:= -9010305070;
      max:= +9000045670;
      cardin (CD);
      for COL1:= 1 step 10 until 71 do
            begin COL2:=COL1+9;
                  I:=cdint2(CD,COL1,COL2,min,max,EL);
                  print digits (11),I;
            end;
end;
end;
```

Chapter 43 : ALGOL MAGNETIC TAPE PROCEDURES

CODE MTALG A

FUNCTION

The procedure package enables blocks of data, usually in the form of Algol arrays, to be read from or written to magnetic tape. The transfers are initiated by simple procedure calls in the main body of the program.

The package also allows the transfer of such blocks to be time-shared with other operations such as a block transfer to a different handler.

Selective overwriting of blocks previously written on a reel is not allowed.

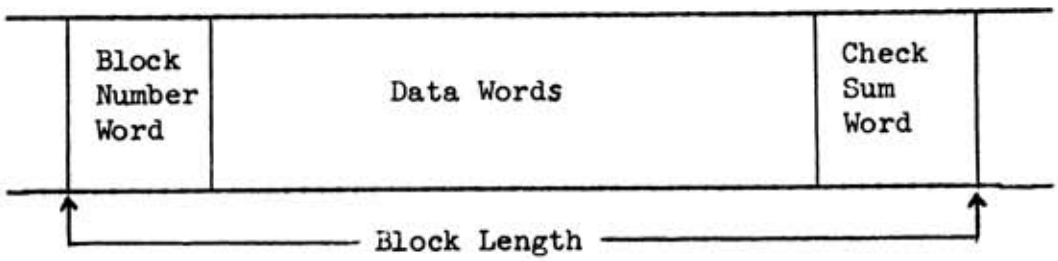
STORE USED

The package occupies approximately 700 locations.

DATA FORMAT

A block number is placed in the first word of each block written. The check sum of all the words in the block is placed in the

last word of the block. The form of each block is thus as follows:



The checksum facility is optional, and if it is not used the checksum word has the value zero.

Blocks are numbered consecutively from the beginning of the tape. The appropriate values for the block number and checksum are automatically placed before the block is written to magnetic tape.

The blocks are written using format 2, odd parity.

GLOBAL DECLARATIONS

The following variables are declared globally in the procedure package:

```
integer  curr, curr1, curr2, curr3, curr4,
         max, max1, max2, max3, max4,
         errs, errs1, errs2, errs3, errs4,
         param, param1, param2, param3, param4;
```

```
boolean  parity, wpremit, shortblock, longblock,
         endtape, begtape, allows, allowL, chsum;
```

The variables curr, max, errs and param are used as workspace by the procedures. They may also be used by the main program, but their values are lost when a procedure is called.

In the following description, the notation currP, maxP, errsP and paramP is used for the variables curr1, curr2, ... curr4, max1 .... max4 , etc., where P is the handler number, $1 \leq P \leq 4$.

currP - the number of the block which has just been read or written or is being read or written on handler P. The value is set by a call of readwd or writewd.

maxP - the number of the last block on handler P. When a block has been written, maxP is given the same value as currP.

errsP - the total number of rereads and the total number of rewrites which have occurred on handler P due to parity errors. The value is set by procedures rready and wready.

paramP - Used as workspace by the procedures, and must not be used by the main program.

All the global integer variables are set to zero by the procedure package. Their values should not be altered by the main program, except in the following cases:-

The variables corresponding to handlers not being used by the main program, may be freely used as Workspace. If a tape is known to contain n blocks written on a previous run, then maxP may be given the value n.

The procedures use the value of maxP to check that no search is made for a block whose number is greater than maxP. The values of the boolean variables parity, wpermit, shortblock, longblock, endtape and begtape are set by procedure control. Their values are not generally of interest to the main program.

The boolean variables allowS and allowL are set to false by the procedure package, and chsum is set to true. Their values may be altered by the main program if required, and the variables have the following meanings when true:

allowS - a short block indication when reading a block is not treated as an error, and the program continues normally.

allowL - a long block indication when reading a block is not treated as an error and the program continues normally.

chsum - the check sum calculated for data transferred to or from magnetic tape.

PROCEDURES AVAILABLE

The procedures called by the main program to read a block are readwd and rready, and to write a block are writewd and wready.

Transfers are time-shared with the main program, i.e. the data is transferred between main store and magnetic tape while the main program is running.

Data Input

```
procedure   readwd (length, address, number, P);
value       length, address, number, P;
integer     length, address, number, P;
```

Parameters:

length - The total length of the block to be read, including the block number and the checksum.

address - The address in main store to which the block is to be read, i.e. the block is read into locations address to (address + length -1) inclusive.

number The block number. If number ≤0, then the next block is read, i.e. currP + 1.

P The handler number.

If the next block on tape is to be read, i.e. number≤0 or number = currP +1, then no searching is required. Otherwise, the tape is moved backwards or forwards until block "number-1" is read.

The transfer to read the specified block is initiated and exit is made from the procedure, i.e. the main program continues while the transfer takes place.

```
boolean procedure rready (P);
value P; integer P;
```

The procedure is used to determine whether the read operation previously specified by readwd for handler P has been successfully completed. The procedure is given the value true on exit if the operation is successfully completed, otherwise false;

The main store area to which the block is read must not be used by the main program until the procedure has the value true.

If a parity error occurs, then up to 100 rereads are attempted, and if none of these are successful, or if a short block error has occurred and allowS is false, or if a long block error has occurred and allowL is false, then a error indication is given.

If chsum has the value true and there is no parity, shortblock or longblock error, then the checksum of the block read is calculated and compared with the last word of the block.

Data Output

```
procedure writewd (length, address, number, P);
value length, address, number, P;
integer length, address, number, P;
```

Parameters:

length - The total length of the block to be written, including the words for the block number and checksum.

address - The address in main store of the block to be written, i.e. the block is in locations address to (address + length -1) inclusive.

number - The number of the block to be written. If number ≤0, then the block is written as the next block on tape.

P - The handler number.

If the block is to be written as the next block on tape, i.e. number ≤0 or number = currP+1, then no searching is required. Otherwise, the tape is moved backwards or forwards until block "number -1" is read.

The transfer to write the specified block is initiated and exit is made from the procedure, i.e. the main program continues while the transfer takes place.

The procedure stores the block number in the first word of the block before it is written. If chsum is true, then the checksum is calculated and stored in the last word of the block, otherwise this word is set to zero. The value of maxP is not yet updated (see below).

```
boolean procedure wready (P);
value P; integer P;
```

The procedure is used by the main program to determine whether the write operation previously specified by writewd for handler P has been successfully completed.

The procedure is given the value true if the operation is successfully completed, otherwise false. The main store area containing the block to be written must not be used until the procedure has the value true. The value of maxP is only updated when this procedure has the value true.

If a parity error occurs, then an attempt is made to write the block again on the same area of tape. If still unsuccessful the cycle retreat, erase, write is obeyed up to 6 times, and if none of these are successful an error message is displayed.

The following note applies to calls of procedures rready and wready: If no operation has been previously specified for handler P, or if a previous call found that the operation was complete, the procedure is given the value true and exits.

Tape Administration

```
procedure conclude (P);
value P;  integer P;
```

The procedure is called by the main program when no further operations are required on handler P, and is called by the other procedures when an 'end of tape' or error message is displayed.

The procedure displays the following information on the output writer:

```
conclude
handler    P   currP     maxP
rereads    x
rewrites   y
```

x and y are the number of rereads and rewrites which have occurred on handler P. Their values are extracted from the variable errsP, which is then set to zero.

The values of maxP and currP are useful as a permanent record of what has been written on a tape, and perhaps for diagnosing an error.

The values of x and y give some indication of the state of the tape or tape handler being used.

```
procedure tmout (P);
value P; integer P;
```

The procedure writes a "tape mark" on handler P. A tape mark is a special block consisting of three words. The first word has the integer value -1, which is the identification of a tape mark if one is read. The procedure exits when the tape mark has been written. Since the tape mark has no block number the values of maxP and currP are not affected.

The procedure should be called when the last block on handler P has been written, so that the tape mark is the last block on tape.

```
procedure rewind (P);
value P; integer P;
```

This procedure rewinds the tape on handler P and sets currP to zero.

N.B. None of the other procedures included in the package may be called by the main program.

METHOD OF USE

General

When consecutive blocks are read or written, the actual parameter 'number' of readwd or writewd may be set to zero, and the main program need not be concerned with the block numbering system. This is the optimum and simplest use of the system, since there is no hold-up for searching. However, if it is required to read and/or write random blocks, the actual parameter should be the required value and any necessary searching is performed automatically.

It should be noted that this package cannot be used for selective overwriting, since the last block written is considered to be the last block on the tape (maxP is set to the number of that block).

Procedure Calls

Every call of readwd and writewd must always be followed by a corresponding call of rready and wready respectively for the same handler. Furthermore, the main store area used for the transfer must not be used by the main program until the appropriate call of rready or wready has the value true. The main program may continue calculating between these calls, and/or may call one of the procedures for another handler. The main program may continue similarly if rready or wready is found to be false, although this will not always be possible.

Procedure "conclude" may be called at any time, but it is recommended that it is called only when the operations on a particular handler are complete, which will usually be at the end of the program.

Procedure "tmout" should be called when the last block has been written on a tape.

Time Sharing

All transfers, including rereads and rewrites, are time-shared with the main program. Since there are two controller channels, two transfers may take place simultaneously. Procedure "select" is used internally to determine which channel to use. If both are busy, i.e. two transfers are in progress, then the procedure waits until one becomes unbusy.

A third or fourth transfer specified is not always held up in this way, it depends on the timing of the demand for a free channel.

Since the present procedure package does not queue or buffer blocks to be transferred, a call of readwd or writewd for handler P (except for the first call of the program) must not be given until a call of rready (P) or wready (P) for the previous operation has the value true.

End of Tape and Tape Marks

The package includes routines for deciding what action to take when the E.O.T. marker is reached or a tape mark is read.

To ensure that a programming error does not cause the tape to unload off the end, which will damage the tape and may damage the handler, it is highly recommended that the following rules are used as standard practice:

- Every magnetic tape for use by MTALG is previously prepared so that there is a tape mark at the end of the tape, well past the E.O.T. marker to allow for a long block to be written.

- After the last information block has been written, a tape mark is written using procedure tmout.

It is the operators responsibility that the correct tapes are used, and the programmers responsibility to write a tape mark as the last block on tape.

The procedure assumes that the tapes are written as above. If they are not, it is the programmers responsibility to ensure that the tape does not unload off the end or that apparent magnetic tape errors occur, due to a programming error.

The table below shows what action is taken when a tape mark is read and/or E.O.T. is reached.

Procedure	Situation	Action
search	Advancing or reading in order to find a block, E.O.T. is reached	Displays the message 'end of tape' calls procedure conclude and waits. After the wait the program continues as though the interruption had not occurred
search	Tape Mark read	
rready	E.O.T. is reached	
rready	Tape Mark read	
wready	E.O.T. is reached	
search	Tape mark read and E.O.T. reached	Displays the message 'prog err4' and stops. Continuation is not possible.
rready	Tape mark read and E.O.T. reached	

Thus the procedure can be used to continue writing blocks from the end of one tape to the beginning of another, displaying a message to the operator when a new tape is to be loaded. The value of the variable "endtape" is of no interest to the main program.

If a tape mark is written as the last block on a tape, then it is possible to read a tape which contains an unknown number of blocks, since an indication is given when the tape mark is read.

The operator must know what action to take when 'end of tape' is displayed.

If consecutive blocks are read or written, then no special precautions need be observed when going over the end of one tape to the beginning of another. But if random blocks are searched for, it should be realised that it is not practical to return to the first tape from the second.

Therefore, if procedure search is giving retreat orders and reaches the B.O.T. marker, an error message is displayed.

Note: When continuing from one tape to another, the second and subsequent tapes must be loaded at B.O.T.

SUMMARY OF PROCEDURES

The tape has the following form:

```
comment  MTALG;
begin  integer  curr, curr1, curr2, curr3, curr4,
                max, max1, max2, max3, max4, errs,
                errs1, errs2, errs3, errs4, param,
                param1, param2, param3, param4;
       boolean  parity, wpermit, shortblock, longblock,
                endtape, begtape, chsum, allowS, allowL;
       procedure display
       procedure delay
       procedure conclude
       procedure control
       procedure select
       procedure search
       procedure readwd
       procedure writewd
       boolean procedure wready
       boolean procedure rready
        comment initial settings;

       (H)

       precompile;

