Tektronix
4050 Series

Applications Library
Program Documentation

Applications Library

Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library
Applications Library

PROGRAMMING AIDS T1

Prepared by the 4050 Series Applications Library. The 4050 Series Applications Library is maintained as a service for our customers by the Information Display Division of Tektronix, Inc., Group 451, P.O. Box 500, Beaverton, Oregon 97077 U.S.A.

Tektronix®
COMMITTED TO EXCELLENCE

PROGRAMMING AIDS T1

062-5971-01

DOCUMENTATION

Applications Library
Group 451
Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97007

062-5971-00

6/81

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	PART NUMBER
PROGRAMMING AIDS T1	062-5971-01
ORIGINAL DATE: June, 1981 / REVISION DATE:	

ABSTRACT

PROGRAMMING AIDS T1 is a tape collection of 14 programs to aid you in creating or dissecting a 4050 BASIC program. Employ these routines to produce your overlays, structure program flow, track variables, convert from FORTRAN to BASIC, draw flow diagrams and aid you in other programming techniques.

The individual abstracts describe the programs.

Tape file 1 contains the directory. Press AUTOLOAD and follow the instructions.

Be sure to read the documentation before running a program.

TITLE	PART NUMBER
PROGRAMMING AIDS T1	062-5971-01

TITLE	PART NUMBER
PROGRAMMING AIDS T1	062-5971-01

TRANSFERRING FILES TO A NEW TAPE

PLOT 50 General Utilities Vol. 1 (TEKTRONIX Part #4050A08) contains a program to transfer any type of 4050 files (program/data/text) quickly and easily along with the header names; however, it requires a 4924 Tape Drive.

Transferring ASCII or BINARY PROGRAMS without a transfer program

Step 1. Do a TLIST of the MASTER program tape.

Step 2. Record which files go with which program (they are all named) and the size of each file.

Step 3. MARK your new tape to accept the respective files for that program, e.g.,

```
FIND 0
MARK 1,20000
FIND 2
MARK 1,4000
etc.
```

Step 4. Insert the MASTER tape.

FIND a file

OLD for ASCII or CALL "BOLD" for BINARY

Step 5. Insert the new tape

FIND the file to receive the file in memory

SAVE for ASCII or CALL "BSAVE" for BINARY

REPEAT Steps 4 and 5 until all files comprising that program are transferred to the new tape. Note: This procedure will not retain the file header names.

Transferring ASCII or BINARY DATA to a new tape

The 4051R06 Editor ROM could be used to transfer ASCII DATA files.

4050 Applications Library program "Binary Data File Duplicator" will transfer BINARY DATA files without any peripheral.

4050 Applications Library program "Tape Duplication" will transfer ASCII or BINARY DATA or PROGRAM files, but requires a 4924 Tape Drive.

Both of these programs are contained on the 4050 Applications Library UTILITIES T1 tape (TEKTRONIX Part #062-5974-01), and UTILITIES D1 disk (TEKTRONIX Part #062-5975-01).

Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
Overlay Drawing Program		
ORIGINAL DATE	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED
February, 1980		16K
AUTHOR	Tektronix, Inc.	PERIPHERALS
LeRoy Nollette	Wilsonville, OR	4662 Plotter

ABSTRACT

Files: 1 ASCII Program
2 ASCII Data (Sample Overlays)
Optional - Pre-MARKed data files

The program draws an overlay on the 4662 Plotter that can be cut out and placed over the User-Definable Keys. Key descriptions may be entered from the keyboard. Data may be saved on a pre-MARKed data file and redrawn at a later date.

The program may be modified (one line of code) to draw a large copy of the overlay and then reduce it on a copy machine having reduction capabilities.

By changing one line of code, the user may preview the overlay on the screen.

TITLE

Overlay Drawing Program

DESCRIPTION

The program allows the user to create a very nice, printed overlay using the 4050 Series Desktop Computer and the 4662 Plotter. The printed overlay may be cut out and glued to an overlay.

The titles and key descriptions may be saved to a pre-MARked file, allowing the user to print the overlay again at a later time.

OPERATING INSTRUCTIONS

The 4662 Plotter should be on and set to the default page size (same as when turned on). The device address should be set at 1 (factory setting). If the device address is not 1 change the value of T in line 120 to the correct address.

Load a piece of paper and remove the pen cap.

Press AUTOLOAD (if the program is located on the first file) or:

```
FIND #  (file number)
OLD
RUN
```

Respond to the questions as they are asked.

TITLE

Overlay Drawing Program

Program Execution

```
OVERLAY DRAWING PROGRAM

(1)TAPE
(2)KEYBOARD
(3)BLANK

 SELECT SOURCE OF KEY LABELS
```

(1) TAPE

Allows the user to draw an overlay that has been created and saved on a pre-MARked tape file.

(2) KEYBOARD

Allows the user to enter from the keyboard the titles and key labels.

(3) BLANK

Allows the user to draw a blank overlay.

TITLE

Overlay Drawing Program

(1) TAPE

```
OVERLAY DRAWING PROGRAM

(1)TAPE
(2)KEYBOARD
(3)BLANK

 SELECT SOURCE OF KEY LABELS 1
INPUT FILE NUMBER
```

Input the file number of the overlay to be drawn.

The plotter pen will move to the lower left corner.

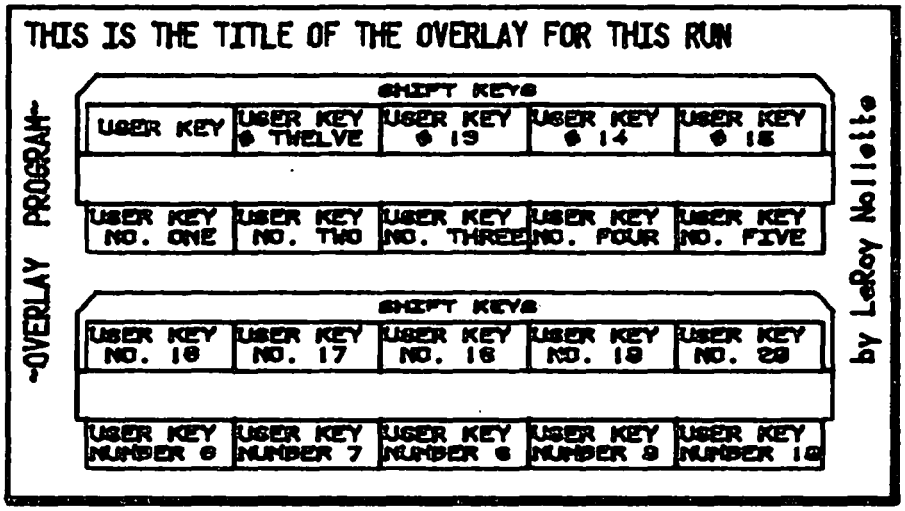
The pen may be positioned to any location. Be sure to leave sufficient room to the right and above to draw the overlay. The pen position will be the lower left corner of the overlay.

When the pen is in position press the CALL button on the plotter. The overlay will then be drawn.

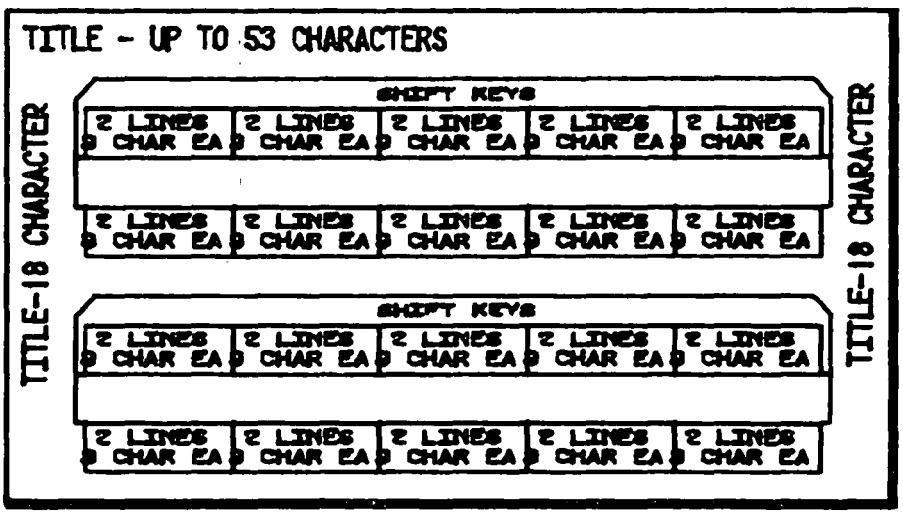
TITLE

Overlay Drawing Program

Example of file 2



Example of file 3



TITLE

Overlay Drawing Program

(2) KEYBOARD

```
OVERLAY DRAWING PROGRAM

(1)TAPE
(2)KEYBOARD
(3)BLANK

 SELECT SOURCE OF KEY LABELS 2
PLEASE ENTER THE TITLE OF THE OVERLAY

_____________________________________________
ENTER THE TITLE ON LEFT EDGE OF OVERLAY _________________
ENTER THE TITLE ON RIGHT EDGE OF OVERLAY _________________

HOW MANY LINES IN DESCRIPTION OF KEY 11 2
ENTER DESCRIPTION OF LINE 1 FOR KEY 11 _________
ENTER DESCRIPTION OF LINE 2 FOR KEY 11 _________

HOW MANY LINES IN DESCRIPTION OF KEY 12 1
ENTER DESCRIPTION OF LINE 1 FOR KEY 12 _________

HOW MANY LINES IN DESCRIPTION OF KEY 13 0
```

The title of the overlay may contain up to 53 characters.

The title on the left edge may contain up to 18 characters.

The title on the right edge may contain up to 18 characters.

The key labels may contain up to 2 lines of 9 characters each.

The program asks for the number of lines of data for each key; if no label is desired enter Ø.

```
DO YOU WANT THE DATA STORED TO TAPE NO
```

The plotter pen will move to the lower left corner.

TITLE

Overlay Drawing Program

The pen may be positioned to any location. Be sure to leave sufficient room to the right and above to draw the overlay. The pen position will be the lower left corner of the overlay.

When the pen is in position, press the CALL button on the plotter. The overlay will then be drawn.

DO YOU WANT THE DATA STORED TO TAPE YES
INPUT FILE NUMBER

Input the file number of a pre-MARked file for the data to be saved.

The plotter pen will move to the lower left corner.

The pen may be positioned to any location. Be sure to leave sufficient room to the right and above to draw the overlay. The pen position will be the lower left corner of the overlay.

When the pen is in position press the CALL button on the plotter. The overlay will then be drawn.

NOTE: Files should be marked for 1000 bytes.

TITLE

Overlay Drawing Program

(3) BLANK

OVERLAY DRAWING PROGRAM

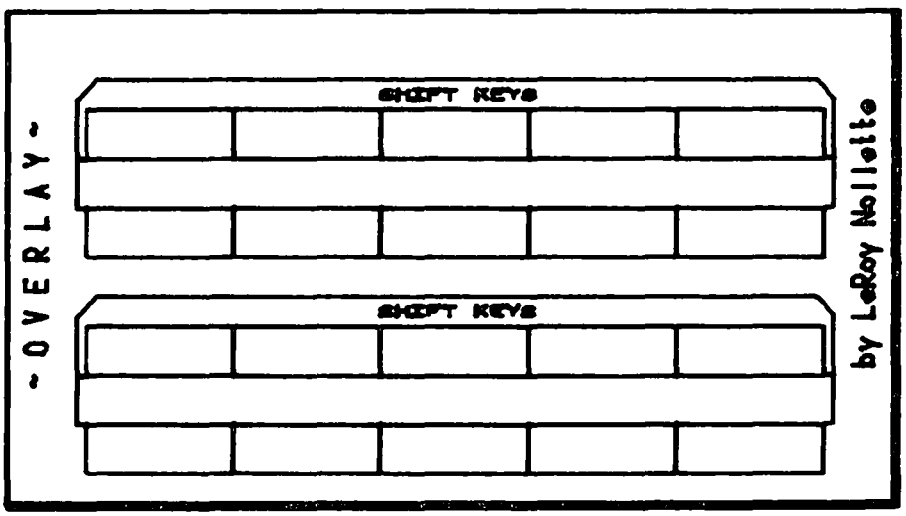
(1)TAPE
(2)KEYBOARD
(3)BLANK

SELECT SOURCE OF KEY LABELS 3

The plotter pen will move to the lower left corner.

The pen may be positioned to any location. Be sure to leave sufficient room to the right and above to draw the overlay. The pen position will be the lower left corner of the overlay.

When the pen is in position press the CALL button on the plotter. The overlay will then be drawn.



TITLE

Overlay Drawing Program

SPECIAL FEATURES

Screen Preview

The overlay may be previewed on the screen. The character rotation and the variable character sizes are not available on the screen. A full screen version will be displayed.

Change:

```
120 T=1  to  120 T=32
```

Enlarge Overlay

The overlay may be enlarged and then reduced on a copy machine with reduction capabilities.

Line 700 is currently:

```
700 S1=1
```

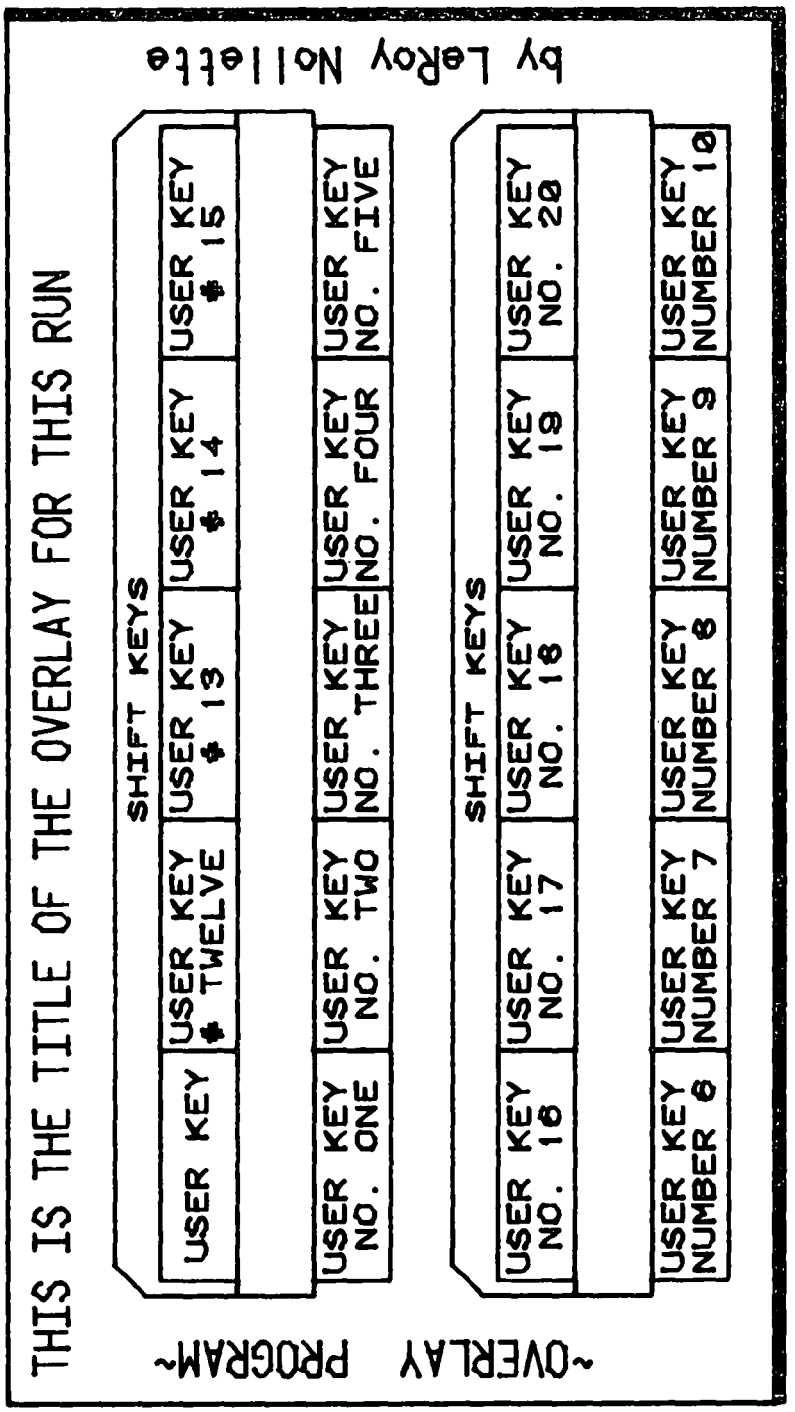
If a copy machine reduces 64%, change line 700 to:

```
700 S1=.64
```

the resulting plot would be 64% larger and after being reduced would give a clear picture the proper size for an overlay.

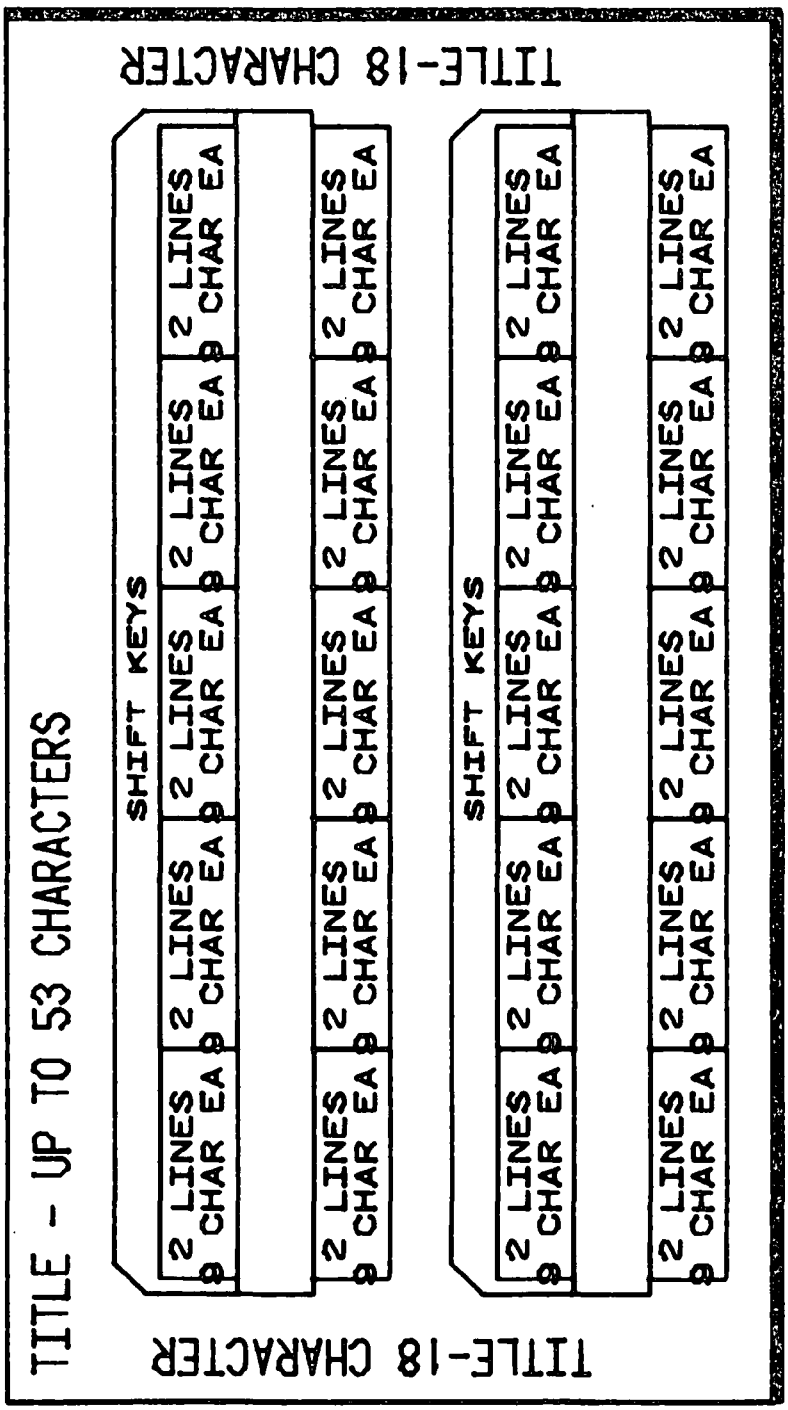
TITLE

Overlay Drawing Program



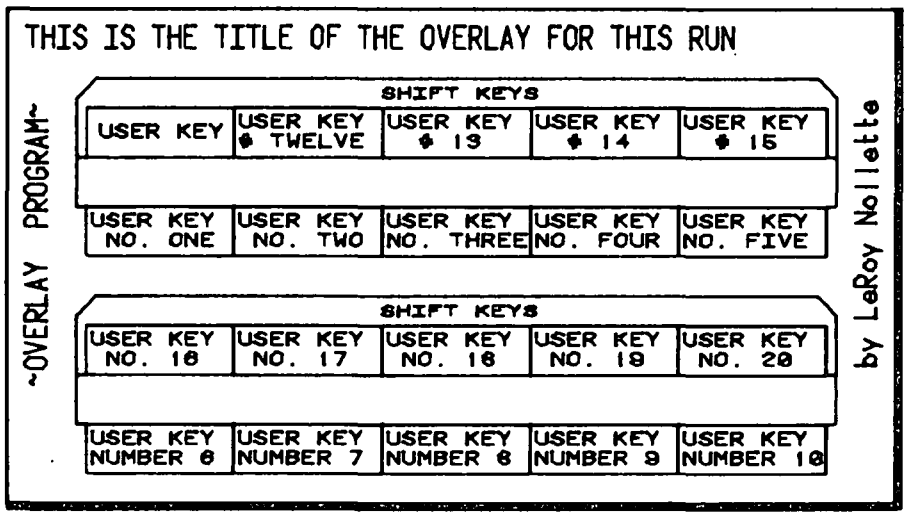
TITLE

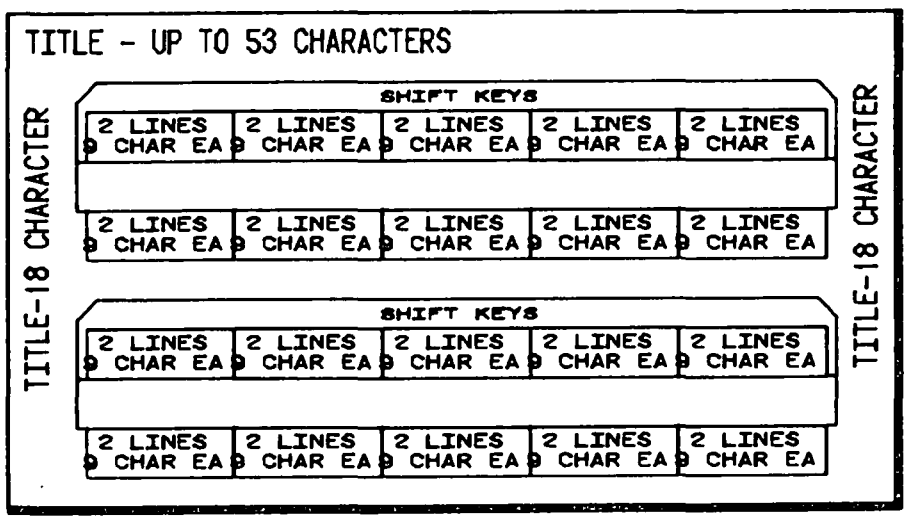
Overlay Drawing Program



TITLE

Overlay Drawing Program





Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	
Enhanced Program Listings	
ORIGINAL DATE: September, 1980 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 8k
AUTHOR: Tim Giesbers, Tektronix, Inc., Beaverton, OR	PERIPHERALS: Optional-4641 Printer

ABSTRACT

Files: 1 ASCII Program

Statements: 144

The program will list any ASCII program file, or consecutive files, stored on tape.

The list can be either to the 4050 screen or a 4641 Printer. If the list is to the screen, copies may be made automatically on a 4631 Hard Copy Unit.

The listing includes file numbers and the length of each file is given in bytes at the end of the listing.

Statements inside FOR/NEXT loops are indented, and REM statements are separated from other program lines by a blank line for emphasis.

New pages are automatic with the user specifying the number of lines per page and the length of the pause between pages. There is no provision for wraparound or truncation of a line which is longer than the width of the printer paper.