       (H)
```

The main program must include one extra end to go with the begin at the beginning of the procedure package.

ERROR INDICATIONS

All messages are displayed by procedure "display". The message is followed by the information: handler P currP maxP

The following control messages are followed by a wait:

Message	Procedure	Reason	Action
unavailable	control	Handler P is not available	Put handler P in the available state and continue
no wpermit	writewd	Writing is not permitted on handler P	Insert the write permit ring and continue
end of tape	search wready rready	EOT has been reached or a tape mark read	According to programmers' instructions

Note: It may be found that a spurious 'no wpermit' message occurs when continuing after 'unavailable'. The message may be ignored.

The following error messages are displayed when a programming error is detected. They are followed by a stop and continuation is not possible.

prog err1	The number of a block to be read is greater than maxP or the number of a block to be written is greater than maxP+1.
prog err2	Procedure search is retreating in order to search for a block and B.O.T. is reached
prog err3	readwd or writewd called when the corresponding call of rready or wready has not yet been given the value true
prog err4	Tape mark is read and E.O.T. is reached (i.e. the tape mark written when the tape was prepared)
prog err5	The parameter P, the handler number, is outside the range $1 \leq P \leq 4$.
prog err6	The block after block "number -1" is not block numbered. A wrong tape is being used or the program is corrupt. This error message will also occur if a negative block length is given as an actual parameter

The following messages are displayed when a magnetic tape error, possibly caused by a programming error, occurs. Procedure conclude is called after the message has been displayed, followed by a wait.

parity, writing	On the seventh attempt to write the block, a parity error occurred
parity, reading	On the 100th attempt to read the block, a parity error occurred
short block	The block read was shorter than specified and "allowS" is false
long block	The block read was longer than specified and "allowL" is false
check sum error	The check sum of the block read does not agree with the checksum value in the last word of the block. This will occur if there is a data error or if an attempt is made to check a block which was not sumchecked when written

The message 'cant write TM' is displayed if procedure tmout cannot write the tape mark because a parity error occurs. The writing is attempted twice. The program continues automatically after the message, although a parity error will occur if it is attempted to read the tape mark. The situation should only occur if the tape is damaged.

If the program is continued after one of the magnetic tape error messages, the procedure rready or wready, which detected the error,

will exit with the value true. The value of maxP will be set to the appropriate value by wready, even though an error has occurred.

If it is required to continue after an error, which will not usually be the case, the main program should check whether an error has occurred after the call of rready or wready has been given the value true.

- After a parity error, the boolean variable parity has the value true.

- After a short block or long block error the boolean variable shortblock or longblock has the value true, though a message is only displayed if allowS or allowL respectively is false.

- After a checksum error, the boolean variable chsum has the value false.

Note: The procedures do not check the values of the actual parameters length and address, and it is the programmers responsibility to ensure that the program does not get corrupted. The procedures cannot detect the following programming error: if a call of readwd for handler P is followed by a call of wready (P) or conversely a call of writewd is followed by a call of rready (P), then the result is undefined.

EXAMPLES

In the following examples, all blocks are sum-checked. The number of words of data is two less than the total length of the block.

Example 1

Versions a, b, c and d all solve the same problem to write 100 blocks of length 52 words on handler 1, the data being read from paper tape.

Program 1a demonstrates the simplest and most obvious method, from the programmer's point of view, of solving the problem.

Program 1b uses a similar method, except that the address of the array is calculated once at the beginning of the program, instead of every time a block is written, and the block count in curr1 is used instead of a for loop. The actual parameter number is set to zero, since consecutive blocks are being written.

Programs 1c and 1d are optimised so that the data is read from paper tape and the previous data written to magnetic tape at the same time. In both cases this is done by buffering the blocks i.e. two arrays are declared and one array is being transferred to magnetic tape while the next block of data is being read to the other array.

Note that program 1c and 1d declare a procedure word, which is called in place of a statement like: L:if not wready (1) then goto L;

If several of these statements occur in a program, it is simpler and more efficient to declare the procedure.

```
comment MTALG example 1a;
begin array a[0:51];
        integer i,j; switch s:=L;
        for i:=1 step 1 until 100 do
        begin for j:=1 step 1 until 50 do read a[j];
                writewd (52,address(a),i,1);
              L:if not wready (1) then goto L
        end;
        tmout (1); conclude (1);
end example;
end MTALG;
```

```
comment MTALG example 1b;
begin array a[0:51];
        integer j,A; switch s:=L,L1;
        A:=address(a);
     L1:for j:=1 step 1 until 50 do read a[j];
        writewd (52,A,0,1);
      L:if not wready (1) then goto L;
        if curr1 ≠ 100 then goto L1;
        tmout (1); conclude (1);
end example;
end MTALG;
```

```
comment MTALG example 1c;
begin array a,b[0:51];
        integer j,A,B; boolean x;
        switch s:=L;
        procedure wend;
        begin switch s:=L;
              L:if not wready (1) then goto L
        end;
        A:=address(a); B:=address(b);
        x:=true;
      L:for j:=1 step 1 until 50 do
        if x then read a[j] else read b[j];
        wend;
        writewd (52,if x then A else B,0,1);
        x:=not x;
        if curr1 ≠ 100 then goto L;
        wend;
        tmout(1);
        conclude(1);
end example;
end MTALG;


comment MTALG example 1d;
begin array a,b[0:51];
        integer j,A,B; switch s:=L;
        procedure wend;
        begin switch s:=L;
              L:if not wready (1) then goto L
        end;
        A:=address(a); B:=address(b);
      L:for j:=1 step 1 until 50 do read a[j];
        wend;
        writewd (52,A,0,1);
        for j:=1 step 1 until 50 do read b[j];
        wend;
        writewd (52,B,0,1);
        if curr1 ≠ 100 then goto L;
        wend;
        tmout(1);
        conclude(1);
end example;
end MTALG;
```

Example 2

The program demonstrates how the transfers can take place simultaneously. One block is read from handler 1 and one block from handler 2 at the same time, and then the results are calculated. Note that in this example the transfers are not time-shared with the calculation because buffering is not used, although the transfers are time-shared with each other.

```
comment MTALG example 2;
begin integer array a,b[0:1001];
        integer A,B; switch ss:=L1;
        max1:=max2:=100;
        A:=address(a); B:=address(b);
      L:readwd (1002,A,0,1);
        readwd (1002,B,0,2);
     L1:if not (rready(1) and rready(2)) then goto L1;
        comment calculation and printing of results;
        if curr1 ≠ 100 then goto L;
        conclude(1); conclude(2);
end example;
end MTALG;
```

Example 3

The program demonstrates how four handlers can be used. Random blocks of length 52 words are read from handlers 1, 2 and 3, then the blocks are sorted in the required order and written as a block length 152 words on handler 4.

The number of the blocks to be read are read from paper tape. Since random blocks are read, the program is held up while the necessary searching takes place, so it is unlikely that the transfers will be held up due to both controllers being busy. The previous transfers will probably be completed, i.e. a controller channel becomes free, while the searching takes place.

```
comment MTALG example 3;
begin integer array a,b,c[0:51], d[0:151];
        integer n1,n2,n3,A,B,C,D;
        switch ss:=L,L1,L2;
        A:=address(a); B:=address(b);
        C:=address(c); D:=address(d);
        max1:=max2:=max3:=500;
      L:read n1,n2,n3;
        readwd (52,A,n1,1);
        readwd (52,B,n2,2);
        readwd (52,C,n3,3);
     L1:if not (rready(1)and rready(2) and rready(3) and rready(4))
        then goto L1;
        comment sort a[1] to a[50], b[1] to b[50] and c[1] to c[50]
                in the required order into d 1  to d 150 ;
                 writewd (152,D,0,4);
                 if curr1 ≠ 500 then goto L;
               L2:if not wready(4) then goto L2;
                 tmout(4);
                 for n1:=1 step 1 until 4 do conclude (n1);
end example;
end MTALG;
```

Example 4

The program demonstrates how the time can be used when waiting for the procedure rready to be given the value true. In most programs this time cannot be used, since rready is called when the data is required. Blocks of length 102 words are read from handler 1 and printed on the lineprinter, and blocks of length 52 words are read from handler 2 and the data used for the calculation.

Since these operations are quite independent of each other, there is no danger of getting out of step.

```
comment MTALG example 4;
begin integer array a[0:101];
        array b[0:51]; integer A,B;
        boolean end1,end2; switch ss:=L;
        A:=address(a); B:=address(b); max1:=max2:+200;
        readwd (102,A,0,1);
        readwd (52,B,0,2); end1:=end2:=false;
      L:if not (end1 and end2) then
        begin if rready (1) and not end1 then
                  begin curr1 ≠ 200 then readwd (102,A,0,1) else end1:=true
                  end
                  if rready (2) and not end2 then
                  begin comment perform calculations on array b
                                and punch results;
                     if curr2 ≠ 200 then readwd (52,B,0,2); else end2:=true
                  end;
                  goto L;
        end;
        conclude(1); conclude(2);
end example;
end MTALG;
```

M. BEAT

FINNISH CABLE WORKS

MAY, 1966

CHAPTER 44: GENERAL PURPOSE ROUTINES

CODE CP S

FUNCTION. To combine a set of useful routines into one program.

CONFIGURATION Basic 503.

SPACE OCCUPIED PROGRAM 127 locations
DATA 10 locations

METHOD OF USE To enter and run any of the routines
LIST, SET, PRINT, ALINK, CANCEL, GET and DUMP
which are described below.

OPERATING To use any of the routines provided assuming that CP is in store, type CP. NAME.
where NAME is the name of the routine to be entered. The routines are as follows:-

1. LIST

Function

To give the user a break-down of the store.

Space occupied

PROGRAM	23	locations
DATA	1	location

Method of use

when 'LIST' is entered by typing CP.LIST. the output to the typewriter is of the form:-

FFLF	n	m
PROG3	n	m
PROG2	n	m
PROG1	5	7885
END		

where n is the first free location and m is the last free location. Let us assume that the output is as follows:-

```
FFLF        2900  7500
PROG3       2731  7635
PROG2       2400  7800
PROG1          5  7885
END
```

We would see from this that PROG1 occupies locations 5 to 2399 and 7801 to 7885 inc.
In the case of SAC programs the low addressed area is program space and the high addressed area is data space. FFLF gives the current value of the first free and last free pointers.

SET

Function

To set any store location to a new value.

Space occupied

PROGRAM	13	locations
DATA	1	location

Method of use

When this routine is entered after typing CP.SET. it requires to know the address of the location, say n, and the new value required, say m. The typewriter outputs the characters 'LOC' and n and m should then be typed in e.g.