User input:

First file number
Last file number
Output device number
Automatic copies YES/NO
How many lines per page
How many seconds of pause

TITLE

Enhanced Program Listings

The program operates by reading a line of code, then checking it for control characters, converting any it finds to its letter equivalent, followed by a backspace and an underscore. The keyword of the statement is found, and spaces are added for indenting if the statement is in a FOR/NEXT loop, or a blank line is printed, if the statement is a REM. A new page and automatic copy is done, if needed, then the next line is read in. After the last line is printed, the length of the file is given, and the program goes to the next file, or ends.

The user enters the beginning and ending file numbers, the number of lines to be printed per page, the number of seconds to pause at the bottom of each page, and the device number the listing is to go to. If the device is 32 (the screen), the user is asked whether or not to automatically copy the listing.

A list of variables used and their purposes is at the beginning of the program listing.

Two examples of the user's input are below. In the first, the underlined defaults are used. This puts the listing on the screen, does not automatically copy it, prints 33 lines per page, and pauses for two seconds at the bottom of each page. The second example shows a listing being sent to a printer (device=41), no pause, and 55 lines per page.

```
ENHANCED PROGRAM LISTINGS


First file to list:  17
Last file to list:  17

Output device (32):
Automatic copies (Y/N):
How many lines per page (33):
How many seconds of pause (2):
```

```
ENHANCED PROGRAM LISTINGS


First file to list:  17
Last file to list:  17

Output device (32):  41
How many lines per page (33):  55
How many seconds of pause (2):  0
```

TITLE

Enhanced Program Listings

One problem may occur when using the automatic copy feature. If you specify 0 seconds of pause, and the last page has only a few lines on it, the program will try to copy the last page before the previous one is finished inside the copier, and you'll lose the copy of the last page. To avoid this, let the program default to the two seconds of pause. This will give the copier a chance to recover from each copy before trying to begin another. This problem is noticeable on a 4052 or 4054, but may not be on a 4051.

The listing of the program is on the following pages. This listing is also its own sample run.

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
REMARK OUTLINER		
ORIGINAL DATE August 1979	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED 8K
AUTHOR Mallory M. Green U.S. Dept. of HUD Washington, D.C.		PERIPHERALS

ABSTRACT

Statements: 140

Files: 1 ASCII Program

REMark Outliner is intended as a tool for the programmer who writes a structured program. It inputs a structured ASCII program and prints out a program outline. The outline includes subroutine names, line numbers and flow between subroutines.

The following programming techniques are required for REMark Outliner to work effectively.

1. Subroutines make up the program with GOSUB or GOSUB OF statements controlling program flow.

2. Subroutines begin with REMark statements describing the subroutine's function. These REMark statements are separated from other REMark statements by special characters; i.e., REM* or REM/ and so on.

3. Hierarchical subroutines.

4. Program's name contained in first program REMark.

REMark Outliner uses the special REMark statement to identify the modules and it traces program flow only through GOSUB or GOSUB OF statements. It makes two passes through a program: the first pass creates a table of subroutine locations; the second pass prints the program outline.

TITLE

REMARK OUTLINER

Methodology

The REMARK OUTLINER routine reads an ASCII program from tape and generates a program outline. In order to generate an outline, the program file is read twice. The first pass through the program file generates a table of subroutine locations. The second pass of the file is used to actually print the program outline. The OUTLINER routine only traces the program flow accomplished through either "GOSUB" or "GOSUB OF" statements.

The following programming techniques are needed for this program to work effectively:

1) The program should be made up of subroutines.
2) Each subroutine should begin with a remark statement describing the subroutine's function.
3) The program should be designed in a hierarchical manner.
4) Program flow between subroutines should be accomplished using either "GOSUB" or "GOSUB OF" statements.
5) The first program remark should contain the program's name.
6) Program name and subroutine name remarks must be identifiable. As an example, each of these remarks may start with "REM *, where other remarks may start with "REMARK" or "REM".

TITLE

REMARK OUTLINER

User Instructions

The first step in using this program is to copy each program file to be outlined onto the internal tape cartridge in ASCII format using the "SAVE" command. Next, the REMARK OUTLINER program is loaded into memory and a "RUN" is issued. The program then prompts the user to enter the type of remark used to identify subroutines, such as, "REM * ". The user is then asked the tape file number where the program to be outlined is located.

TITLE

REMARK OUTLINER

Program Outline

REMARK OUTLINER PROGRAM '$OUTLINE' - 7/24/79

M#	LINE#	MODULE NAME	MODULES CALLED
1	110	CONTROL MODULE	2 3 4 5
2	170	SELECT REMARK TYPE AND FILE NUMBER	
3	300	FIRST REMARK FOR PROGRAM TITLE	
4	410	FIRST PASS TO RECORD MODULE LOCATIONS	
5	530	SECOND PASS CONTROL MODULE	6 7 8
6	680	PRINT FIRST REMARK AS PAGE TITLE	
7	760	IF MODULE NAME - PRINT	6
8	980	EVALUATE GOSUB TYPE	9 10
9	1070	NORMAL GOSUB TYPE	11
10	1120	GOSUB OF TYPE	11 11
11	1240	PRINT GOSUB MODULE NUMBER	
12	1420	SET EOF FLAG	

TITLE

REMARK OUTLINER

Example Outlines

BARGRAPH II - '@B2/MAIN' - LAST REVISED 7/23/79

M#	LINE#	MODULE NAME	MODULES CALLED
1	2	USER KEYS	12 2 3 8 13 13
2	100	PROGRAM INTRO & INITIALIZATION	
3	1090	PROGRAM RESTART	4 6 5 10 11
4	1190	NEW CHART	13
5	1240	READ GRAPH FROM DISK	7
6	1370	READ PLOT FROM TAPE	7
7	1463	TIME TO BAR CONVERSION	3
8	1480	SELECT BAR FORMAT	9
9	1897	CHECK PLOTTER	9
10	1928	LIST PLOTS ON TAPE OR DISK	
11	2030	DELETE PLOTS FROM DISK OR TAPE	
12	2240	SAVE GRAPH TITLES AND DATA	
13	2650	OVERLAY CONTROL MODULE	

"@B2/SUB" BARGRAPH II OVERLAY

M#	LINE#	MODULE NAME	MODULES CALLED
1	2880	CONTROL MODULE	2 7 8 9 10 11 12 13 2 23 24 25 26
2	2940	CREATE A NEW BARGRAPH	3 4 5 6 22 18 20
3	3030	TITLE LINES	
4	3230	NUMBER OF GROUPS & LABELS	
5	3410	NUMBER OF BARS & LABELS	
6	3590	AXIS LABEL	
7	3670	DATA VALUES DISPLAY	14 15 17
8	3750	ADD GROUPS	15 16 18 19
9	4030	DELETE GROUP	18 20
10	4200	ADD BAR TYPE	15 18 19
11	4550	DELETE BAR TYPE	18 20
12	4710	CHANGE TITLES	14
13	4950	MAKE DATA CORRECTIONS	15 17 18 19
14	5150	PRINT TITLES	
15	5240	PRINT BAR LABELS AS WORDS	
16	5510	INPUT DATA ROW	
17	5620	DISPLAY DATA ROW	
18	5770	GLOBAL MIN MAX	
19	5890	AUTO SCALE IF NEEDED ONLY	20
20	5930	AUTO SCALE	21 21
21	6210	SELECT END OF SCALE	
22	6450	DATA ENTRY	15 16
23	6550	CHANGE AXIS SCALE	
24	6790	CHANGE AXIS LABEL	
25	6900	CHANGE GROUP LABELS	
26	7140	CHANGE BAR LABELS	

TITLE

REMARK OUTLINER

Example Outlines Continued

"@B2/HOR" BARGRAPH II OVERLAY

M#	LINE#	MODULE NAME	MODULES CALLED
1	2880	HORIZONTAL PLOT CONTROL	2 3 4 5 6 20 7 8 16 15
2	3010	PLOT SETUP	
3	3220	EVALUATE DATA VALUES	
4	3330	PLOT AXIS	
5	3430	PRINT X-AXIS TIC VALUES	
6	3620	PRINT X-AXIS LABEL	
7	3700	PRINT TITLES	
8	3920	PRINT BAR LABELS	9
9	4090	RIGHT JUSTIFY AND PRINT STRINGS	
10	4200	DRAW BOX FOR LEGEND	
11	4290	DRAW LEGEND 1	10 18
12	4400	DRAW LEGEND 2	10 19 18
13	4520	DRAW LEGEND 3	10 12
14	4610	DRAW LEGEND 4	10
15	4690	PRINT DATA VALUES	
16	5090	DRAW AND SHADE GROUPS	
17	5140	DRAW LEGENDS	11 12 13 14 11 12 12 17
18	5340	HORIZONTAL GROUP SHADE ROUTINE	
19	5470	VERTICAL GROUP SHADE ROUTINE	
20	5640	LABEL GROUPS	9

"@B2/VER" - BARGRAPH II OVERLAY

M#	LINE#	MODULE NAME	MODULES CALLED
1	2880	VERTICAL FORMAT PLOT CONTROL	2 15 3 4 5 6 10 21 7 17 16
2	3140	DEFINE PLOT AREA IN GDU'S	
3	3350	DRAW AXIS	
4	3460	PRINT Y-AXIS TIC VALUES	26
5	3620	PRINT Y-AXIS LABEL	
6	3730	PRINT TITLES	
7	3940	PRINT BAR LABELS	8
8	4110	RIGHT-JUSTIFY AND PRINT STRINGS	
9	4230	DRAW BOX FOR LEGEND	
10	4320	DRAW X-AXIS TICS	
11	4430	DRAW LEGEND 1	9 19
12	4540	DRAW LEGEND 2	9 20 19
13	4660	DRAW LEGEND 3	9 12
14	4750	DRAW LEGEND 4	9
15	4830	EVALUATE DATA VALUE DISPLAY	
16	5080	DISPLAY DATA VALUES	
17	5430	DRAW AND SHADE GROUPS	
18	5480	DRAW LEGENDS	11 12 13 14 11 12 12 18
19	5650	VERTICAL GROUP SHADE ROUTINE	
20	5780	HORIZONTAL GROUP SHADE ROUTINE	
21	5950	LABEL GROUPS	
22	5980	PRINT GROUP LABELS ON PLOTTER	8
23	6100	DRAW CHARACTERS ON SCREEN	
24	6160	DRAW GROUPS LABEL RIGHT JUSTIFIED	
25	6480	LOOP TO DRAW CHARACTER	25 25
26	6800	# OF DECIMAL PLACES	

TITLE

REMARK OUTLINER

Program Variables

A$	- Program line input string
E	- EOF flag (1-EOF, Ø-not EOF)
F	- Program file #
H$	- GOSUB #'s string
I$	- GOSUB OF # string
I	- Line counter
I9(50)	- Remark #'s array
J	- Scratch
K	- GOSUB calls counter
KØ	- GOSUB calls maximum per line
L$	- Remainder on long remarks
L	- Scratch
M$	- Subroutine label from remark
M	- Module number
N$	- Line # string for remark
S$	- Remark type
S	- Length of S$
S1	- GOSUB call line #
T$	- Title remark
T	- Total # of modules

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
Tape Directory		
ORIGINAL DATE	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED
December, 1978		8K
AUTHOR	Comalco Aluminum Ltd.	PERIPHERALS
Nick Ogbourne	Tasmania, Australia	Optional - 4051R06 Editor ROM

ABSTRACT

Files: 1 ASCII Program
1 ASCII Text

Statements: 90

The program, located as the first ASCII program file on a tape, operated using the AUTOLOAD, provides a tape "directory," multipage if necessary, and controls access to, and execution of any required program files.

The user creates and maintains an "index" in File 2 (ASCII) which provides file number, program name and program description to the DIRECTORY program. An example of the index is included.

It is not necessary to specify to the DIRECTORY the type of the program (ASCII or Binary). Programs not required to be accessed by the DIRECTORY, i.e., files called by main modules of programs, data files and text files may also be recorded in File 2, providing a rapid means of "TLIST"ing a tape.

TITLE

Tape Directory

PRELIMINARY OPERATING INSTRUCTIONS

Transfer "TAPE DIRECTORY" to the first file of your tape. Using a simple BASIC program or the Editor ROM, create and maintain an index in file 2 according to the following structure:

Data File Structure for INDEX

File 2, on the same tape as the DIRECTORY is an ASCII file--data or text--together with the variable names used to read the file.