```
LOC . n . m.
```

The routine will then attempt to read another integer i.e. n. If there are no further locations to be altered then a full stop will act as a terminator. Both n and m must be terminated by a full stop.

PRINT

Function

To print out any location of store as a pseudo-instruction to the typewriter.

Space occupied

PROGRAM	24	locations
DATA	4	locations

Method of use

This routine will print out the contents of any location as a pseudo-instruction to the typewriter. The only information that is required is the address of the location to be printed.

Operating.

After PRINT has been entered by typing CP.PRINT. the typewriter outputs the characters LOC and the address n must now be typed in e.g. LOC • n . When all the locations have been printed then the routine should be terminated by typing in a full stop.

4. ALINK

Function

To print out the contents of the two links in the ALGOL dynamic routines.

Method of use

This routine searches for the ALGOL dynamic routines and then prints the contents of the two links using the print routine in CP.

Operating

After entering the routine by typing CP.ALINK. the typewriter output is as follows:-

LINK pseudo-instruction
LINK pseudo-instruction
END

5. CANCEL

Function

To remove from the RAP list any named program and those compiled after it.

Space occupied

PROGRAM	11	locations
DATA	1	location

Method of use

This routine replaces the facility 'CANCEL' from RAP which has been removed from RAPMT. If wishing to remove a program from store it should be noted that all programs input afterwards will also be removed. (see diagram below). The routine searches for the program to be removed and when it has been found, resets the various RAP pointers to the new position.

RAP pointers which must be reset are:-

7920 - RAP pointer

7925 - FIRST FREE

7926 - LAST FREE

Operating

When the routine has been entered the program name is typed in terminated by a full stop.

e.g. CP. CANCEL. NAME.

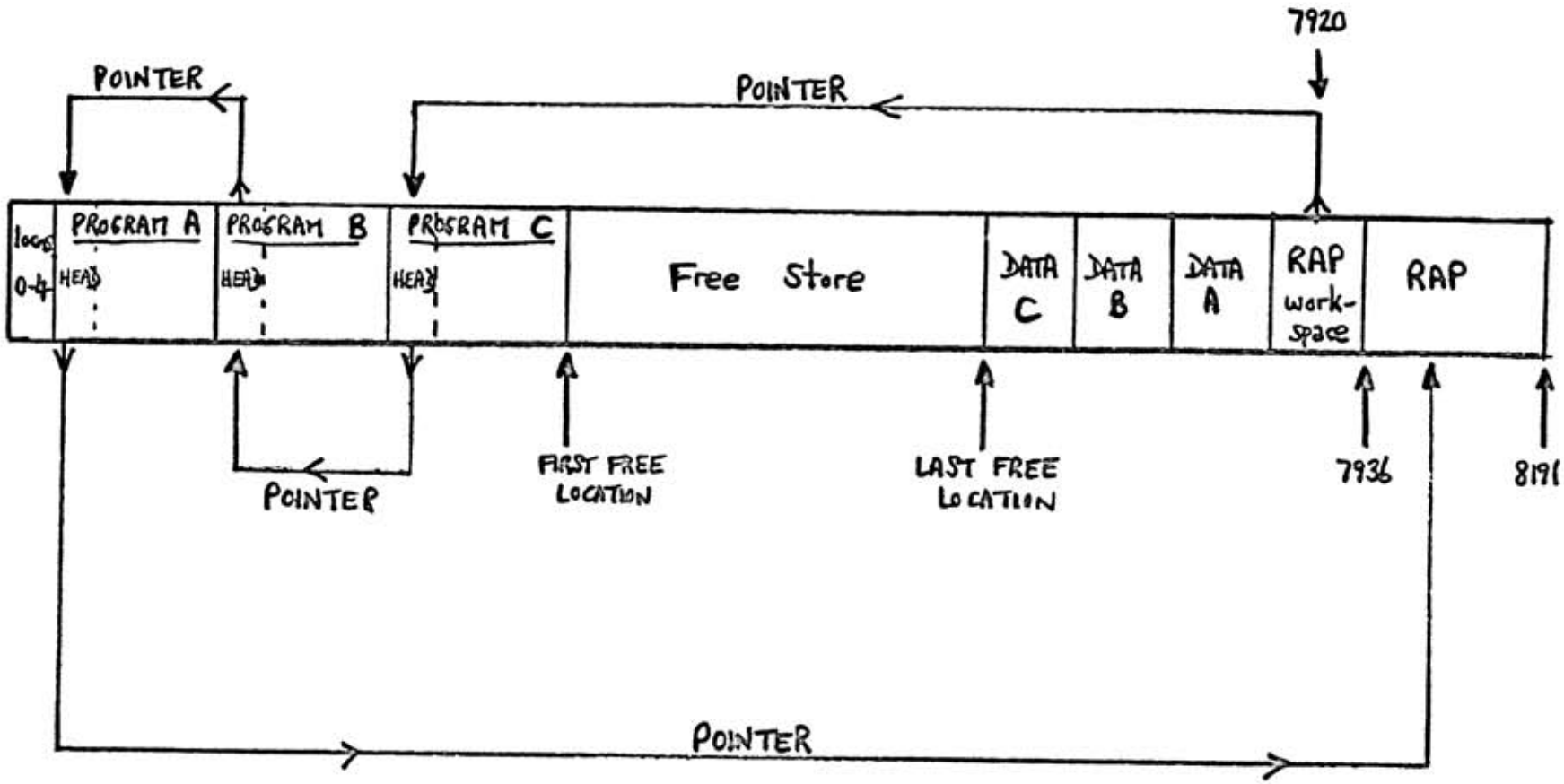


DIAGRAM OF 503 MAIN STORE

6. DUMP

Function

To write the contents of main store to any specified half unit of core-backing store.

Method of use

To dump 0-7935 locations from main store to any half unit of backing store, specified by the control typewriter. The backing store is numbered from 1 to n where n is the number of half units of CBS available.

Operating

Once the routine has been entered by typing CP.DUMP. n is typed in followed by a full stop. Where $1 \leq n \leq 8$ since it was designed for up to 4 units of CBS.
e.g. CP.DUMP.n.

7. GET

Function

To retrieve the contents of any half unit of core-backing store, as specified.

Method of use

This has the reverse effect of the routine DUMP and should be used to retrieve copies of main store written by the routine DUMP.

Operating

After entering the routine type in n. where n is the number of the half unit which holds the required copy of main store
e.g.

CP.GET.n.

TAPE

A tape coded in SAC is provided. The program is sum-checkable.

General Purpose Software Group.

Chapter 45 : LINE PRINTER LINE ASSEMBLER

CODE LINEAS S

FUNCTION

To facilitate the assembly of a line of characters to be printed on the line printer. The routine is designed for commercial use but may also be of value to the scientific user. It is also possible to output the line assembled to the paper tape punches or the control typewriter.

PCPP (2.5.3 Appendix 3) is used as a common program in order to handle the devices used. The user's program must be entered via PCPP in the manner described in Appendix 3, 2.5.3. of the Manual.

TAPES

A library tape is provided in S.A.C. form for input by SAP.

STORE USED 301 (including workspace)

METHOD OF USE

The operation to be performed is determined by the parameter word which accompanies the common program entry instruction.

The routine caters for fifteen commands, each of which allow certain options as shown below.

An internal count of the characters assembled for a line is kept and this count will hereafter be referred to as the word count. It defines the position of a character within the line to be printed.

Entry and Exit

An entry to LINEAS the accumulator has special significance according to the command specified. It will be referred to as "X" in the following definitions.

Entry to LINEAS must take the form:

```
COMP, LINEAS
aa  b :  cc  d
```

where aa specifies the command and b, cc and d further qualify the effect of each command as described below.

The calling program is always re-entered at the word following the parameter word.

List of Commands

The precise effect of each command is specified below. The effect of an error exit is described fully in ERROR INDICATIONS.

Command	Effect
aa = 01	Set the word count to X. X is held as an integer. Error exit occurs if X is outside the range $0 < x < 121$.
aa = 02	Set the 'paper controller' to X. Only the 7 least significant bits of X are recognised. The word count is not affected.
aa = 03	Assemble the character specified by the 7 least significant bits of X. Increase the word count by 1.
aa = 04	Assemble b spaces. Increase the word count by b.
aa = 05	Unpacking of alpha-numeric groups Assemble b characters starting from position cc of location d. The characters are packed 5 to a word using the £ facility of SAP. Under this

system the characters are packed in reverse order, the least significant 7 bits specifying character 1 (cc = 1).

If cc is set greater than 05 it will be reset to 01 and d will become d+1 so that the first character of word d+1 is next assembled.

The word count is increased by b.

aa = 06 Set the conversion register to X (see "Conversion Register settings"). The word count is unaffected. Error exit occurs if X is zero or negative.

aa = 07 Assemble b decimal digits of the value X, using the conversion register.

In order to supress the plus sign cc must be set to 00.

The remainder, R (if any), is held by the routine in the form R x 10^{b-1} (see notes on conversion register settings).

The word count is increased by b. If the sign is printed, this is counted as one of the b digits.

Error exit occurs if no conversion register has been set.

aa = 10

Assemble a further b decimal digits from the remainder held by LINEAS following the use of command 07.

Should it be required to output the single numbers 10 or 11 (as in a pence column) then the B-digit must be present in the parameter word. An error exit occurs if the number is greater than 11.

The plus sign may be suppressed by setting cc = 00. The word count is increased by b.

aa = 11

Assemble the value X with b digits before the decimal point and d digits following the decimal point, using the conversion register.

Non-significant zeros are suppressed and the remainder, R, is held by the routine in the form $R \times 10^{b + d - 1}$.

The word count is increased by $b + d + 1$.

Error exit occurs if no conversion register has been set.

aa = 12

Initialisation Procedure

X specifies the address of a location which contains the address of the first word of the 121 word buffer into which a line of characters may be assembled. The first word of the buffer will contain the "paper control" character.

This procedure clears the "paper controller" and puts a "space" character in each of the words of the line buffer. The word count is set to zero.

Error exit occurs if the address specified by X is zero.

aa = 13

Assemble X as a b digit integer. The routine manufactures its own conversion register setting (which destroys any previous setting). The plus sign may be suppressed by setting cc = 0. The word count is increased by b. No remainder is retained.

aa = 14

Assemble X as pounds, shillings and pence according to the following rules:

pounds : b digits

shillings: 2 digits

pence : 1 or 2 digits according to whether or not the B-digit of the parameter word is present (see aa = 10).

d spaces are inserted between each of the three amounts. To suppress the plus sign, set cc = 00.

aa = 15 Obtain from PCPP a standard size (121) buffer (or line) and prepare for LINEAS control as in operation 12.

Exit occurs with the buffer pointer in the accumulator, this is the value X to be taken in command 12.

The conversion register is cleared.

aa = 16 Output the line assembled using PCPP

If the parameter word B-digit is present the line specified by X, the buffer pointer, is output.

If the B-digit is not present, the line at present under the control of LINEAS will be output.

The setting of cc in the parameter word determines the output device and the manner in which output occurs. For a full description of the precise meaning of output functions the reader is referred to the PCPP description. The effects of the various settings of cc are as follows:

cc = 00 Output to lineprinter

cc = 01 Output to punch 1

cc = 02 Output to punch 2

cc = 03 Output to typewriter

cc = 10 "Output and wait" to lineprinter

cc = 11 "Output and wait" to punch 1

cc = 12 "Output and wait" to punch 2

cc = 13 "Output and wait" to typewriter.

When the user specifies his own buffer, "output and wait" orders are always given. This is in order to ensure that the user does not attempt to overwrite information in a line which has yet to be printed. "Output" orders enable PCPP to form a queue of lines to be printed and the operations of assembling lines and printing lines are thus time-shared.

When LINEAS outputs characters to the punches or typewriter, only the significant characters are printed. Trailing null characters are not output.

If the line of characters output was assembled into a buffer obtained from PCPP, then PCPP will automatically regain control of that buffer on completion of output and will re-allocate it on subsequent receipt of a request through command 15.

aa = 17 Return the buffer at present held by LINEAS to PCPP if the parameter word B-digit is not present.

If the B-digit is present the buffer specified by X is returned to PCPP.

Conversion Register settings

The conversion of a number to a string of decimal digits involves the setting of a "conversion register". This register determines the way in which the number is interpreted.

If the number to be assembled has n digits and is held as an integer in the accumulator, the operation is as follows:

The conversion register must be set to the value K x 10^{n-1} by means of operation 06.

On execution of operation 07 the following sequence occurs:

The conversion register constant is subtracted from the number to be assembled until the remainder is less than the constant, and the number of subtractions required gives the first decimal digit which is then assembled into the line. The remainder is then multiplied by ten and the process is repeated until the last digit has been assembled.

The constant set in the conversion register may then be altered and the remainder, if any, may be output under command 10.

If the conversion register has not been reset then the remainder is multiplied by ten before the first subtraction process, but if the conversion register has been reset, this initial multiplication by ten is omitted.

Leading zeros are always suppressed.

Example

The following section of program will cause the value held in the location 'INCHES' to be printed in terms of yards, feet and inches.

```
:
:
COMP, LINEAS                (obtain buffer from PCPP)
15  0:00  0
30   <+3600>
COMP, LINEAS                (Set conversion register for
06  0:00  0                 conversion to yards)
30    INCHES
COMP, LINEAS                (Assemble three digits ready
07  3:00  0                 for printing)
30   <+14>
COMP, LINEAS                (Assemble the character *)
03  0:00  0
30   <+1200>
COMP, LINEAS                (Set conversion register for
06  0:00  0                 conversion to feet)
COMP, LINEAS
10  1:00  0                 (Assemble one digit from remainder)
30   <+14>
COMP, LINEAS                (Assemble *)
03  0:00  0
30   <+1000>
COMP, LINEAS                (Set register for inches)
06  0:00  0
COMP, LINEAS                (Assemble two digits)
10  2:00  0
COMP, LINEAS                (Output the assembled line using PCPP)
16  0:00  0
:
:
```

If 'INCHES' contains the value 5830 the output resulting from the above operation will be

161*2*10

ERROR INDICATIONS

When an error occurs due to one of the causes listed below, the message

ERROR LINEAS COMMAND "aa"

is displayed, where aa specifies the offending command. Control returns to the calling program with the accumulator set to contain -1.

The possible causes are as follows:

1. The command specification is out of range.

2. One of parameters b, c or d is zero or out of range.

3. During number output, the digit in question is greater than 9 (see commands 10 and 14).

4. The word count has exceeded 120.

M. PUTT.
INTERNAL COMPUTING
BUREAU

JUNE, 1966.

CHAPTER 46 : DUMP, MARK 2

CODE DUMP2 S

FUNCTION

To facilitate the creation of program batches on magnetic tape of the same type as those created using the BATCH S (2.2.3.25) system but with an improvement in operating efficiency and an increased range of facilities.

It is intended that this system should replace the BATCH S system but it has been issued as a separate library program so that the user may make a choice. Existing batches created by BATCH S may not be input under this system and arrangements must be made to copy batches to another tape by inputting them using the BRING & LOAD retrieval tapes and using DUMP2 to create new versions which may subsequently be input under the new system.

DUMP2 works in conjunction with a special version of RAP known as RAP MT - see 2.2.1. Appendix 1 for a description of its features. By making use of RAP MT, no leader tape, such as BRING & LOAD in the BATCH S system, is required (see METHOD OF USE, Batch Retrieval).

TAPES

The tape is provided in a mnemonic form suitable for input under SAP Mark 1. It may be used to produce a binary version using SAP in the normal way (see SPECIAL OPTIONS for alternative actions).

STORE USED

Program : 414 locations

Workspace: 548 locations, including 512 words for the transfer buffer (see SPECIAL OPTIONS for alternative size buffer).

N.B. Location 4 is not preserved at any stage of the dumping procedure.

BATCH TYPES

The type of batch created is determined by the operator's choice of one of the eight entry points (see METHOD OF USE). The various types of batch which may thus be obtained are described below. The operator may also obtain extra information by use of the B-digit of the word generator (see METHOD OF USE).

The batch types, and their possible applications, are as follows:

Partial Dump — This is the type of batch created by BATCH S. It comprises only the area of main store apart from RAP and the free store area defined by the free store pointers. According to the entry point chosen part or all of the core backing store is also dumped.

In order to achieve this, DUMP2 is input after the required programs have been assembled and it does not form part of the batch.

Total Dump — All of the main store from locations 5 to 7935 is dumped. DUMP2 itself is included in the batch. The user may also choose to dump all or part of core backing store as well.

The total dump enables the user to manually interrupt during a run, dump the store configuration as a batch and type CONT. to continue. The run may be either a program run or a stage in the compilation of a program, but it has special value in enabling the operator to split a long run into short stages and thus to provide "restart" points in case of machine failure.

METHOD OF USE

For a Partial Dump, DUMP2 must be the last program input and it will automatically remove all trace of its presence before writing the main store to magnetic tape.

For a Total Dump, DUMP2 may be input at any stage.* DUMP2 itself will be written away to magnetic tape together with the rest of the main store (locations 5 to 7935 inclusive).

* But must not be placed below location 100.

All batches are dumped on Handler 1.

Reel Preparation

Before a scratch magnetic tape may be used for the first time to contain program batches it must be prepared as follows:

1. Load the reel on handler 1 at B.O.T.

2. Type RESET. to clear the main store.

3. Input DUMP2 (type IN.) .

4. Type DUMP2;5. This has the effect of writing a special block at the beginning of the reel. For this operation to be performed correctly, DUMP2 must be stored from location 5 onwards.

The reel is now ready for the addition of program batches. Provided the rules are followed (e.g. see entry 8 below), the reel preparation sequence need not be repeated.

Batch Names

The user must specify at the time of the dumping procedure a batch name. This is typed in as part of the message(see "BNAME" below). According to the name chosen the reel will be either a single batch reel or a multi-batch reel. A multi-batch reel is obtained only by naming the first batch "BATCH1". Unless this is done, the first batch on a reel is over-written by the next batch dumped.

The names "LAST" or "CBS" must not be used as batch names.

If two batches are assigned the same name it will not be possible to access the second batch dumped.

Batch Creation

The type of message required for each type of batch is shown below.

ENTRY	MESSAGE	EFFECT
1	DUMP2. "BNAME".	Partial Dump plus complete dump of core backing store.
2	DUMP2;2."BNAME".	Partial Dump of main store only.
3	DUMP2;3."BNAME".	Total Dump plus complete dump of core backing store.
4	DUMP2;4."BNAME".	Total Dump of main store only.
5	DUMP2;5.	Prepare scratch reel for dumping (see "Reel Preparation").
6	DUMP2;6.N."BNAME".	Partial Dump plus part of core backing store where N specifies the number of units of 1024 locations to be dumped. Thus locations 0 to 1024N-1 of backing store are dumped.
7	DUMP2;7.N."BNAME".	Total Dump plus 1024N locations of backing store (see entry 6).
8	DUMP2;8."BNAME".	Cancel all batches on reel from "BNAME" onwards. By typing DUMP2;8.BATCH1 the user can re-prepare a multi-batch reel.

Batch Retrieval

To input a batch named X type

IN;X.

RAPMT recognises this message and initiates the retrieval of the batch.

If the batch cannot be found, or if it cannot be read due to a machine fault, an error message is displayed - see ERROR INDICATIONS.

Listing of Batch Names

When a batch is being created or retrieved the operator may obtain a list of the batches on the reel by depressing the B-digit of the word generator.

Each batch name is displayed on a new line on the output writer. On input only the names of those batches situated before the batch being retrieved are displayed.

A blank line indicates the presence of a poor area of tape on the reel (see ERROR INDICATIONS).

ERROR INDICATIONS

The following error messages may be displayed during the creation or retrieval of the batch.

Message	Interpretation
H1MNL	Handler 1 is set to "LOCAL". Set to "AUTO" and continue by typing CONT..
XREEL	The batch requested has not been loaded. This is only displayed when a single batch reel has been loaded.
NO BATCH FND	The batch requested has not been found on the multi-batch reel loaded.
NO WPT	The reel is protected when an attempt is made to dump a batch. Remove the protection and continue by typing CONT..
SUMERR	A batch has been brought down incorrectly. Each batch has a checksum which is checked on oompletion of input. A subsequent retrieval attempt should be made.
TP	Displayed each time a parity error occurs when writing the batch. This gives an indication of the reliability of the reel and/or handler.
RP, P	Displayed each time a parity error occurs when retrieving a batch.
CANT WRITE	Unable to write a block after five attempts. This area will be skipped over completely if the dumping process is repeated. No continuation is immediately possible.
LNOISE	Unable to read a block after sixteen attempts. No continuation possible.

SPECIAL OPTIONS

The speed at which a batch may be retrieved depends entirely on the size of the transfer buffer which governs the size of the magnetic tape blocks into which the batch is split. Large blocks enable high speed retrieval but require more main store workspace which interferes with large systems such as ALGOL. Because the workspace is necessarily claimed when DUMP2 is input it is not possible to give the user a dynamic choice of buffer size. In order to give the user a choice it has been made a simple matter to modify the program itself. The tape is supplied in mnemonic form and the instructions given below enable the user to quote a larger or smaller buffer size according to his requirements. The size specified must be a power of 2.

Batches which have different block sizes because they have been dumped by different versions of DUMP2 created in this way may still be contained on the same reel of magnetic tape.

The user, in order to change the transfer buffer size must supply a tape containing two new SAP declarations as shown below:

The mnemonic tape supplied has the following form:

```
program DUMP2;
    .
    .
    .
    &
data  A(511);
replace BUF1 [;<00 512 : 00 A>],
        BUF2 [;<00 512 :>] ,
        BUF3 [; <+512>],
        BUF4 [;< +506 >],
        BUF5 [;< +2>],
        BUF6 [; <+16>];
    &
    .
    .
    .
```

All the constants used are calculated from 512 according (where not obvious) to the following formulae:

data	-	A(X-1)
BUF4	-	X-6
BUF5	-	1024/X
BUF6	-	16384/X

where X is the buffer size. Thus the new BUF5 constant for a transfer buffer of 64 words would be 16.

Each installation is recommended to decide on the maximum and minimum buffer sizes appropriate to their requirements and, where two versions are in common use, to add a distinguishing letter or digit to the program name, e.g. DUMP2F (Fast) and DUMP2 S (Slow but small).

R.E. NELSON
C.F. DEAL

May, 1966.

APPENDIX I

DUMP2 requires free store of the order of 600 locations in order to be input, thus situations will arise in which insufficient space is available. To enable such batches to be written to magnetic tape the following version of DUMP2 has been produced.

DUMP2M

CODE DUMP2M S

FUNCTION

DUMP2M is designed to produce main store batches on magnetic tape previously prepared by DUMP2. The batches may be mixed on the same reel with DUMP2 batches and are retrieved using RAP MT. The program may be used to cancel any batch on a reel but it cannot be used to produce core backing store batches or to prepare an unused reel (see DUMP2 description).

TAPE

The program is written in SAC and the tape is provided in sum checked binary form.

STORE USED

Minimum free store required for input:		267 locations
Program	:	127 locations
Workspace	:	11 locations

N.B. The extra free store on input is required for the input routine which is situated at the front of all SAC - binary tapes.

METHOD OF USE

DUMP2M may be used in any position in store above location 92; if it is the last program input DUMP2M will automatically delete itself from the RAP list before writing the store to magnetic tape. Locations 5 to 7935 inclusive are dumped as a single block onto handler 1.

DUMP2M may also be used to cancel any batch on handler 1.

ENTRY	MESSAGE	EFFECT
1	DUMP2M. "BNAME".	Cancel all batches on handler 1 from "BNAME" onwards
2	DUMP2M;2. "BNAME".	Write the main store as a batch called "BNAME" to handler 1.

STAR

STAR will automatically be entered when a batch has been retrieved, provided it is the first program in store.

Batch Names
Batch Retrieval

See DUMP2 description.

Listing of Batch Names

When a batch is retrieved, a list of all batches situated before it will be obtained on the output writer if the B-digit key is depressed. No listing is possible on creation of a batch.

ERROR INDICATIONS

Message	Interpretation
Derror	DUMP2M is placed below location 92 in store when attempting to dump.
Xavail	is displayed if handler 1 is in manual or there is no write-permit ring fitted to the reel. The program will continue automatically on setting the handler to remote and/or fitting the write-permit ring.
R	displayed each time a parity error occurs on reading a block.
P	displayed each time a parity error occurs on writing a block.

Message	Interpretation
XWRITE	unable to write a block after 6 attempts. Repeat the dumping process.
SUMerr	a batch has been brought down incorrectly. Each batch has a checksum which is checked on completion of input. Repeat the retrieval attempt.

R. Gordon

October, 1966.

Chapter 47: BACKGROUND PROGRAM DEVICE ROUTINES

CODE BGPROG S

FUNCTION

To facilitate the development of "background" programs which perform pseudo off-line functions by making use of peripherals not currently being used by the main program. The object is achieved by providing a set of common programs each of which controls a particular device.

The programs provided control the following devices:

Tape Readers

Tape Punches

Digital Plotter

Card Reader

Lineprinter

Magnetic Tapes

For the user wishing to time-share the operation of a non-standard device connected through an I.M.U. the process used is described below so that extra device programs may be produced by him.

STORE USED

Device Program	Code Name	Locations (including workspace)
Tape Readers	READ	33
Tape Punches	PUNCH	40
Digital Plotter	PLOTTER	36
Card Reader	CARDR	51
Lineprinter	LINEPR	54
Magnetic Tapes	MTAPE	83

TAPES

A composite tape is provided which contains each program in mnemonic form for input under SAP Mark 1. The tape is labelled "BGPROG S(COMBINED TAPE)". The user is recommended to produce binary versions of the programs he requires by translating using SAP Mark 1 with key 36 of the number generator depressed.

METHOD OF USE

1. Writing the Background Program

Each device program provided ensures that the group 7 orders which are issued are only issued if the peripheral device controlled by that program is not busy. The background program itself must not itself issue group 7 orders directly.

Entry to the device programs is by means of a common program entry instruction, e.g. COMP,READ for the tape reader. Each program has two entry points. The normal entry, that used for transfer of information, is the first entry point but before any information is transferred a setting-up entry (entry point 2) must be performed which gives advance warning of intention to use the devices in question. The setting-up entry to READ must always be made, whether or not READ is to be used again. This is because the location of the tape reader interrupt registers is used to form a communication link between the device routines.

In the case of devices for which error states may be detected using control words there is a reporting technique which enables the writer of the background program to display an appropriate message to cause an error state to be rectified. This technique enables the user to decide on the extent to which messages should be used and enables the size of the device program to be kept to a minimum.

The background program is entered initially following a manual interruption of the main program. The technique used for continuation from this point depends on the fact that a manual interrupt has taken place prior to initial entry. Since the main and background programs are unlikely to finish together a special arrangement must be made on the part of the background

program in order to allow the main program to continue once the job of the background program is completed. For this reason, the normal 'STOP' instruction must not be used. Each background program must end with the instruction

```
66  7888
```

A 'signing-off' message such as "BGEND" could be issued in the following way:

```
30  <9BG END>
RAPlprint
66  7888
```

Warning: These background device programs may not be run in conjunction with a main program which is attempting to use peripherals under interrupt control.

2. Device Program Entry Parameters

The method of entry to transfer information through each device program is as follows:

READ

Entry: Accumulator contains the reader number.

Exit: Accumulator contains the character read in bits 1 to 7 inclusive.

Example: To input a series of characters from reader 2 and form their sum:

```
        :
        :
    COMP,READ,2
    26   SUM
next)  30   <+2>
    COMP,READ
    24   SUM
    40   next
        :
        :
```

PUNCH

Entry: Accumulator contains the punch number.

The word following the entry instruction is a parameter which gives the address of the location holding the character to be punched.

Exit: No information on exit.

Control returns to the location following the parameter word.

Example: To output the character with value 1 on punch 1:

```
      :
      :
COMP,PUNCH,2
30    <+1>
20    character
COMP,PUNCH
+     character
      :
      :
```

PLOTTER

Entry: The accumulator contains a value which determines the pen movement. This value is formed by subtracting the instruction 72 7168 from the appropriate instruction as defined on page 2 of section 1.4.7, of the 503 Manual.

Exit: No information is supplied on exit.

Example: To carry out one West movement followed by one South East movement:

```
        .
        :
COMP,PLOTTER,2
30   <+2>
COMP,PLOTTER
30   <+9>
COMP,PLOTTER
        :
        .
```

LINEPR

Entry: The word following the entry instruction is a parameter which gives the address of the first word of the buffer to be output to the lineprinter. The character in the first word determines the vertical throw before the line is printed.

Exit: The accumulator is zero if the lineprinter was available.

Example: To output a line of 120 characters contained in an area from 'L' onwards to the lineprinter:

```
                :
                :
            COMP,LINEPR,2
AGAIN)      COMP,LINEPR
            +L
            42   CONTINUE
            30   <9LP MAN>
            RAPlprint
            40   AGAIN
CONTINUE)       :
                :
                .
```

CARDR

Entry: The word following the entry instruction is a parameter which gives the address of the first word of the 80 word buffer into which a card is to be read.

Exit: The accumulator is zero if the cardreader was available.

Example: To read a card into an area from location 'L' onwards.