T	Tape number)	Used for 'in house' tape usage control
T$	Tape name)	and backup file protection

P(i)	File number
P$	Program name
D$	Program description (maximum 50 characters)

P(i)	)
P$	) Repeated for each program
D$	)

Internal Data Storage

Variable	Used to Store	Type
X	Company Logo	Array (44)
T	Tape Number	Simple
T$	Tape Name - then Scratch	String
I	Number of directory-controlled program files	Simple
S	Scratch	Simple
P$	Program Name	String (8)
D$	Program Description	String (50)
X1) Y1)	Scratch	Simple
S$	Scratch	String
P	Directory-controlled file numbers	Array (100)

TITLE

Tape Directory

METHODOLOGY

DIRECTORY reads file 2, printing the tape number and tape name, then the program name, the program number, which is the count of directory-controlled programs listed in file 2, and the program description.

The file number of the program number count (i.e., i) is stored in array P, such that P(i) contains the file number of program i. Any file numbers recorded in file 2, which are greater than 255 are read and ignored. Thus, data files or non-directory controlled files may be recorded in file 2.

Each page contains 23 programs and following a full page, a reply of a valid program number will load the relevant program as described below, or as instructued, a blank RETURN prints the next page.

A non-full page omits the RETURN prompt.

A response to the 'Select Program' prompt is checked to ensure that the reply is an integer, greater than Ø and not greater than the maximum number of programs on file. The user is warned (bell) of invalid entries and another selection may be made.

Following a valid selection, recorded in variable S, file P(S) is found, the header read (using PRI @33,Ø:Ø,Ø,1 (no header mode) and a check made to ensure that the file is not a data file (character 17). If it is a data file, the user is prompted for another selection.

If the file is not data, the file type character (9) is accessed.

If the character is 'B', a binary 'OLD', i.e., CALL "BOLD" is executed; otherwise the file will be ASCII (program or text) and an OLD command is executed.

TITLE

Tape Directory

OPERATING INSTRUCTIONS

Once your index file is created, operation is automatic on pressing the AUTOLOAD key (providing the DIRECTORY program is stored on file 1 and the index on file 2).

The directory is listed; if there are less than 23 programs controlled by the directory, the user is prompted to select a program number.

A check is made to ensure that a valid integer is selected, the user being warned and the prompt repeated in case of error.

If more than 23 programs exist on the tape, a prompt to press RETURN for the next page is added.

The user may select a program from that page, or press RETURN for the next page.

If the program selected is a program or text file, DIRECTORY finds, loads and executes it. If the user has selected a data file, DIRECTORY will prompt him for reselection.

No protection is provided to prevent an attempt to load an ASCII text data file.

As noted earlier, data files or non-directory controlled program files (to be appended or linked to other files) may be recorded in file 2, adding 1000 to the file number. DIRECTORY ignores such files.

Listing file 2:

```
100 ON EOF (Ø) THEN 200
110 FIND 2
120 INP@33:S,P$,D$
130 PRI S;P$;D$
140 GOTO 120
200 OFF EOF(Ø)
210 END
```

or using the EDITOR ROM provides a rapid 'TLIST' of the tape.

TITLE

TAPE DIRECTORY

Listing of File 2

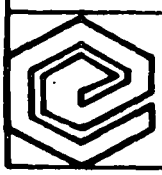
```
:14 APPLICATIONS LIB.
:3
:ASCII P
:ASCII program file.
:4
:ASCII T
:ASCII text file
```

TITLE

TAPE DIRECTORY

PROGRAM DIRECTORY TAPE:-`14' ` APPLICATIONS LIB.'

PROGRAM	NUMBER	PROGRAM DESCRIPTION
ASCII P	1	ASCII program file.
ASCII T	2	ASCII text file



Select your program number =

Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	
List Program's Variables	
ORIGINAL DATE: October, 1976 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 8K
AUTHOR: Brian Diehm, Tektronix, Inc., Wilsonville, Or	PERIPHERALS:

ABSTRACT

Files: 1 ASCII Program

Statements: 105

This program reads a tape file containing a BASIC ASCII program and prints an alphabetized list of all the variables used in that program.

The program first asks the user which tape file contains the program to be analyzed. Then, after reading the file, two alphabetized lists of variables are printed on the screen. The first list gives all of the numeric variables' names, the second list gives all the string variables' names. Listing of the files as they are being processed is optional.

Provision is made to allow processing of several files, combining the results into one list. The files do not have to be sequential, but input must be made for each one as they are processed.

TITLE

LIST PROGRAM'S VARIABLES

HARDWARE REQUIREMENTS

Any size 4050 will support this program. The built-in tape drive of the 4050 is utilized.

PROGRAM LIMITATIONS

The tape file that is analyzed with this program must be valid BASIC program code. It should have been stored on the tape by utilizing the SAVE statement. Any size program may be analyzed.

METHODOLOGY

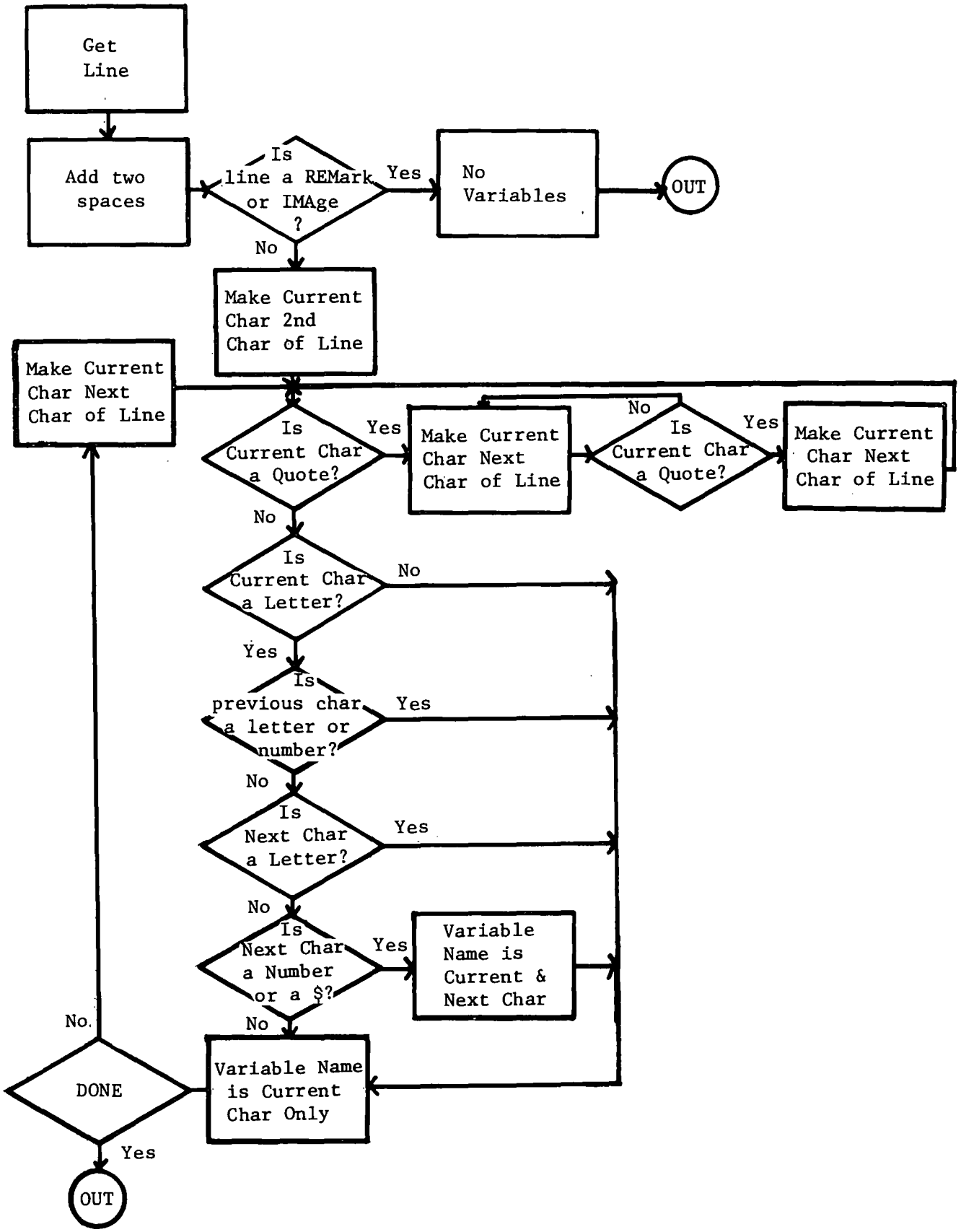
Variable names are considered two-character entities, and single-letter names are stored with a trailing space. Since there are a possible 286 different numeric-variable names, a string containing all character names would need to be 572 characters long. Similarly, there are a possible 26 string-variable names, requiring 52 characters to store. Numeric-variable names and string-variable names are stored separately.

The input file is read line by line and searched for variable names. When a variable name is found, it is compared with the names already found, to eliminate duplication. If the name is not already stored, it is inserted in its proper alphabetical location. When the file is completely read, the names are printed.

The flow chart on the next page describes the method used to search program lines for variable names.

TITLE

LIST PROGRAM'S VARIABLES



TITLE

LIST PROGRAM'S VARIABLES

OPERATING HINTS

If you wish to list the file as it is being processed, insert the following line into the code:

```
295 PRINT L$
```

Statements 390 and 490 contain 16 spaces between the quotes.

OPERATING INSTRUCTIONS

When first run, the program will print the following message:

```
SORT AND LIST PROGRAM VARIABLES
Enter File Number:  ?
```

Enter the number of the file on which the program you wish to analyze is stored. Follow with a RETURN.

Next the program asks:

```
Is this a continuation of a previous run?:
```

Type Y or N, followed by RETURN. If you respond N, the program's list of already found variable names is cleared before proceeding. If Y is the response, it is assumed that the results of this run should be combined with the results of the previous run of the program. If the program has not been run since loading, you must respond N.

TITLE

LIST PROGRAM'S VARIABLES

VARIABLE MAP

VARIABLE	TAPE FILE	USAGE
A		Contains ASCII equivalent of A$ during line examination
B		Contains ASCII equivalent of B$ during line examination
F		Contains file number at first, later is switch for quote mode
I		Work and index variable
J		Work and index variable
N		Number of numeric-variable names
S		Number of string-variable names
A$(3)		Work string
B$(2)		Work string
L$(74)		String to hold line of input from file
N$(572)		String to hold numeric-variable names
S$(52)		String to hold string-variable names

Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	
Cross-Reference & List Program Variables	
ORIGINAL DATE: November, 1976 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 8K
AUTHOR: Dan Taylor — Tektronix, Inc. Wilsonville, Or	PERIPHERALS: Optional - 4641 Printer

ABSTRACT

Files: 1 ASCII Program

Statements: 192

This program reads a BASIC ASCII program from tape and produces an alphabetized table of the variables used in the program. It also produces a cross-reference for each variable used which shows the BASIC line numbers where that variable is used and indicates if a value is assigned to that variable in that line of code. The BASIC program may be stored on multiple sequential tape files.

Three variables must be changed to output to the Printer.

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

Program Limitations

The tape files (s) to be analyzed must be valid 405 BASIC programs. The programs should have been stored on tape with the SAVE command.

An asterisk is normally printed with the line number when a value is assigned to the variable in that line of code. (eg X=2.7 INPUT X, READ X, etc.) But an asterisk is not printed if a value is assigned to an element of an array (eg X(2) = 4.6).

Your program may be too large for all the variables to be listed at once. If this happens the following message will be printed:

STORAGE FILLED IN THE MIDDLE OF FILE # 1. THE PROGRAM WILL CONTINUE WITH FILE # 1 AFTER THE FOLLOWING INFORMATION IS PRINTED:

VARIABLES:

A A$
B$
C$

' etc. '
' '
' '

The larger size memory you have the slower storage will be filled (see Methodology).

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

Methodology

The input files are read line by line and searched for variable names. When a variable is found its name, line number, and file number are saved at the end of the string N$. An asterisk is also put in N$ if a value is assigned to the variable in this line. A total of 15 characters are required in N$ for each occurrence of every variable.

If N$ fills up before all your files are analyzed then the information in it is immediately printed. The cross-referencing then continues. In this case your cross-reference would be in two parts.