```
                :
                :
          COMP,CARDR,2
AGAIN)    COMP,CARDR
          +L
          42   CONTINUE
          30   <9CARD M>
          RAPlprint
          40   AGAIN
CONTINUE)       .
                :
```

MTAPE

Entry: The accumulator contains the handler number.

The first parameter word (following the entry instruction) contains an integer which defines the function to be performed:

Value	Function
1	Read
2	Write
3	Advance
4	Retreat
5	Erase
6	Rewind
7	Check Control Word, clearing error state.

In the case of read and write instructions a second parameter word must be supplied as follows:

00 <length> :00 <position>

where 'length' defines the number of words in the block and 'position' defines the position in main store of the first word of the block to, or from, which the information is to be transferred.

Exit: Parameter 1<7

Accumulator Value	Interpretation
Zero	Handler in manual, or writing not permitted when attempting to write.
40 0:00 X	Normal exit; X takes the value of the handler control word.

Parameter 1=7

The Accumulator contains the handler control word. The sign bit (39) is also present if a parity error has occurred.

Example: To write a block onto magnetic tape at the B.O.T. where the block starts from location L and has length 500. The handler used is number 5.

```
        :
        .
    COMP,MTAPE,2
AGAIN)  30   <+5>
    COMP,MTAPE      rewind to B.O.T.
    +6
    42   ON
    30   <9H5 MAN>  Test whether
    RAPlprint       handler available
    40   AGAIN
ON)     30   <+5>
    COMP,MTAPE
    +2
    00 500:00 L
        .
        :
```

SPECIAL NOTES

1. The main program may use any peripheral it wishes to without disturbing the background program.

2. If the background program finishes before the main program, the main program will continue and vice versa. A second background program may then be entered by interrupting the

main program again and entering it. However, if a second main program is entered, the background program will not continue automatically, but may be re-entered in the normal way.

3. The background program must be kept in a fixed position in main store, that is it must not be translated by SAP2 where, in the event of a 'collapse', the background program could be moved from main store.

4. It should be noted that when using the MTAPE routine the easiest way of forming the second parameter word is by labelling it and placing the required word in position, in which case there must be no checksum while assembling this program i.e. key 35 should be depressed.

5. The first routine on the composite,mnemonic tape provided is the routine READ, which has detailed comments added. These will help the programmer to understand the basic principles on which this and all of the other device routines depend.

The lineprinter routine (LINEPR) also contains detailed comments. The additional contingency to be catered for here is that location 8176, which is used by the hardware for peripheral transfers (see the description of the 76 instruction, 1.3.1), must be preserved on interrupt and restored before control

is returned to the main program. This must be done for all device routines which issue 76 instructions.

R.E. NELSON,
AUGUST, 1966.

SOFTWARE SUPPORT GROUP
SCIENTIFIC COMPUTING DIVISION

Appendix 1

Magnetic Tape Input and Output Routines

FUNCTION

To provide separate routines for input and output of data by a background program using magnetic tape so that the background program which uses either routine will require a minimum of main store space. The routines include comprehensive error retrieval techniques. Each routine makes use of the basic routines READ and MTAPE.

Each block, before being written to magnetic tape, is assigned a block number which occupies bits 21-39 of the first word of the block of data. On input a block may thus be requested by number. The first block on a reel is numbered 1.

The input routine is MTREAD. The output routine is MTWRITE.

STORE USED

MTREAD	88 Locations (including workspace)
MTWRITE	81 Locations (including workspace)

TAPES

The tape provided is in mnemonic form for input under SAP Mark 1. Both MTREAD and MTWRITE are held on the same tape. The tape is labelled MTWRITE AND MTREAD (BG).

The user is recommended to produce binary versions of the programs by translating using SAP Mark 1 with keys 35 and 36 of the number generator depressed.

METHOD OF USE

Each program is entered as a common program in the same way as the basic device routines. Both of the device routines READ and MTAPE must be stored before MTWRITE or MTREAD are input.

The users background program need only be concerned with entry to MTWRITE or MTREAD; any setting-up entries normally necessary when READ or MTAPE are employed are performed automatically.

Throughout the running of the background program the user may only read from, or write onto, one handler. He may not change the handler number required once the background program has been triggered. That is to say, for example, although he may read from handler 1 and write on handler 2,

he may not read alternately from handlers 1 and 3 and write on handlers 2 and 4. The handler to be used is specified through the initial setting-up entry (see below).

The block number assigned to each block being written is placed in bits 21-39 inclusive of the first word of the block. The user should therefore arrange to leave this part of the first word free.

The method of entry to each routine is shown below:

MTREAD

Setting-up Entry: COMP,MTREAD,2

Entry: The accumulator contains the number of the handler as an integer.

Normal Entry: COMP,MTREAD

Entry: The accumulator contains the block number required as an integer.
A parameter word must be supplied which contains
00 <length> :00 <position>
where "length" is the length of the specified block to be input, and "position" is the address of the first word in m.s. to which the block is to be read.

Example:

```
30 <+3>
COMP,MTREAD,2
   :
   :
30 <+5>
COMP,MTREAD
00 200:00 address
   :
   :
```

indicates that handler 3 is to be used and that block number 5 should be read into an area beginning at "address" and should be of length 200 locations.

MTWRITE

Setting-up Entry: COMP,MTWRITE,2

Entry: The accumulator contains the number of the handler as an integer.

Normal Entry: COMP,MTWRITE

Entry: The accumulator contains no relevant information and is not preserved on exit. A parameter word must be supplied which contains

00 <length> :00 <position>

where "length" is the length of the block for output, and "position" is the address of the first word in main store from which the block is to be written.

Example:

```
30 <+4>
COMP,MTWRITE,2
    :
    :
COMP,MTWRITE
00 100:00 address
    :
    :
```

indicates that handler 4 is to be used and that a block should be written on to tape length 100 locations starting from the location specified by "address"

MAGNETIC TAPE FORMAT

All blocks are written in binary form (Format 2) using odd parity. The same format is assumed for input.

ERROR INDICATIONS

MESSAGE	REASON	ACTION
H MAN	requested handler in manual	message repeated until handler released from manual
ERR LS	long or short block detected	Irretrievable error Main program continues
ERR P	parity error detected	Irretrievable error Main program continues

R.E.NELSON,
AUGUST,1966.

SOFTWARE SUPPORT GROUP
SCIENTIFIC COMPUTING DIVISION

Appendix 2

EDIT8 (Background Program Version)

FUNCTION

To perform the same function as the standard version of EDIT8 (2.2.3.27) but to perform the function as a background program by making use of the necessary background program device routines READ and PUNCH.

CODE

The program tape is labelled:

EDIT8 (BG)

In order that it may be used to replace the standard version of EDIT8 when used as a common program, this version is still called EDIT8. Since it is impossible to use it independently of the basic device routines there is little chance of confusion between this and the standard version.

METHOD OF USE

Initiate the main program to be run. It is assumed that EDIT8, READ and PUNCH are already in main store.

Interrupt the main program by depressing the MESSAGE button.

Use EDIT8 according to the normal operating instructions (2.2.3.27). The main program will continue automatically in parallel with EDIT8.

ERROR INDICATIONS

The messages below are those which may occur in addition to those detailed in 2.2.3.27. They concern only the fact that one or both of the routines READ and PUNCH may not have been input before EDIT8. On attempting to input EDIT8 the appropriate message

READ		PUNCH
	or	
NOPROG		NOPROG

will be displayed if either or both routines are not present.

TAPE

The tape is supplied in mnemonic form for input under SAP Mark 1.

A binary tape should be produced from this by translating using SAP with keys 35 and 36 of the number generator depressed.

A.W.PORRITT,
AUGUST, 1966

SOFTWARE SUPPORT GROUP
SCIENTIFIC COMPUTING DIVISION

Appendix 3

EDIT8 INTERFACE (Background Program Version)

FUNCTION

To perform the same function as the standard version of EDIT8I (2.2.3.37) but to perform the function as a background program by making use of the necessary background program device routines READ and PUNCH.

CODE

The program tape is labelled:

EDIT8I (BG)

In order that it may be used to replace the standard version of EDIT8I when used as a common program, this version is still called EDIT8I. Since it is impossible to use it independently of the basic device routines there is little chance of confusion between this and the standard version.

METHOD OF USE

Initiate the main program to be run. It is assumed that EDIT8I, READ and PUNCH are already in main store.

Interrupt the main program by depressing the MESSAGE button.

Use EDIT8I according to the normal operating instructions (2.2.3.37). The main program will continue automatically in parallel with EDIT8I.

ERROR INDICATIONS

The messages below are those which may occur in addition to those detailed in 2.2.3.37. They concern only the fact that one or both of the routines READ and PUNCH may not have been input before EDIT8I. On attempting to input EDIT8I the appropriate message

READ		PUNCH
	or	
NOPROG		NOPROG

will be displayed if either or both routines are not present.

TAPE

The tape is supplied in mnemonic form for input under SAP Mark 1.

A binary tape should be produced from this by translating using SAP with keys 35 and 36 of the number parameter depressed.

A.W.PORRITT
AUGUST, 1966.

SOFTWARE SUPPORT GROUP
SCIENTIFIC COMPUTING DIVISION

Appendix 4

LPRINT (Background Program Version)

FUNCTION

To perform the same function as the standard version of LPRINT (2.2.3.22) but to perform the function as a background program by making use of the necessary background program device routines READ and PUNCH.

CODE

The program tape is labelled:

LPRINT (BG)

In order that it may be used to replace the standard version of LPRINT when used as a common program, this version is still called LPRINT. Since it is impossible to use it independently of the basic device routines there is little chance of confusion between this and the standard version.

METHOD OF USE

Initiate the main program to be run. It is assumed that LPRINT, READ and PUNCH are already in main store.

Interrupt the main program by depressing the MESSAGE button.

Use LPRINT according to the normal operating instructions (2.2.3.22). The main program will continue automatically in parallel with LPRINT.

ERROR INDICATIONS

The messages below are those which may occur in addition to those detailed in 2.2.3.22. They concern only the fact that one or both of the routines READ and PUNCH may not have been input before LPRINT. On attempting to input LPRINT the appropriate message

READ		PUNCH
	or	
NOPROG		NOPROG

will be displayed if either or both routines are not present.

TAPE

The tape is supplied in mnemonic form for input under SAP Mark 1.

A binary tape should be produced from this by translating using SAP with keys 35 and 36 of the number generator depressed.

A.W.PORRITT
AUGUST, 1966.

SOFTWARE SUPPORT GROUP
SCIENTIFIC COMPUTING DIVISION

Chapter 48: GENERAL MAGNETIC TAPE ROUTINES

CODE GMT S

FUNCTION

The routines enable blocks to be written to and read from any specified magnetic tape handler.

FORMAT

The first word of each block written contains the block number. This is written in by the routine and is put in the upper half of the first word.

Blocks are read and written using odd parity, format 2.

Writing and reading may be alternated on any one or more handlers.

STORE USED

216 locations, including workspace.

DESCRIPTION OF ROUTINE

The routine divides into three different sections. These are (i) A setting-up procedure

(ii) Writing onto tape

(iii) Reading from tape.

(i) The setting-up section must be entered before any other entry, to set markers and check the state of the handler.

(ii) The writing section puts given blocks to tape, first checking the state of the handler. It checks that the block is of a suitable length, i.e. ≥ 4 locations and then writes onto tape a block following the last one written. The parity bit is checked after writing and, if set, the position of the block just written is found by, retreating to the block before the one now being written, and reading (unless at the beginning of tape). Part of the tape is then erased and writing takes place again. After ten attempts to write an error message is output An error message is also output after the end of tape marker is reached.

(iii) This section reads a required block from tape. It checks that the block number required is within the range of blocks already written onto tape, and also the state of the handler is checked. If no blocks have been written to tape, then no checks are made on the block number range. The block number is compared with the position of the tape and the appropriate position aimed for by retreating or advancing. A block is then read, and its number checked with the one required. If this is not correct, the appropriate retreats are given or the next block read.

When the correct block number is found, if the parity bit is set on reading, retreats are made and the block read again. After ten attempts an error message is output.

If the parity bit is not set on reading, a short or long block is checked for. If there is a noise block an error message is output and the next block read. If there is an ordinary short block, ten attempts are made at

reading and then an error message output. On detecting a long block an error message is output.

METHOD OF USE

There are three different entries. These are available by

COMP, GMT - setup
COMP, GMT, 2 - write
COMP, GMT, 3 - read.

Entry 1, used for setting-up, needs to be given the handler number in the accumulator. If more than one handler is to be used, this entry must be made for each handler.

Entry 2, used for writing must be given the handler number in the accumulator and there must be one parameter word containing the following information:-

00 length of block : 00 m.s. address for transfer.

The programmer must remember his own block number.

Entry 3, used for reading must be given the handler number in the accumulator and there must be two parameter words containing the following information:-

00 length of block : 00 m.s. address for transfer
+ block number.

TAPES

The tape is coded in SAP1 code; it is recommended that a binary tape is produced with keys 35 and 36 of the word generator depressed.

ERROR MESSAGES

Message	Meaning and Effect
MAN H 'N'	The required handler 'N' is in manual or has no write permit ring. The program continues when this is corrected.
H EOT	End of tape is reached. Program stops.
BL TS	The block length given for writing is too short, i.e. <4 locations. Program stops.
CANNOT WRITE	Handler unable to write. Program stops.
CANNOT READ	Handler unable to read. Program stops.
NOISE	Noise block encountered. Program continues.
LTH ER	Long or short block found on reading without parity. Program stops.
S	Displayed when a short block which is not a noise block is encountered on reading. Program continues.
BLNOTL	Block number too large, i.e. out of range, Program stops.

Appendix 1: MAGNETIC TAPE READ ROUTINES

CODE GMTR S

FUNCTION

These routines enable blocks to be read from any specified handlers.

FORMAT

It is assumed that each block read has a block number, held in the upper half of the first word of each block.

Blocks are read using odd parity, format 2.

There is a parallel routine 'GMTW' which writes with the appropriate block numbering form.

STORE USED

122 locations including workspace.

DESCRIPTION OF ROUTINE

The routine divides into two different sections. These are:-

(i) A setting-up procedure

(ii) Reading from tape.

The setting-up procedure must be entered before reading. It sets various markers and checks the state of the handler.

When reading there is always, a check on the condition of the handler.

The block number is compared with the position of the tape and the appropriate position aimed for by retreating or advancing, i.e. if no blocks have previously been read an advancing action will be taken.

A block is then read and its number compared with the one wanted. If it is not the required block, the appropriate retreats are given or the next block read.

When the correct block is found, if the parity bit is set on reading, retreats are made and the block read again.

After ten attempts an error message is displayed.

If the parity bit is not set on reading, a check is made for a short or long block.

If there is a short noise block an error message is output and the next block is read.

If there is an ordinary short block, ten attempts are made at reading and then an error message is displayed. On detecting a long block an error message is displayed.

METHOD OF USE

There are two different entries, which are available by

COMP, GMTR - setup
COMP, GMTR, 2 - read.

Entry 1, used for setting up needs to be given the handler number in the accumulator. If more than one handler is being used, this entry must be made for each handler.

Entry 2, used for reading, must be given the handler number in the accumulator and there must be two parameters words containing the

following information.

(i) 00 length of block : 00 m.s. address for transfer

(ii) + block number required.

TAPES

The tape is coded in SAP1 code and it is recommended that a binary tape is made, entering SAP with keys 35 and 36 of the word generator depressed.

ERROR MESSAGES

Message	Meaning and Effect
MAN H 'N'	The program waits until handler 'N' is released from manual and then continues.
CANNOT READ	Handler unable to read. Program stops.
NOISE	Noise block encountered. Program continues.
LTH ER	Long or short block encountered, without parity. Program stops.
S	Displayed when a short block which is not a noise block is encountered on reading.

Appendix22: MAGNETIC TAPE WRITE ROUTINES

CODE GMTW S

FUNCTION

These routines enable blocks to be written to any specified handlers.

FORMAT

The first word of each block written contains the block number. This is written in by the routine and is put in the upper half of the first word.

Blocks are written using odd parity, format 2.

There is a parallel routine 'GMTR' which reads and assumes that there is block numbering as written by GMTW.

STORE USED

119 locations including workspace.

DESCRIPTION OF ROUTINE

The routine divides into two different sections. These are (i) A setting-up procedure
(ii) Writing onto tape.

The setting-up procedure must be entered before writing. It sets various markers and checks the state of the handler.

When writing there is always a check on the condition of the handler. There is a check that the block is of a suitable length, i.e. ≥ 4 locations, and then the block is written onto tape at the current position.

The parity bit is checked after writing and, if set, the position of the block just written is found by retreating to the block before the one now being written and reading (unless at the beginning of tape). Part of the tape is then erased and writing takes place again. After ten attempts to write an error message is displayed. An error message is also output after the end of tape is reached.

METHOD OF USE

There are two different entries. These are available by

COMP, GMTW - setup
COMP, GMTW,2 - write.

Entry 1, used for setting up, needs to be given the handler number in the accumulator. If more than one handler is being used for writing this entry must be entered for each handler.

Entry 2, used for writing must be given the handler number in the accumulator and there must be one parameter word containing the following information:-

00 length of block : 00 m.s. address for transfer.

The programmer must remember his own block number.

TAPES

The tape is coded in SAP1 code, and it is recommended that a binary tape is made, entering SAP with keys 35 and 36 of the word generator depressed.

ERROR MESSAGES

Message	Meaning and Effect
MAN H 'N'	The required handler 'N' is in manual or has no write permit ring. The program continues when this is corrected.
H EOT	End of tape is reached. Program stops.
BL TS	The block length given is too short, i.e. <4 locations. Program stops.
CANNOT WRITE	Handler unable to write. Program stops.

THE 503 BASIC TEST TAPE

This chapter describes the test programs. It is recommended that they are run at least once each day. They comprise programs designed to test for malfunction of the basic 503 system. Although the programs test each unit of the machine separately, they should not be used for precise diagnosis of any fault. Failure of any test should be reported to the maintenance engineer.

The 503 basic test tape contains 4 separate tests, they are:—

TST01S—Paper tape input/output, control station interrupt, and reserved area protection test.

TST02S—Store test.

TST03S—Function test.

TST04S—Typewriter and Word Generator test.

All the tests are on one tape, punched in a special binary code which is read in via the initial instructions and includes a routine designed to cope with any error interrupt which may occur whilst reading in the tape (see Input Routine).

A message routine (see below) is always available, control is transferred to this routine after some errors, and control can be obtained at any time by pressing the MESSAGE button. The error interrupt routine (see below) is common to all programs.

N.B.—The basic test tape does not use R.A.P.

MESSAGE ROUTINE

This is a common program and is always available. Control is obtained either by pressing the MESSAGE button or by program control. A 'message' is a sequence of characters terminated by . (full stop) typed by the operator on the input keyboard. In the cases where control is transferred to this routine after an error a message CONTERR. will mean continue from the point where the error occurred. If control was obtained by pressing the MESSAGE button, then the message CONT. will mean continue from the point when the button was pressed. The other form of message accepted by the routine is a decimal integer x (unsigned) where $x \leqslant 8191$, terminated by a full stop. This means transfer control to location x.