How soon N$ fills depends on the machine size you have:

16K - 390 variable occurrences

24K - 930 variable occurrences

32K - 1480 variable occurrences

The information stored in N$ is organized as a collection of linked lists. There is one list for each variable you use. The array A (A(26,12)) consists of pointers into N$. For example, A (2,2) might be 6181.0625. This means that:

1 BØ is used

2 4096 (=.0625 x 2 ↑ 16) is the start in N$ of the linked list for BØ.

3 6181 is the end in N$ of the linked list for BØ.

A (j,k) = Ø means that that variable is not used at all.

A (1,1) = A
1,2 = AØ
.
.
.
1,1Ø = A9
1,12 = A$

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

Operating Hints

If you wish to use this program with a printer you must change 3 lines of code:

FROM	TO
1030 F4 = 6	1030 F4 = 10
1040 F5 = 1.0E + 300	1040 F5 = 60
1050 J5 = 32	1060 J5 = 41

F4 = the number of items to print per line (each require 12 spaces)

FS = the number of lines to print before doing a form feed (control L)

JS = device address of where the cross-reference information will be printed (32 is for the screen)

Line 1010 is a REM which also tells you about these changes.

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

Operating Instructions

After the program is in the machine type RUN:

CROSS-REFERENCE AND LIST PROGRAM VARIABLES

NOTE: YOUR PROGRAM MAY BE ON SEVERAL TAPE FILES.

Begin with File Number = 29

End with File Number = 30

FILE # 29

FILE # 30

When the last file has been processed*(30 in this example) the program prints a table which gives you an overview of which variables were used:

* or when N$ is filled - see Limitations

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

VARIABLES:

AØ					A6			A9	
									B$
DØ			D3						D$
EØ	E1								
	F1	F2							
HØ	H1	H2							H$
	L1							L9	
MØ									
			N3						
	P1								P$
RØ	R1		R3	R4				R9	
SØ		S2	S3	S4	S5	S6	S7	S9	S$
TØ									T$
	U1	U2	U3	U4		U6	U7	U8	U9
	V1	V2	V3	V4	V5	V6	V7	V8	
	W1	W2	W3	W4	W5	W6	W7		
									Y$

The blank spaces in the table are the variables which are not used at all in your program. (If you want the table sent to a printer, see Operating Hints).

Press the PAGE key and the actual cross-reference will be printed:

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

VARIABLES:

```
AØ
 3Ø  4Ø5Ø

A6
 3Ø  422Ø    3Ø  422Ø    3Ø  422Ø    3Ø  422Ø    3Ø  426Ø *  3Ø  427Ø

A9
 29  4Ø1Ø    3Ø  4Ø1Ø    3Ø  428Ø *  3Ø  428Ø    3Ø  43ØØ *  3Ø  43ØØ
 3Ø  432Ø *  3Ø  432Ø

B$
 29  484Ø *  29  487Ø *  29  499Ø    29  511Ø    29  514Ø    29  635Ø *
 29  636Ø

DØ
 29  592Ø    29  594Ø    29  6Ø7Ø *

D3
 29  416Ø *

D$
 3Ø  414Ø

EØ
 29  438Ø *

E1
 3Ø  423Ø *

F1
 29  4Ø7Ø *  29  4Ø8Ø    29  4Ø9Ø    29  41ØØ    29  411Ø    29  415Ø
 29  432Ø    29  433Ø    29  447Ø    29  461Ø    29  462Ø    29  471Ø
```

The number 29 and 30 are the tape file numbers in which this line of code appears. For example:

AØ
3Ø 4Ø5Ø means aØ is used in line 4050 of file 30

D3
29 4160 * means D3 is assigned a value (indicated by the asterisk) in line 4160 of file 29.

If your program is on only one tape you may not want the file number printed. This can be done by changing the 1360 REM.... to 1360 F$ = " ".

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

Variables

A (26,12) - an array of "pointers" into N$ - see Methodology

A$ (1) - the character previous to the one being analyzed

B$ (1) - the character being analyzed

C$ (1) - the character following the one in B$

F - the number of the tape file currently being read

FØ = MEMORY of your machine - 250. N$ is considered to be full when it has FØ characters in it.

F1 - beginning file mumber

F2 - ending file number

F3 - scratch used when printing the cross-reference map

F4 - the number of items to print per line (each require 12 spaces)

F5 - the number of lines to print before doing a form feed (control L) - used only if you are using a printer.

F6 = F5 - 2

F7 - scratch used when printing the cross-reference map

F$ (3) - the current tape file number

6$ (5) - the line number of the current line of code

H$ (8) = F$ & 6$

I$ (12) = " Ø 1 2 3 4 5 6 7 8 9 $ "

J - character position in L$ of A$
JØ - character position in L$ of B$
J1 - character position in L$ of C$

J2,J3,J4 - scratch variables

J5 - device address of where the cross-reference information will be printed. See Operating Hints

TITLE

CROSS-REFERENCE & LIST PROGRAM VARIABLES

J6 = 2 ↑ 16, used for unpacking the pointers from A

J7 = 1 + 1/J6, used for packing the pointers into A

J8 - position in L$ of the second character after the first space.

J9 = 15, the total number of characters required in N$ for each occurence of every variable

J$ (7) - Scratch

K$ (3) - the first 3 characters of a line of code which follow the first space. (eg. K$ might be REM or PRI, etc.)

L$ (75) - string to hold a line of input from the file

N - the number of characters currently in N$

N$ - holds the information about each occurrence of every variable. DIM N$ (FØ + 2 * J9), FØ + MEM - 250

R$ (6) - scratch

S$ (10) - 10 spaces, used for padding other strings with spaces

T$ (J9) - holds the information which is to be appended at the end of N$

V$ (2) - the name of a variable (eg A1)

W$ (3) - scratch

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
Device Address Adding Program		
ORIGINAL DATE	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED
August, 1979		16K
AUTHOR		PERIPHERALS
Jan Broenink	Tektronix Marketing Centre Amstelveen, Holland	4924 Digital Tape Drive

ABSTRACT

Optional - 4641/4642 Printer

Files: 1 ASCII Program
Requires a data tape

Statements: 402

The program reads a tape file from the 4924 containing a 4050 BASIC program in ASCII format and updates the program by adding a device address (for graphics and alphanumerics) to output statements without a device address or with address 32 (without a secondary address).

The program searches for the following output statements without a device address or with address 32:

PRINT	RMOVE	MOVE	AXIS
LIST	RDRAW	DRAW	GIN

and will automatically or with user interaction add a device address. Interaction allows the user to define more than one output address within a program. For instance, user instructions may be directed to the screen while graphs may be directed to the plotter.

If APPEND, OLD and FIND's are used in a program, a message is given how many APPEND's, etc., have been traced. In some cases the user has to check the result if the new program is still usable in relation with other program(s) or routine(s).

A routine is added to the original program to define a device address for graphical and alphanumerical output. An unused User-Definable Key in the original program may be used to call this routine.

The original program may be stored on several sequential tape files.

TITLE

DEVICE ADDRESS ADDING PROGRAM

PROGRAM LIMITATIONS

The tape file(s) to be updated must be valid 4050 BASIC programs.

The original program(s) may be stored on several tape files, as long as the files are sequential on tape.

The programs should have been stored on tape in ASCII format (i.e. by use of the SAVE command).

METHODOLOGY

The program starts with a dialogue in order to assign values to the different processing parameters. After this definition phase the program starts the processing with the first specified file on the original tape and proceeds the processing file after file up to and including the last file. In this section the processing of a program file is being described.

1. Search is made to the file resided on the tape inserted in the 4924 Digital Cartridge Tape Drive.

   A check is made if the file is in a valid format using the header of the file.

2. When the specific file has been found the program searches for:

   - UDK's used by the program
   - Names of numerical variables used by the program
   - Last statement number
   - Statements containing APPEND, OLD and FIND

3. If APPEND, OLD and FIND's are used in a program a message is given how many APPEND's etc. have been traced. In some cases the user has to check the result if the new program is still useable in relation with other program(s) or routine(s). If the program traces an APPEND the routine for device address specification is added starting at statements # 65390, otherwise the routine is added starting at last statement # +10.

4. The user has to specify which variable name to be used for graphical and alpha-numerical device address.

   Default names are Z9 and Q9, but if these are already in use in the original program the user should submit new names.

TITLE

DEVICE ADDRESS ADDING PROGRAM

Variable	Used to Store	Type
N2	Number of names for numerical variables	Simple
N3	Scratch variable	Simple
N4	Scratch variable	Simple
N5	Scratch variable	Simple
N$	String containing names of numerical variables of original program	String (572)
O$	String used to display statements on output device for program messages (screen or printer) Statements will be displayed with suppressed control characters	String (216)
P$	String containing variable name for printer output	String
S1	Start statement number minus 10 of routine to define output device addresses in new program	Simple
S2	Statement number to be used in relation with the first free UDK	Simple
S3	Number of first free UDK	Simple
S4	Flag to signal APPEND statements = 0 NONE > 0 # of APPEND's	Simple
S5	Flag to signal FIND statements = 0 NONE > 0 # of FIND's	Simple
S6	Flag to signal OLD statements = 0 NONE > 0 # of OLD's	Simple
S$	Scratch string	String
T$	Scratch string	String
V$	Scratch string	String (3)
W$	Scratch String	String (2)
X$	Scratch string	String
Y$	Scratch string	String (74)
Z$	Scratch string	String

TITLE

DEVICE ADDRESS ADDING PROGRAM

PROGRAM VARIABLES

Variable	Used to Store	Type
A	ERROR-CODE FOR MAG-TAPE (see error codes in 4924 manual)	Simple
A$	String for reading statements from original program	String
B9	Output-device-address for messages from program	Simple
B$	String to trace I/O statements	String
C$	String containing @... (=output-dev. address) to be inserted in original statement	String
D	Device-address of 4924 tape-unit	Simple
E	File number of last file to be processed on original tape	Simple
F	SRQ-flag	Simple
F1	Flag for message on how to define device-address- variables in new program	Simple
F2	Flag for adding statements 1 and 2	Simple
F3	Flag for interaction on PRINT/LIST statements	Simple
F5	Flag for editting too long statments	Simple
F$	String to trace free and occupied UDK's	String (20)
G$	String containing variable name for graphic output	String
I	Counter	Simple
J	Counter	Simple
M	Current statement number	Simple
N	Current file number of original tape	Simple
NØ	Number of first file of original tape	Simple
N1	Number of first file on output tape (i.e. file number NØ of original will be transferred to file number N1 on new tape)	Simple

TITLE

DEVICE ADDRESS ADDING PROGRAM

When the program is finished with all the specified files it will stop after the message:

END OF TRANSMISSION

In case an error occurs on the output tape the following message will be displayed and the program will stop its execution.

MAGNETIC TAPE-ERROR. CODE: 6. FILE 1 (NEW 1)

TITLE

DEVICE ADDRESS ADDING PROGRAM

When the option of user interaction is chosen the statement will be displayed on the screen (with suppressed contorl characters) and the user must respond to the question:

```
** STATEMENT # 62
62 PRINT "LGLINE LABELS MUST BE REPOSITIONED IN NEXT PLOTJ"
  WHICH DEVICE (Graphics/Alpha-num./Screen) ?
```

The user must respond by: G, A or S, otherwise the question reappears until a satisfactory answer has been given.

When @32 is contained in a statement combined with a secondary address the following message is given:

```
NEW FILE 1: LINE 300 CONTAINS @32 WITH SECONDARY ADDRESS.
THE STATEMENTS IS :
PRINT @32,20:A$
THIS IS NOT CHANGED IN THE NEW PROGRAM.
```

When a statement which needs to be updated is too long (more than 68 characters) the user can change the line if desired.

```
NEW FILE 1: LINE 6370 IS TOO LONG!
  THE STATEMENT IS :
6370 PRINT "LTHE THREE CHART TITLE LINES ARE:_1 ";T$;"_2 ";U$;"_3 ";V$
DO YOU WANT TO CHANGE THE STATEMENT (YES/NO)
```

A YES response allows you to input a new statement.

A NO response copies the original statement without making any changes.

TITLE

DEVICE ADDRESS ADDING PROGRAM

The program searches for a free User-Definable Key to be used for the selection of output devices by specifying the address.

When statement numbers 1 and 2 are not in use and the first executable statement is not INIT the program will add 1 and 2 as follows:

```
1 GOSUB----
2 GOTO 100
```

where statement # at the GOSUB is the first line of a small routine (which is added at the end of the current program).

This routine will set both variables for the output devices addresses to 32 (=screen). Otherwise the following message will be given:

```
NEW FILE 1 :
  BEFORE RUNNING THE NEW PROGRAM ONE HAS TO DEFINE THE
  THE OUTPUT-DEVICE-ADDRESSES.
  THIS CAN BE DONE IN TWO WAYS:
  (1) TYPE RUN 65490 FOLLOWED BY [CR]
      ALL OUTPUT ON THE SCREEN (ADDRESS 32).
  (2) TYPE RUN 65480FOLLOWED BY [CR]
      TO SELECT OUTPUT-DEVICES FOR GRAPHICS
      AND ALPHA-NUMERICAL OUTPUT.
```

When a free UDK is available the following is added to this message:

```
  (3) USE UDK # 19
      TO DEFINE THE DEVICE-ADDRESSES.
```

The program proceeds by checking and updating (when needed) statement by statement.

When a statement contains:

@32,PRINT,DRAW,MOVE,RDRAW,RMOVE,AXIS,LIST or GIN

an update is carried out by inserting:

@ variable name for appropriate device

after the first 3 letter of the BASIC keyword and ignoring the rest of the keyword.

TITLE

DEVICE ADDRESS ADDING PROGRAM

ADDRESS OF OUTPUT-DEVICE FOR THIS PROGRAM : 32

Specify the output address for messages and other relevant information.
- 32 - for the screen
- ## - for the printer (address of the printer)

The program checks if the file is in the proper format (ASCII)
A message will be given for each processed file.

****

TRANSFER OF FILE : 1 TO FILE : 1

****

***** WARNING *****
FILE # 1 CONTAINS :
1 APPEND`s
1 FIND`s
***** CHECK THE RESULT *****

WHICH VARIABLE DO YOU WANT TO USE AS ADDRESS FOR GRAPHIC-OUTPUT
DEFAULT : Q9 YOUR VARIABLE :

If Q9 is not in use in the program press RETURN, if it is in use enter another variable.
A check is made to see if the variable is correct (simple variable unsubscripted), otherwise the question will reappear until a satisfactory response is given.

WHICH VARIABLE DO YOU WANT TO USE AS ADDRESS FOR PRINTER-OUTPUT
DEFAULT : Z9 YOUR VARIABLE :

If Z9 is not in use in the program press RETURN, if it is in use enter another variable.
A check is made to see if the variable is correct (simple variable unsubscripted), otherwise the question will reappear until a satisfactory response is given.

TITLE

DEVICE ADDRESS ADDING PROGRAM

LAST FILE OF ORIGINAL : 1

Specify the last file on the original tape to be updated. The processing of the program files is done from first up to and including last file numbers. A check is made if the specified number is in the range of 1 up to 255, otherwise the question will reappear until a satisfactory response is given.

FILE #1 OF ORIGINAL TO BE COPIED TO FILE #1 OF NEW TAPE.

Specify the first file on the output tape to be used. The transfer of the first file of the original tape is done to this file. A check is made if the specified number is in the range of 1 up to 255, otherwise the question will re-appear until a satisfactory response is given.

AUTOMATIC MARKING (YES/NO) : YES

Specify whether or not to mark the files on the output tape.
- YES - creates files on the output tape of a size over the original size, since additional statements are added (768 bytes more for a new file)
- NO - expects pre-MARked files on the output tape.

Only YES or NO responses may be made, otherwise the question will reappear until a satisfactory response is given.

INTERACTION PRINT & LIST-STATEMENTS (YES/NO) :YES

Specify whether or not to interact on PRINT and LIST statements.
- YES - the program updates automatically all statements like MOVE, DRAW, etc. for graphic output device and the statements like PRINT and LIST are updated by user interaction. The user selects how these statements are to be updated: graphic device, alpha-numeric device, or screen.
- NO - the program updates automatically all statements with PRINT and LIST for alpha-numerical output device and statements like MOVE, DRAW, etc. with graphic output device.

TITLE

DEVICE ADDRESS ADDING PROGRAM

OPERATING INSTRUCTIONS

Press AUTOLOAD if the program is the first file on tape, otherwise:

FIND #
OLD
RUN

Respond to the questions accordingly.

***** ADD OUTPUT-DEVICE ADDRESSES (incl. selection) *****

GRAPHICS TO PLOTTER
ALPHA-NUMERICS TO PRINTER

SET ORIGINAL TAPE ON SAFE AND INSERT IT IN THE 4924
TYPE [DONE] AND PRESS [RETURN] WHEN READY :DONE

To avoid the destruction of files on an original tape the user is warned to put the tape on SAFE.

WHAT IS THE GPIB-ADDRESS OF THE 4924 : 4

Specify the address of the 4924 (see switch setting on back)

SET NEW TAPE ON UNSAFE AND INSERT IT IN THE 4051
TYPE [DONE] AND PRESS [RETURN] WHEN READY :DONE

To SAVE the updated program the user is instructed to put the new tape on UNSAFE.

FIRST FILE OF ORIGINAL : 1

Specify the first file on the original tape to be updated. A check is made if the specified number is in the range of 1 up to 255 otherwise the question will reappear until a satisfactory response is given.

TITLE

DEVICE ADDRESS ADDING PROGRAM

When the names are given the program checks if they are not in use in the original program and checks if the given variable name is unsubscripted.

5. If a free UDK is available the program will add a new function for use with this key. The usage of this key in the new program allows the user to specify the addresses for:

   - Graphical output
   - Alphanumeric output

6. When specified the user can interact with the program during its execution in order to add for PRINT and LIST statements the appropriate address. This is very useful since some of the PRINT statements are used in an average program for axis labeling or adding text to graphics.

   Other PRINT (and LIST) statements are used for user dialogue and should give the output on the screen.

7. When the program finalizes the program update with the last file the user does have direct access to the new programs.

   They simpley reside on the files on the new tape as specified by the user.

Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
Log/Linear Axis Labeling Routine		
ORIGINAL DATE	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED
November, 1976		16K
AUTHOR		PERIPHERALS
Steven Den Beste	Tektronix, Inc. Wilsonville, Or.	

ABSTRACT

Files: 1 Program

Statements: 241

This program is a subroutine designed to be used with a user program. The subroutine generates an L-shaped axis with logarithmic or linear labeling on either axis, covering any range of positive values, and placed anywhere on the screen.

All labels are 4 characters, including a decimal point and a sign (if negative).

A pair of transformation functions are defined by the user before generating a plot.

TITLE

LOG/LINEAR AXIS LABELING ROUTINE

MAN/MACHINE INTERFACE

This routine is configured as a subroutine. 11 specific variables are preset by the user, the subroutine is called, and returns after generating the axis.

The preset variables are:

V1	–	MIN	X	VIEWPORT	(in GDU's)
V2	–	MAX	X	VIEWPORT	
V3	–	MIN	Y	VIEWPORT	
V4	–	MAX	Y	VIEWPORT	

(These specify exactly where on the screen the plot lies.)

W1	–	MIN	X	WINDOW	(in user data units)
W2	–	MAX	X	WINDOW	
W3	–	MIN	Y	WINDOW	
W4	–	MAX	Y	WINDOW	

(These specify the range of the plot.)

T1 – MINIMUM NUMBER OF TICS IN X DIRECTION (IF LINEAR)
T2 – MINIMUM NUMBER OF TICS IN Y DIRECTION (IF LINEAR)

T0 – SELECT VARIABLE
IF 0 THEN LINEAR X, LINEAR Y
1 THEN LOG X, LINEAR Y
2 THEN LINEAR X, LOG Y
3 THEN LOG X, LOG Y

METHODOLOGY

This subroutine will steal some of the allotted viewport for the axis and labels. The coordinates of the actual plottable area within the axis is returned in V1, V2, V3, and V4.

The triplets W1, W2, T1 and W3, W4, T2 are processed separately. If a given direction is processed as a linear plot, the low and high limits and number of tics are adjusted so that tic intervals are 1*10↑N, 2*10↑N, 2.5*10↑N or 5*10↑N. The modified low and high window are passed back. If a given direction is processed as a logarithmic plot, the min window value is dropped to the next lower power of 10, and the max window value to the next higher power of 10. In this case, the log of the min window value and the log of the max window value are passed back.

PROGRAM LIMITATIONS

All labels are 4 characters, including a sign (if negative) and a decimal point. For linear labels, if the significant digits of the label are not within those 4 characters, the label will be the same.

Obviously, a logarithmic plot must involve only positive values. A negative number in a logarithmic plot will halt the program with an error.

TITLE

LOG/LINEAR AXIS LABELING ROUTINE

OPERATING HINTS

The functions FNX and FNY are set by the routine to be transformation functions. When generating the actual plot, use them in the following way:

For all moves:

MOVE FNX (expression for X), FNY (expression for Y)

For all draws:

DRAW FNX (expression for X), FNY (expression for Y)

If the WINDOW or VIEWPORT must be changed, they can be restored by:

VIEWPORT V1, V2, V3, V4
WINDOW W1, W2, W3, W4

If the above is done, and plot values in a given direction are all positive, the plot type in that direction (specified by the parameter T0) can be changed without otherwise modifying your program.

One consequence of this is that the window values will be passed back if the plot is linear, but the LOG of the window values will be passed back if the plot is logarithmic. The input window values must be the same whether the plot is linear or logarithmic.

TITLE

LOG/LINEAR AXIS LABELING ROUTINE

VARIABLE MAP

VARIABLE	TAPE FILE	USAGE
T3 (1)		Tic increment in X direction
T3 (2)		Tic increment in Y direction
T3 (5)		Factor in X direction
T3 (7)		Factor in Y direction
T3 (10)		Factor passed back from sub at 6330
T3 (12)		Range of plot passed to sub at 6330
T3 (13)		Number of tics passed to sub at 6330
T3 (14)		Value of increment passed back from sub at 6330
T3 (15)		Min window passed to sub at 6330
T3 (16)		Max window passed to sub at 6330
T3 (20)		Tics per label passed back from sub at 6620
T3 (23)		Value of increment passed to sub at 6620
T3 (26)		Window value of lowest X label
T3 (27)		Window value of highest X label
T3 (28)		Increment of labels in X
T3 (29)		Window value of lowest Y label
T3 (30)		Window value of highest Y label
T3 (31)		Increment of labels in Y