A message accepted by the routine results in the character $>$ being output on exit from the routine. A message (or character) not accepted results in $*$ being output and the routine awaits another message.

CHARACTERS ACCEPTED AND NON-ACCEPTED

Accepted characters	L T S (ignored) digits 0-9 upper case letters A-Z upper case full stop
Non-accepted characters	10 11 = ; | - + and all lower case characters

MESSAGES NOT ACCEPTED

(a) a decimal integer > 8191

(b) any combination of upper case letters, terminated by a full stop which is not CONT or CONTERR.

(c) any combination of digits and letters.

Thus it can be seen that comments can be typed, using upper case letters and terminating the comment with a full stop (i.e. unacceptable message).

ERROR INTERRUPT ROUTINE

This routine is common to all programs; when an error interrupt occurs control is transferred to this routine; the output on the typewriter is of the following form:—

ERRINT PARITY 1234

The types of error indications possible are:—

POWER	Mains turned off
FLPT	Floating point overflow
RESAR	Reference to Reserved Area when protected
PDVCE	Reference to an unavailable peripheral device
PARITY	Parity Error
NOBIT	i.e. no indication present in the word 'L + 3' as to why the interrupt occured.

The integer, following the error interrupt indication is the location in the store at which the error occurred, it will be followed by S if it refers to the second half of the word.

On exit from this routine control is usually transferred to the message routine. In the store test control is transferred to a simple diagnostic routine if the error interrupt was caused by a parity error. In phase 1 of the function test floating point overflow is tested and there will be no output from the Error Interrupt routine if overflow occurs. If the floating point overflow should not have occurred then phase 1 of the function test outputs an error indication.

INPUT ROUTINE

The tape is punched in a special binary code. The input routine itself is read in by the initial instructions and contains a simple routine designed to cope with any error interrupt which may occur while the input routine is reading in the tape.

The error indications output by this routine are:—

E 1—Floating Point overflow

E 2—Usually spare, but could be output if Power supply fails.

E 3—Parity Error

E 4—Reference to unavailable Peripheral Device

E 5—Reference to protected reserved area.

R. A. Finch *January*, 1964.

503 BASIC TEST TAPE
OPERATING INSTRUCTIONS

The 503 basic test tape contains 4 separate tests. They are:—

TST01S Paper tape input/output, control station interrupt, and reserved area protection test

TST02S Store Test

TST03S Function Test

TST04S Typewriter and Word Generator Test.

The operating instructions are as below. For any error indication obtained while running the tests, see the relevant description of the test concerned.

	ACTION	EXPECTED RESULT
1.	Clear store	
2.	Load tape in reader (normal Mode)	
3.	Press INITIAL INSTRUCTIONS button	Tape read in
4.		TST03S displayed
5.		After 2¼ mins approx. TST03S COMPLETE displayed
6.		TST02S displayed
7.		After 2¾ mins approx. TST02S COMPLETE displayed
8.		TST04S displayed
9.	Clear Word Generator and ensure that tabs are set at 18 space intervals (see TST04S)	A displayed
10.	Depress all keys on Word Generator when B is displayed clear the Word Generator	B displayed
11.	Depress all keys on the input writer, as specified in the description of TST04S	
12.		Various patterns are printed on the output writer
13.		TST04S COMPLETE displayed
14.		TST01S displayed
15.		WAIT displayed
16.	Set address keys of the Word Generator to mode of output required as specified in TST01S description and change the sign of the Word Generator	Length of tape punched on the channels specified
17.		WAIT displayed
18.	Load first non-blank character of the length of tape into the respective reader and change the sign of the Word Generator	Reading/Punching cycle of paper tape test begins

	ACTION	RESULT
19.		End of reading/punching cycle results in last block of characters being read while blanks are punched
20.		A character is read and punched on each device being tested (with the exception of the input typewriter) to check control station control word
21.		A length of blank tape is punched; then tape is read and punched under interrupt control while TST03S is running. Thus TST03S is displayed for each channel tested
22.		PROTECT AND ENTER AT 7733 displayed. (For a non-standard reserved area proceed with the special reserved area test program.)
23.	(a) Release NO PROTN button (b) Press RESET button (c) Press MESSAGE button (d) type 7733.	Reserved Area tested for protection
24.		TST01S COMPLETE displayed, control transferred to the Message Routine

This is the end of the Test.

If no re-entry was made the time taken is approximately 12 mins. To enter at a specific test: —

(a) Press the MESSAGE button.

(b) Type entry point (as given below).

Re-Entry Points for the tests:

TST01S — 7370.

TST02S — 4630.

TST03S — 5380.

TST04S — 5140.

503 RESERVED AREA TEST PROGRAM FOR RESERVED AREAS OF A NON-STANDARD SIZE

FUNCTION

To test that the reserved area is protected.

STORE USED

Locations 259 to 309 and locations 8174 and 8175.

OPERATING INSTRUCTIONS

1. Unprotect Reserved Area.
2. Clear Store.
3. Place tape in reader and press INITIAL INSTRUCTIONS button.
4. PROTECT PRESS MESSAGE will be displayed.
5. Release the NO PROTN button.
6. Press RESET (the light in the NO PROTN button should go out).
7. Press MESSAGE.
8. If the test is completed correctly, PROTECTED will be displayed.

ERROR INDICATIONS

A. If the reserved area is not protected then NOT PROTECTED will be displayed.

B. If any other error interrupt occurs except 'reserved area', then EI NOT RA will be displayed and the contents of location 'L + 3' (i.e. the location containing the error bits) will be held in the accumulator.

PROCEDURE TO RESTART AFTER ANY ERROR

Restart from 1. of the operating instructions after any error.

TAPE

The tape is punched in the same binary code as the 503 BASIC TEST TAPE.

503 PAPER TAPE INPUT/OUTPUT, CONTROL STATION INTERRUPTS, AND STORE PROTECTION TEST

TST01S comprises four tests, they are:—

Phase 1—Paper tape input/output test.

Phase 2—Control Station control word test.

Phase 3—Control Station interrupts test.

Phase 4—Reserved area protection test.

PHASE 1—PAPER TAPE INPUT/OUTPUT

FUNCTION

To test that the readers and punches operate correctly. The program will test both channels or either channel, and each channel can be tested in normal mode or in any other mode fitted to the control station.

OPERATING PROCEDURE

On entry to this test TST01S will be displayed on the typewriter followed by WAIT. At this point the channel selection should be made on the Word Generator according to the following chart:—

N1 address = channel 1

N2 address = channel 2

For each channel the number of bits which are mixed into the accumulator for the given mode should be set thus:—

Setting on Address Keys	Input/Output
0	Channel not to be tested
5	5 bit
7	7 bit with parity (i.e. 8 hole tape)
etc.	etc.

Changing the sign of the Word Generator will then cause a tape to be punched on the selected channels, sufficient to reach the readers. (Care should be taken that the punches are loaded with tape and that the channels are set to the correct mode.) Load the first character of the tape produced into the respective reader and change the sign of the Word Generator, this will start the Reading/Punching cycle of the program.

Errors

There are two forms of error detection:—

(a) The checking circuits of the control station.

(b) An error detected by the program.

(a) An error detected by the logic of the control station will result in an error condition indicated by the light in the appropriate LOAD button.

Procedure: Restart test by pressing the MESSAGE button and typing 7370. the re-entry point.

(b) An error detected by the program will result in output on the typewriter in the form:—

C1 00111000 00111001 CYC No. 2

i.e. channel number (C1 or C2), then the correct version of the character followed by the version read, then the cycle number. Control will then be transferred to the message routine.

Procedure: Either type CONTERR. to continue from the point where the error occurred, or, if the program is 'out of step' restart by typing 7370. .

RE-ENTRY POINT—7370.

TIME TAKEN FOR PHASE 1—each cycle takes about 35 seconds and the test is set for 2 cycles, taking approximately 3 minutes including operating time.

Brief Description of Program

Two types of pattern are punched and read. A JU pattern (alternate 42 and 85) and a binary count. The program reads and punches alternately, each block of characters read or punched is split into sections, and associated with each section is a particular speed of reading or punching.

PHASE 2—CONTROL STATION CONTROL WORD TEST

This phase of the program checks the control word of the control station, and will output a ▲ during the test. If only one channel was tested in phase 1 then the channel not tested in phase 1 will be ignored in the control word test. If the control word indicates that a devise is busy or unbusy when it should not be, then the program outputs CONTROL WORD ERROR and control is transferred to the message routine, otherwise the program will continue with the interrupt test.

RE-ENTRY POINT FOR PHASE 2—7746.

TIME FOR PHASE 2—about 3 seconds.

PHASE 3—CONTROL STATION INTERRUPT TEST

Both interrupt lines are tested, i.e. the interrupt associated with the punches and the interrupt associated with the readers. The two input/output channels are tested separately, if only one channel was tested in phase 1 of this program then only that channel will be tested with interrupts.

The test procedure is as follows:—

A length of tape (100 inches) will be punched (not under interrupt control) on each of the channels to be tested, then reading (1000 characters) and punching (100 characters) occurs under interrupt control while the function test (TST03S) is running. Thus TST03S will be displayed on the typewriter at the beginning of the test on each channel. If the required amount of tape has been read and punched, the program transfers control to phase 4 (see below) which will output PROTECT AND ENTER AT 7733; Phase 3 should take less than 20 seconds. If the test does not work correctly then at the end of about 2 minutes TST03S COMPLETE will be output, but it should be obvious to the operator, if phase 4 is not entered after about 20 seconds, that something is wrong and he should halt the function test by pressing the MESSAGE button.

RE-ENTRY POINT FOR PHASE 3—7776.

TIME FOR PHASE 3—about 20 seconds.

PHASE 4—TEST RESERVED AREA FOR PROTECTION (256 locations)

If phase 3 is completed correctly then the following output will occur:—

PROTECT AND ENTER AT 7733.

Note.—Phase 4 of the 503 BASIC TEST procedure is to ensure that the last 256 locations of the store can be protected. This test will not be performed correctly if the reserved area differs in size. In this case ignore the message 'PROTECT AND ENTER AT 7733' and proceed with the special reserved area test program (already described).

Release NO PROTN button, depress RESET, (the light should go out in the NO PROTN button), depress the MESSAGE button and type 7733., the program tests if the the reserved area is protected and, if it is, will output TST01S COMPLETE, and control is transferred to the Message Routine, if the reserved area is not protected then NOT PROTECTED is output and control is transferred to the message routine.

RE-ENTRY POINT FOR PHASE 4—7816.

TIME TAKEN FOR PHASE 4—1 second + operating time.

TOTAL TIME FOR TST01S is about 4 mins but depends on the operating time.

RE-ENTRY POINT FOR TST01S—7370.

503 STORE TEST

The program contains three separate tests:—

Phase 1—Writes and checks a 'shifting 0 and 1' pattern in each store location.

Phase 2—Critical Pattern Test.

Phase 3—Random Numbers Test.

These are performed first with the program in the upper store testing the lower store, then with the program in the lower store testing the upper store. If an error occurs during the latter it is possible to continue by typing CONTERR. but none of the entry points can be used. It should be noted that locations 8174 and 8175 (Manual Interrupt and Error Interrupt locations respectively) are not tested. A complete cycle (i.e. testing both upper and lower store) takes approximately three minutes.

On entry TST02S is output on a new line; on completion of the test TST02S COMPLETE is output on a new line. If there is no intermediate output the test may be assumed to be successful.

ERROR INDICATIONS

There are two types of errors expected, in each case control is transferred to the message routine after the appropriate indications have been given; and the program can be continued by typing CONTERR.

(a) A discrepancy between the pattern stored and the pattern read back. This only applies to phases 1 and 3.

(b) Parity Errors.

Error Type (a)

An error of this form will result in the following form of output:—

ERR. PHASE 1 1234 CYC No. 1

The integer following the phase number indicates the address from which the 'wrong' pattern was read.

(CYC No. means cycle number)

Error Type (b)

On detecting a parity error control is transferred, as normal, to the Error Interrupt routine which will result in, say,

ERR INT PARITY 1234 being displayed

control is then transferred to a diagnostic routine which will result in, say,

PHASE 1 6789 CYC No. 1 being displayed

The integer following the Phase No. refers to the test location being accessed when the parity error occurred, a parity error may occur when the location containing the instruction to be obeyed is accessed, in which case this integer may not represent the location where parity occurred.

ENTRY POINT

Location 4630.

TIME

A cycle runs for approximately 3 minutes and 1 cycle is run.

503 FUNCTION TEST

This test comprises 2 phases:

Phase 1 a comprehensive function test using randomised numbers.

Phase 2 a Function Unit Test.

On entry to the test TST03S is output on a new line on the typewriter; on exit TST03S COMPLETE is output; if no intermediate output occurs the test may be assumed to be successful.

PHASE 1

Three pseudo random numbers a, b and c are generated each cycle such that a + b = c; this property is used for simple arithmetic tests, but simulators are used for functions such as multiply, divide, etc. Shifts are tested by moving one pattern a number of places determined by one of the other patterns. The results of all floating point functions are checked against those of simulated operations using fixed point arithmetic.

An error will usually result in output on the output writer of:

(i) the function that failed

(ii) a binary output of various registers of interest to the engineer

(iii) ERR PHASE 1

(iv) COUNT N (where N is the cycle number); the program will then attempt to repeat the test using the same operands. However some errors will output only the function that failed followed by S, i.e. 42S, where the S refers to some 'Simple' tests which are entered at the beginning of each cycle

PHASE 2

Various patterns are fed through the function unit, the patterns are designed to apply critical tests to the function unit with regard to the logical design.

In the event of an error, ERR PHASE 2 will be displayed on a new line and the test will restart from the beginning of phase 2.

ENTRY POINT FOR TST03S 5380.

TIME TAKEN

Approximately 2¼ mins (2500 cycles of phase 1 taking approximately 2 mins and 500 cycles of phase 2 taking approximately 15 secs).

503 TYPEWRITER AND WORD GENERATOR TEST

This test is comprised of four distinct phases:—

(a) Test the Word Generator for zero.

(b) Test the Word Generator for 'full house'.

(c) Test the Input Typewriter.

(d) Test the Output Typewriter.