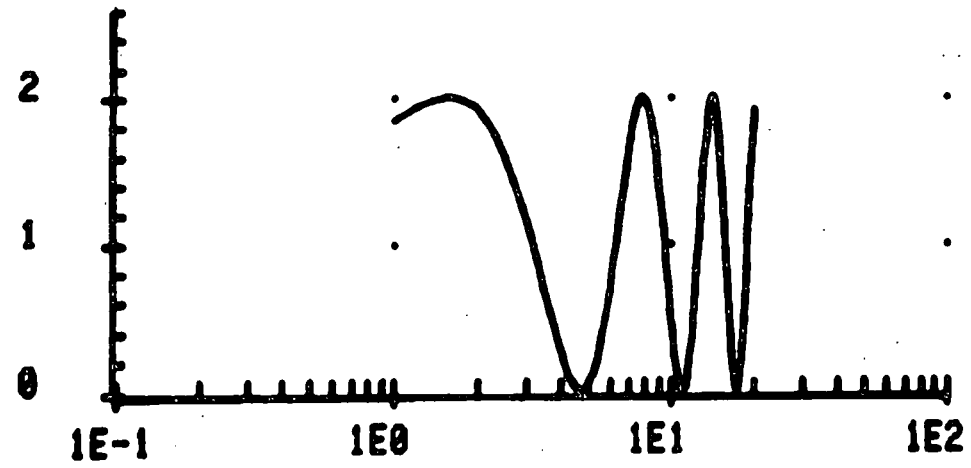
TITLE

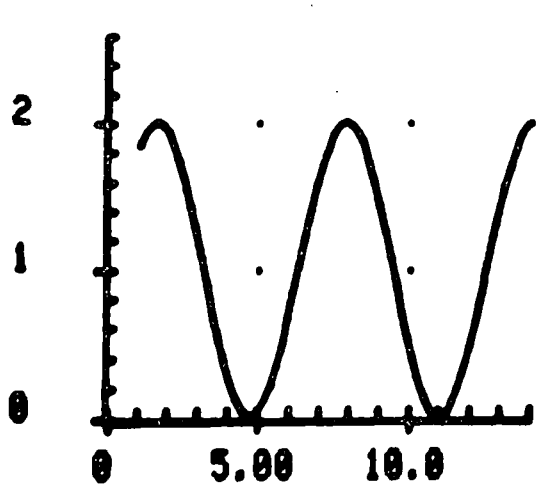
LOG/LINEAR AXIS LABELING ROUTINE

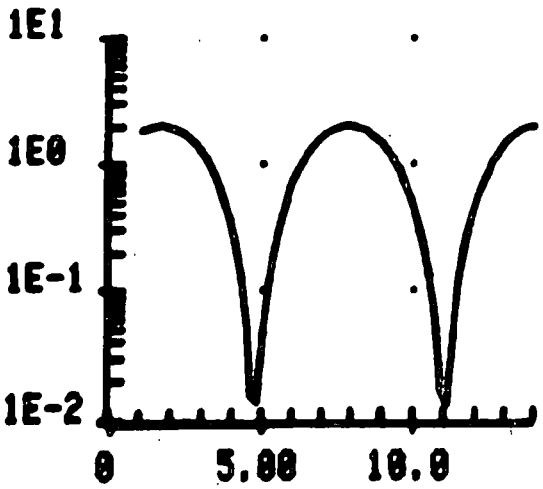
```
100 DEF FNG(Q)=Q*Q*(1-SIN(Q/1.5))
110 REM THE FUNCTION TO PLOT
120 DATA 0,60,0,47,65,130,0,47,0,60,52,100,65,130,52,100
130 REM DATA FOR THE FOUR VIEWPORTS
140 RESTORE
150 T1=14
160 T2=14
170 REM THE NUMBER OF TICS FOR LINEAR AXIS
180 PAGE
190 FOR T0=0 TO 3
200 REM CHANGE THE SELECT VARIABLE
210 REM *** PLEASE NOTE THAT WITHIN THIS LOOP, NOTHING CHANGES
220 REM    BUT THE VIEWPORT AND THE SELECT VARIABLE
230 W1=0
240 W2=100
250 W3=0
260 W4=20000
270 REM SET WINDOW
280 READ V1,V2,V3,V4
290 REM THE NEW VIEWPORT
300 PRINT @32,21:V1+15,V3
310 PRINT "T0=";T0
320 REM PRINT THE SELECT VARIABLE FOR DOCUMENTATION
330 GOSUB 5000
340 REM GENERATE THE AXIS
350 MOVE FNX(1),FNY(FNG(1))
360 FOR I=1 TO 100 STEP 0.5
370 DRAW FNX(I),FNY(FNG(I))
380 NEXT I
390 REM GENERATE THE PLOT
400 NEXT T0
410 HOME
420 END
```

TITLE

LOG/LINEAR AXIS LABELING ROUTINE

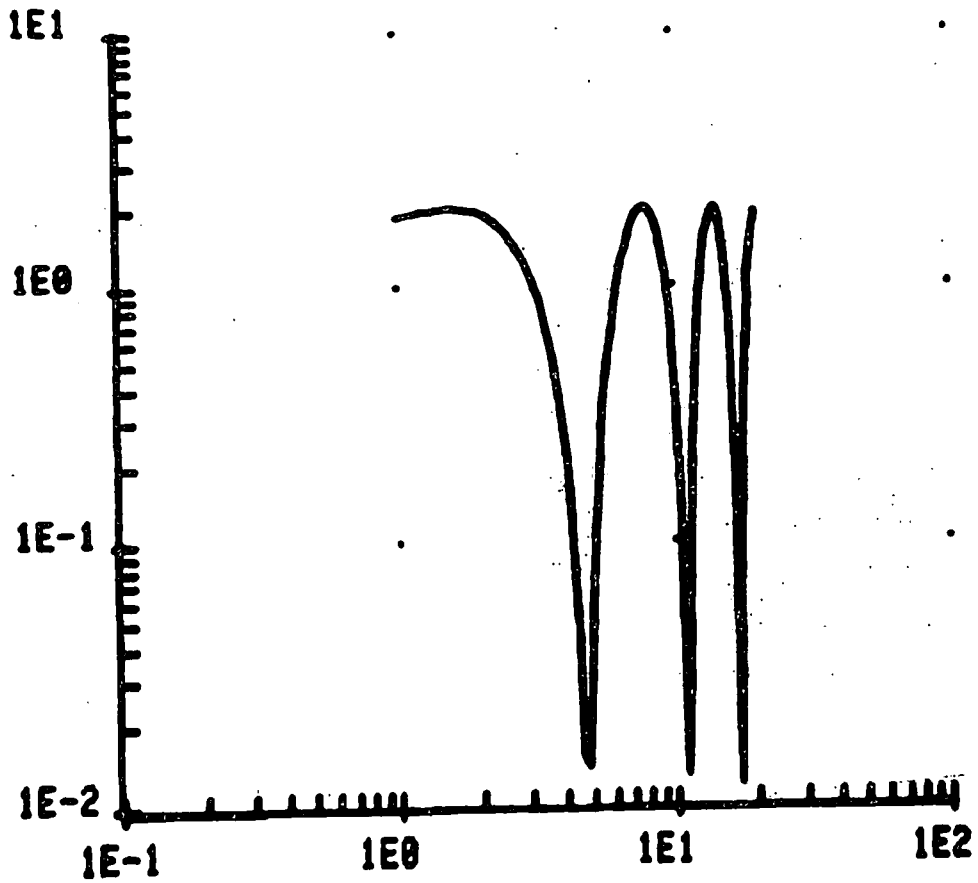






TITLE

LOG/LINEAR AXIS LABELING ROUTINE



Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
Dashed Lines		
ORIGINAL DATE	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED
Spring, 1976	Spring, 1978	8K
AUTHOR	Tektronix, Inc.	PERIPHERALS
Bob Ross	Wilsonville, OR	Optional - 4662/4663 Plotter

ABSTRACT

Files: 1 Program

Statements: 154

Three subroutines draw dashed lines for:

1) a Y array with X values stepped linearly from a starting to an ending value;

2) points stored in X and Y arrays;

3) a sequence of X and Y values.

The dashes are a constant length regardless of the viewport and window chosen. The dash length and ratio of dash to dash plus space are selectable. The line can start and end on a full dash or full space.

TITLE

DASHED LINES

OPERATING INSTRUCTIONS

Subroutine 1 - Dashed Line for Stepped X and Array Y

The user provides the following information in the mainline program:

X1 The starting value of X

X2 The ending value of X

X3 The X step length

YØ The Y array (one-dimensional)

N1 The dash length in terms of the X-axis units if N1>Ø. If N1<Ø, then the magnitude of N1 gives the dash length in terms of the Y-axis units. The actual dash length may be modified slightly to satisfy the end point constraints if N3≠Ø.

N2 The ratio of dash length to dash plus space length. (Ø<N2≤1)

N3 Flag to specify the starting-ending, dash-space conditions of the line drawn.

N3	Start	End
2	dash	space
1	dash	dash
0	dash	depends on dash length
-1	space	space
-2	space	dash

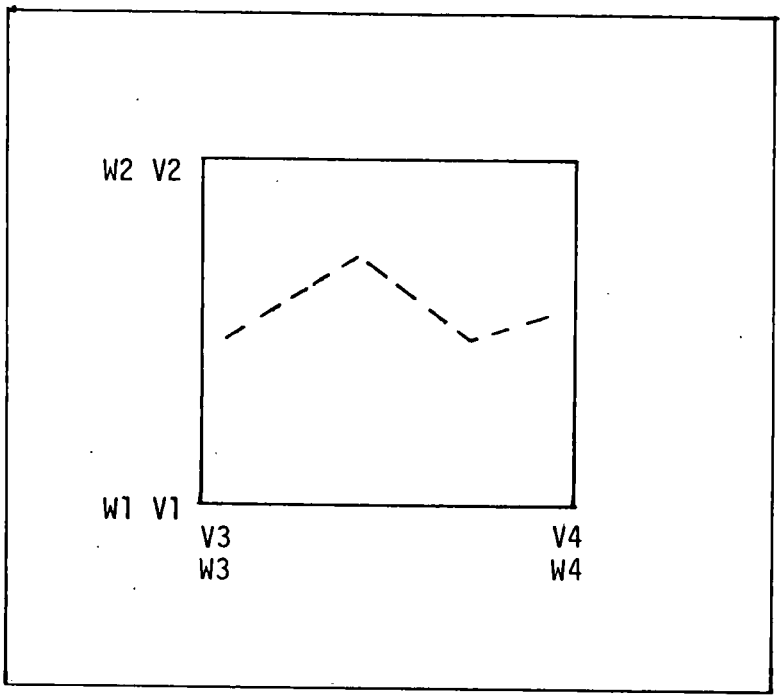
N9 The display address for the plotting device used. For the 4051 screen, N9=32.

VØ The ratio of Y viewport length to X viewport length.

WØ The ratio of Y window length to X window length.

TITLE

DASHED LINES



$$V\emptyset = \frac{V2-V1}{V4-V3}$$

$$W\emptyset = \frac{W2=W1}{W4-W3}$$

To draw the line, do GOSUB 116Ø.

Operating Hints

At least one dash is drawn even if the dash length specified exceeds the total length of the line.

If some of the options in this subroutine are not required, then several lines of code may be simplified or eliminated. For example, if N3=0, then lines 1220-1280 may be deleted.

TITLE

DASHED LINES

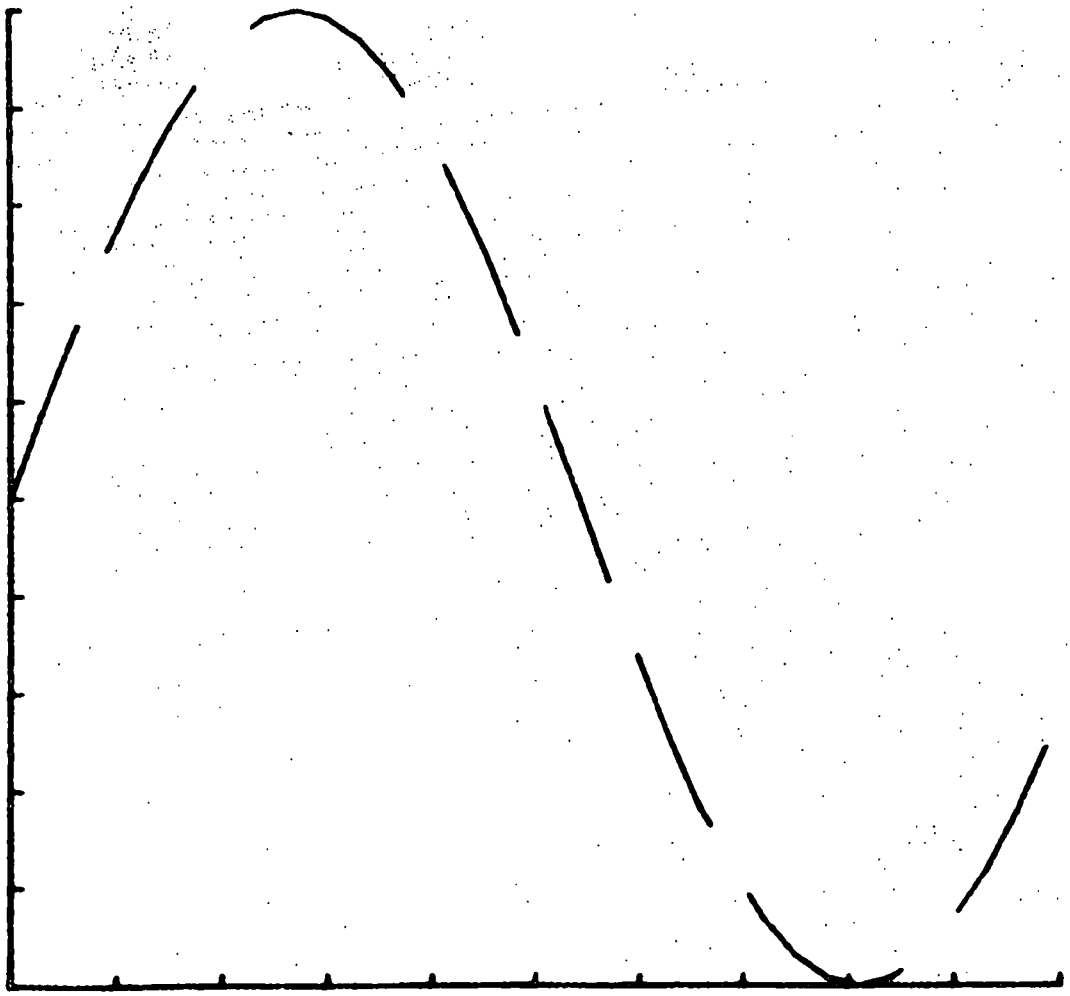
VARIABLE MAP

VARIABLE	TAPE FILE	USAGE
N1		Dash length
N2		Ratio of dash length to dash plus space length
N3		Flag to specify start-end, dash-space constraints
N4-N8		Working variables
N9		Display address (N9=32 for 4051)
VØ		Ratio of Y to X viewport length
WØ		Ratio of Y to X window length
X1		Starting X
X2		Ending X
X3		X step
X5-X7		Working variables
YØ		Array of Y values
Y5,Y6		Working variables