On entry TST04S is displayed on a new line and on exit TST04S COMPLETE is displayed on a new line.

(a) After the heading is output the program will wait for the Word Generator to be zero. On finding it so, the upper case letter A will be output on the typewriter.

(b) After finding the Word Generator zero the program will wait for it to read 'full house', on finding this the upper case letter B will be output on the typewriter.

(c) After outputting B the program will wait for every key on the input typewriter to be depressed. The order in which they must be depressed is left to right, starting at the top left hand corner, in upper case, then repeat the procedure in lower case.
i.e.

012 - - - - - - - - - 1011
Tab QW - - - - - - - - - P = New Line
AS - - - - - - - - - ; |
Z - - - - - - - - - + -
SPACE

The characters input are checked against a stored table, any discrepancy will result in ERR being displayed on a new line, and control will be transferred to the beginning of this phase of the program, i.e. will be ready to accept another attempt at typing in the characters.

(d) After successful completion of this test the program displays a series of patterns and combinations of the characters to test the output writer. They are:—

(i) Outputs the full typewriter repertoire in the following format:—

( ) , £ : & * /
0 1 2 3 4 5 6 7 8 9 10 11
= + - . ;
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
| [ ] $_{10}$ < > ↑ ▲ % ?
a b c d e f g h i j k l m n o p q r s t u v w x y z

(ii) Outputs ('new line' 20 'spaces' ' ↑ ') 3 times thus showing if the output writer misses spaces, it should be checked visually that 20 spaces are output.

(iii) Ensures that the output writer is capable of dealing with characters of alternate case and outputs the following, twice:

8 [9] 10 $_{10}$ 11 < = > + ↑ - ▲ . % ; ? Aa Bb.....Zz

(iv) This test ensures that the tab and space functions are working correctly. The tabs should be set at 18 space intervals for this test otherwise the test may not work correctly. For a description of how to set tabs see 3.1.2. The test consists of typing 5 tabs each followed by +, then on the next line 5 tabs each followed by a space then + .

This is done twice in all; therefore the output should look like this:

(v) Variable speed test. The upper case alphabet is output with a varying delay between each character.

(vi) Outputs the 37 forbidden output writer codes in ascending order of value. These will be typed as 3 rows of 10 solid triangles and one row of 7 solid triangles. Thus all the output should be checked visually for any missing characters.

ENTRY POINT 5140.

TIME TAKEN

Approximately 3 mins (approximately 2 mins for phases (a) (b) and (c) and 1 min for phase (d)).

503 CORE BACKING STORE TEST PROGRAM

The program comprises 4 phases, which are:

Phase 1 To ascertain the size of the core backing store and print out the apparent number of words fitted.

Phase 2 A test using the A.D.T. system to write and read blocks of pseudo-random numbers, using the interrupt facilities to run a simple function test in the central processor during the core backing store 'busy' periods.

Phase 3 Using single word transfers, a 'chequerboard' pattern is set up in each of the units of core backing store fitted. (See Description of Program below for a more detailed description of phase 3).

Phase 4 To test that the main store is tagged while using the A.D.T. system.

TAPE

The tape is issued in SAC, but it is recommended that this tape is preserved as a master copy and a relocatable binary tape is made and used for the daily test of the core backing store.

DESCRIPTION OF PROGRAM

Phase 1 The apparent size of the store is determined by writing '+1' in the first location of each unit (i.e. 0, 16384, 32768,) and reading back from that location. The size of the store is determined when zero is obtained on reading back.

Phase 2 A block of 1024 pseudo-random numbers is generated, and this block of data is written into 128 consecutive areas of the core backing store by A.D.T. It is then retrieved by A.D.T.; when the data is retrieved from the backing store locations actually fitted it is checked for any discrepancy between the words written and the words read back. When it is retrieved from 'non-existent' backing store locations, it is checked for zero.

Phase 3 A chequerboard pattern is set up using single word transfers in a unit (16384 words) of core backing store. Each location is then:

(a) Checked that it contains the correct pattern ('full house' or zero).

(b) Written into with the inverse of the correct pattern.

(c) Checked that the inverse was stored correctly.

(d) Restored to its original state.

When each location of the unit has been tested in this way, the whole pattern in the unit is inverted and the checking procedure repeated.
Each unit of core backing store fitted is tested, separately, in this way.

The 'chequerboard' pattern written is of the form:

	0 64	4032
0	0 1 1 0 0 1 1	1 1 0
1	1 0 0 1 1 0	0 0 1
2	1 0 0 1 1 0	0 0 1
3	0 1 1 0 0	1 1 0
4	0 1 1 0 0	
5	1 0 0 1	
6	1 0 0	
	.	
	.	
	.	
61	1 0 0	0 0 1
62	1 0 0 1 1	0 0 1
63	0 1 1 0 0	1 1 0

Phase 4 A block of 1023 words, each word containing 'all ones', is transferred by A.D.T. to the core backing store. When the last main store location to be transferred becomes available (i.e. untagged), location 8186 (the A.D.T. count) is tested to see if it is negative. A block of 1023 words in the main store is cleared, and a block of 1023 words consisting of 'all ones' is read down from the core backing store into this area . The first and last main store locations referred to in the A.D.T. are tested. If either of them contain zero when they are read, they are considered untagged.

OPERATING INSTRUCTIONS

The operating instructions assume the program tape is coded in relocatable binary. In order to read in CBSTST, RAP must be in the store.

1. Read in the CBSTST tape by typing IN..
2. Ensure that the Reserved Area is unprotected and enter the test by typing CBSTST..
3. On entry to the test 503 CORE BACKING STORE TEST will be displayed, followed by NUMBER OF WORDS IS x where x is a 6-digit integer representing the apparent size of the core backing store. The operator should ensure that the figure printed is in fact the number of words of core backing store fitted to the machine.
4. Phase 2 is then entered. During this phase of the test, which uses the A.D.T. system and interrupts, FUNCTION TEST will be displayed twice, once for the writing cycle and once for the reading cycle.
5. Phase 3, the chequerboard test, is entered.
6. Phase 4 is entered. This writes and reads a block of words by A.D.T., testing the main store for tags. This is the last phase of the test. Thus on successful completion CBSTST COMPLETE will be displayed and control is transferred to RAP.

TIME TAKEN

Approximately (25 + 22N)secs, where N is the number of units fitted.

ERROR INDICATIONS

If the core backing store is unavailable, then on entry, immediately after the title is printed, an error interrupt will occur. There is no special facility for interpreting the error interrupt within CBSTST. (ERRINT 4 will be displayed by RAP).

Control is transferred to RAP after any of the following errors. A 'Detected Error' in the following description means an error which is detected by the parity checking logic of the core backing store. 'Undetected Error' means an error which has not been detected by the parity circuits of the core backing store.

Errors in Phase 1

If the apparent size of the core backing store is zero words then control is transferred to RAP.

Errors in Phase 2

(a) 'Detected' Errors. If an error condition is indicated in the control word after reading a block of locations then PARITY ADT TRANSFER is displayed. The block of locations read is checked against the original block of data.

If a discrepancy is discovered,

ERROR PHASE 2
LOCATION x
A
B

is displayed, where x is a 6 digit integer representing the backing store location in which the error occurred and A and B are the 39 bit binary patterns of the word written (A) and the word read back (B).

If it is correct

DATA CORRECT
LOCATIONS x-y is displayed,

x and y are two 6 digit integers representing the area of core backing store being accessed when the parity error was indicated.

(b) 'Undetected' Errors. If a discrepancy is discovered between a word written into the core backing store and the word read back

ERROR PHASE 2
LOCATION x
A
B

is displayed, x, A and B having the same significance as the error indication for 'detected' errors.

(c) Function Test Error. If an error occurs in the function test then

ERROR FUNCTION TEST N

is displayed, where N is an integer N denoting the number of the individual test within the function test that failed.

There are also error indications which are (almost) self-explanatory:
ERROR NOT BUSY DURING TRANSFER, i.e. bit 4 was not set in the control word during the A.D.T. transfer.

ERROR BUSY AFTER INTERRUPT, i.e., bit 4 was still set in the control word after an interrupt had occurred. (The program is written so that this should not be so).

ERROR PARITY DURING WRITING, i.e., bit 5 was set in the control word while writing blocks of data to the core backing store.

ERROR PARITY AFTER CLEARING ERRORS, i.e., after clearing the errors at the core backing store (by a 75 5120 order) bit 5 was still set in the control word.

ERROR PARITY READING FROM NON-EXISTENT LOCATIONS, i.e., bit 5 was set in the control word while reading from non-existent locations.

Errors in Phase 3

There are two forms of error indication

(a) Detected Errors.
(b) Undetected Errors.

(a) Detected Errors

If a parity error is indicated after reading a core backing store location into the core backing store register

PARITY ERROR CHEQUERBOARD
LOCATION x
A
B

will be displayed. x is the backing store location being accessed when the error occurred; A and B are, respectively, the binary configurations of the word written into the location (A) and the word read back (B).

(b) 'Undetected' Errors

If a discrepancy is found between a word written to the core backing store and the word read back, then

ERROR CHEQUERBOARD
LOCATION x
A
B

will be displayed. x, A and B have the same significance as in 'Detected' Errors (see above).
N.B. - The binary print routine will print out ALL ONES or ALL ZEROS if the word to be printed is 'full house' or zero.

Errors in Phase 4

(a) If a parity error occurs while writing then the indication is the same as in Phase 2.
(b) If a location is found to be untagged then TAG ERROR will be output on a new line.

R. A. Finch. *November, 1964*

503 CARD READER/PUNCH TEST PROGRAM

The program comprises 3 phases which are:

Phase 1 The standard pack of cards is copy punched using the interrupt facilities to run a simple function test in the central processor during the card reader/punch 'busy' periods.

Phase 2 The copies punched in Phase 1 are read and checked, again under interrupt control.

Phase 3 A facility for producing a standard pack if required, not under interrupt control.

TAPE

The tape is issued in SAC, but it is recommended that this tape is preserved as a master copy and a relocatable binary tape is made and used for the daily test of the card reader/punch.

DESCRIPTION OF PROGRAM

The program uses a standard pack of 280 cards, 180 of Type A, 90 of Type B, and 10 blank cards. Card A fulfils the following requirements:

(a) contains all permitted codes at least once

(b) has no blank columns

(c) the total number of holes in each row is odd.

Card B fulfils the following requirements:

(a) contains all forbidden decodes

(b) the total number of holes in each row is even.

These cards are shown in Figure 1.

The program is divided into 3 phases

Phase 1 The standard pack is copy punched under interrupt control, while a simple function test is being run during the card reader/punch 'busy' periods. The cards are checked against a stored image as they are read in and then copies are punched in the following order:-

(a) 90 A type cards, binary image to binary image (i.e. bits 28 to 39 with bit 27 false).

(b) 90 A type cards, decode to decode (i.e. bits 1-7, with bit 27 true).

(c) 90 B type cards, binary image to binary image. In addition 10 B type cards are attempted to be punched decode to decode. These 10 cards will be blank since the forbidden code will not be recognised.

The copies and the 10 blank cards are sent to pocket P2 whilst the standard pack exits at the main stacker. Phase 1 involves about 560 card movements, therefore the total time taken should not be more than 2 minutes.

Phase 2 The track is cleared and the copies from the pocket P2 are placed in the main hopper. The program is entered to read and check, under interrupt control, that the copies have been correctly punched, while a function test is in progress.

Phase 3 This is a facility for producing a standard pack. It does not operate under interrupt control.

OPERATING INSTRUCTIONS

The operating instructions assume the program tape is coded in relocatable binary. In order to read in CRPTST, RAP must be in the store.

Phase 1 and 2

1. Read in CRPTST tape by typing IN.

2. Clear the main track (depress CL.M) and load the main hopper of the card reader/punch with the standard pack (see below to obtain standard pack) and load at least 280 blank cards in the sub-hopper.

3. Depress buttons ($\frac{R}{P}$ + S), START and END on the card reader/punch control panel, ensuring that the lamp within each button is lit.

4. Ensure the Reserved Area is unprotected and enter the test by typing CRPTST.

5. On entry to the test
   503 CARD READER PUNCH TEST
   FUNCTION TEST
   is displayed and the standard pack copy punched under interrupt control.
   The standard pack is ejected into the main stacker, the copies (the last ten cards of which should be blank) are deflected into pocket P2, and
   END PHASE 1
   WAIT
   is displayed on satisfactory completion of Phase 1.

6. Clear main track. Remove and preserve the standard pack from the main stacker. Remove the copies from pocket P2 and place them in the main hopper.

7. Depress START and either change the position of Key 39 of the word generator or type CRPTST; 2. to begin phase 2 of the reading cycle, when
   PHASE 2
   FUNCTION TEST
   is displayed. The copies made in phase 1 are read and checked against a stored pattern under interrupt control, and ejected into the main stacker.
   CRPTST COMPLETE
   is displayed on satisfactory completion of phase 2.

Phase 3 to obtain a standard Pack of cards

1. Clear the main track and place at least 280 blank cards in the subsidiary hopper.

2. Press ($^{R}_{P}$ + S), START, and END

3. Type CRPTST; 3.

4. On completion
   END
   is displayed and the standard pack produced is found in the main stacker. The pack can be checked by entering Phase 2.

ERROR INDICATIONS

1. In the event of a read error CRPTST will stop and either
   (a) NOT AVAILABLE
   DETECTED READ ERROR
   is displayed, and lamp RE is lit.
   Procedure:- Press START, this moves the error card to pocket P1 and stops.

or

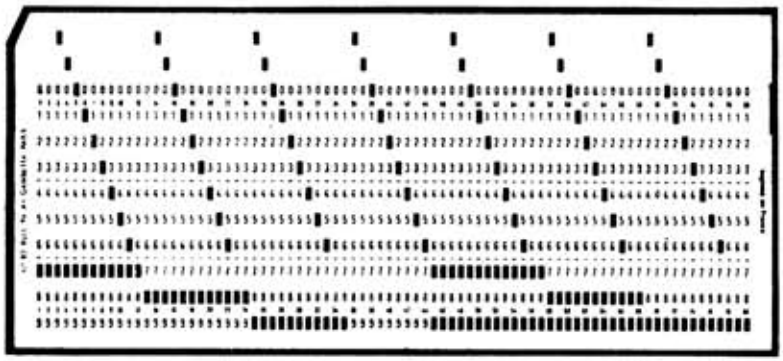
   (b) UNDETECTED READ ERROR
   is displayed in which case the program sends the error card to pocket P1 and stops.

2. In the event of a detected punch error
   NOT AVAILABLE
   DETECTED PUNCH ERROR
   is displayed, the card reader/punch stops with the lamp PE lit and the error card in pocket P1. An undetected punch error will be discovered in Phase 2 which checks the cards punched in Phase 1 against a stored pattern.

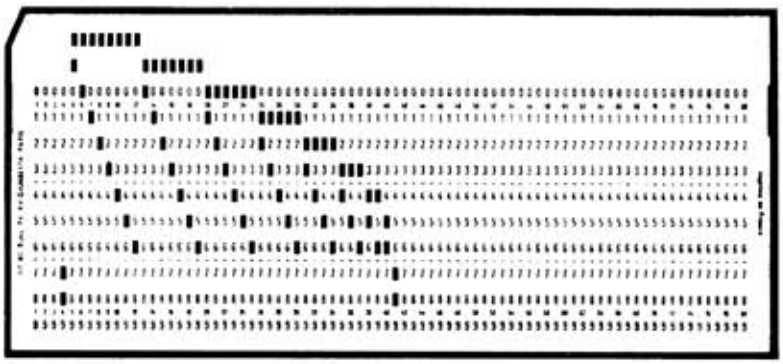
   There are also the following error indications:-

NOT AVAILABLE - i.e. bit one is set in the control word.
NOT RPS - i.e. the correct mode of reading/punching (bit 9 of the control word true and bits 7, 8 and 10 false) is not set.
BUSY AFTER INTERRUPT
- i.e. bit 4 was still present in the control word after an interrupt occurred. The program is written so that this should not happen.
ERROR FUNCTION TEST n
The simple function test is run during the card reader/punch 'busy' periods, n is the number of the particular test within the function test that failed.

R. A. Finch *December, 1964*



TEST CARD 'A'



TEST CARD 'B'

Figure 1