TITLE

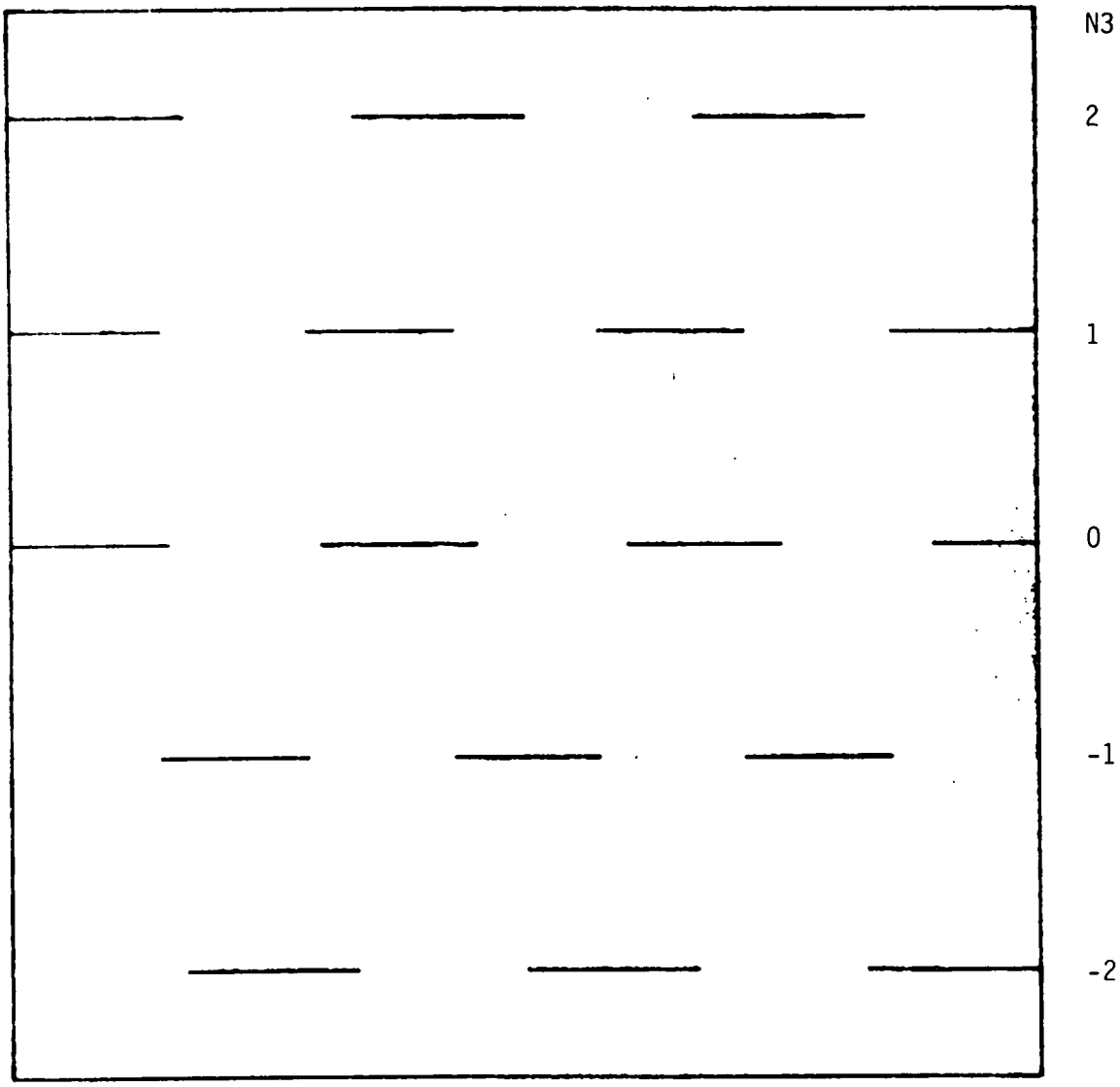
DASHED LINES

```
100 DIM X0(50),Y0(50)
110 N0=37
115 N1=2
120 N2=0.7
130 N3=1
160 N9=32
170 V0=1
180 W0=1
190 SET DEGREES
200 PAGE
210 VIEWPORT 10,110,0,100
220 WINDOW 0,10,0,10
222 X1=0
224 X2=10
226 X3=0.3
230 FOR I=1 TO 37
250 Y0(I)=SIN(10*(I-1))*5+5
260 NEXT I
270 AXIS 1,1
280 GOSUB 1160
300 HOME
390 END
```



TITLE

DASHED LINES



EXAMPLES OF N3 OPTIONS

TITLE

DASHED LINES

Subroutine 2 - Dashed Line for X and Y Arrays

The user provides the following information in the mainline program:

XØ The X array (one-dimensional).

YØ The Y array (one-dimensional).

NØ The number of elements in XØ and YØ which are to be plotted starting with XØ(1) and YØ(1) and ending with XØ(NØ) and YØ(NØ).

N1 The dash length in terms of the X-axis units if N1>Ø. If N1<Ø, then the magnitude of N1 gives the dash length in terms of the Y axis units. The actual dash length may be modified slightly to satisfy the end point constraints if N3≠Ø.

N2 The ratio of dash length to dash plus space length. (Ø<N2≤1)

N3 Flag to specify the starting-ending, dash-space conditions of the line drawn.

N3	Start	End
2	dash	space
1	dash	dash
0	dash	depends on dash length
-1	space	space
-2	space	dash

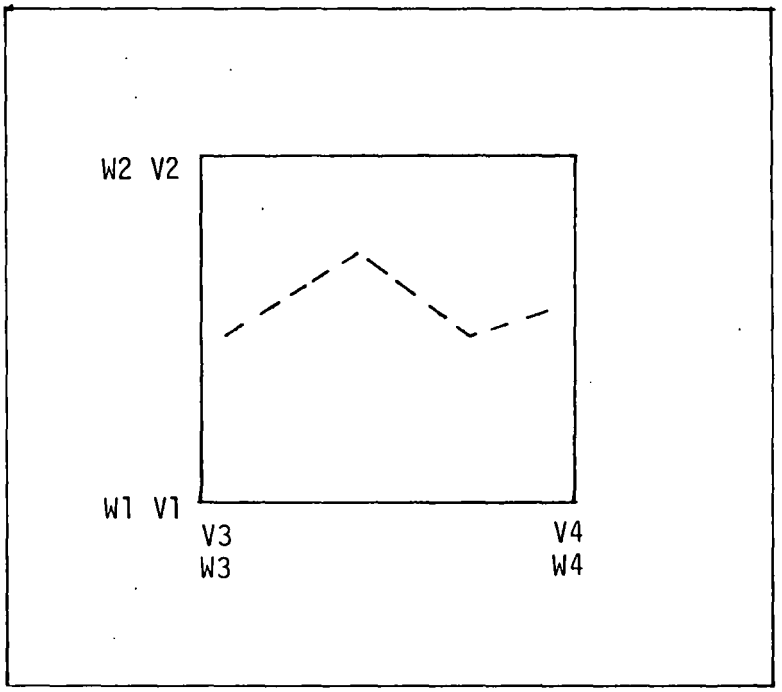
N9 The display address for the plotting device used. For the 4051 screen, N9=32.

VØ The ratio of Y viewport length to X viewport length.

WØ The ratio of Y window length to X window length.

TITLE

DASHED LINES



$$V\emptyset = \frac{V2-V1}{V4-V3}$$

$$W\emptyset = \frac{W2-W1}{W4-W3}$$

To draw the line, do GOSUB 2150

Operating Hints

At least one dash is drawn even if the dash length specified exceeds the total length of the line.

If some of the options in this subroutine are not required, then several lines of code may be simplified or eliminated. For example, if N3=0, then lines 2200-2250 may be deleted.

TITLE

DASHED LINES

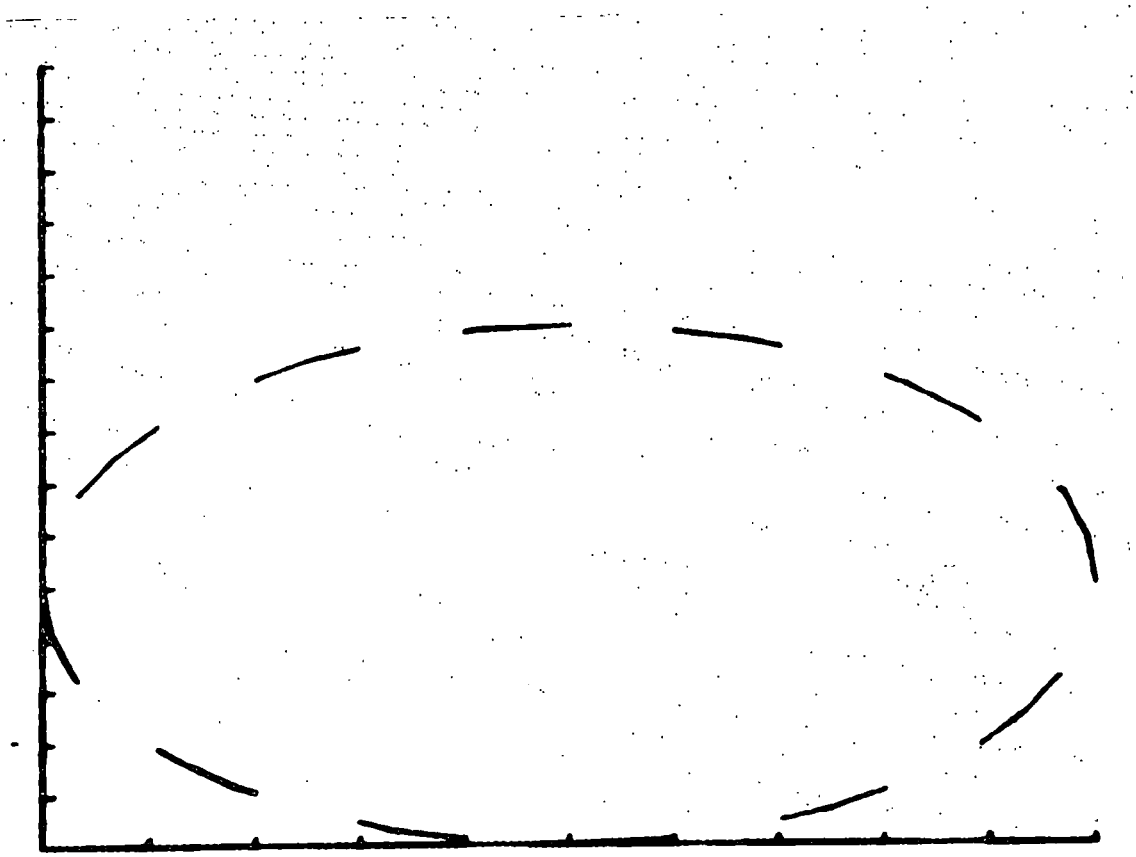
VARIABLE MAP

VARIABLE	TAPE FILE	USAGE
NØ		The number of points in XØ and YØ which are plotted
N1		Dash length
N2		Ratio of dash length to dash plus space length
N3		Flag to specify start-end, dash-space constraints
N4-N8		Working variables
N9		Display address (N9=32 for 4051)
VØ		Ratio of Y to X viewport length
WØ		Ratio of Y to X window length
XØ		Array of X values
X5,X6		Working variables
YØ		Array of Y values
Y5,Y6		Working variables

TITLE

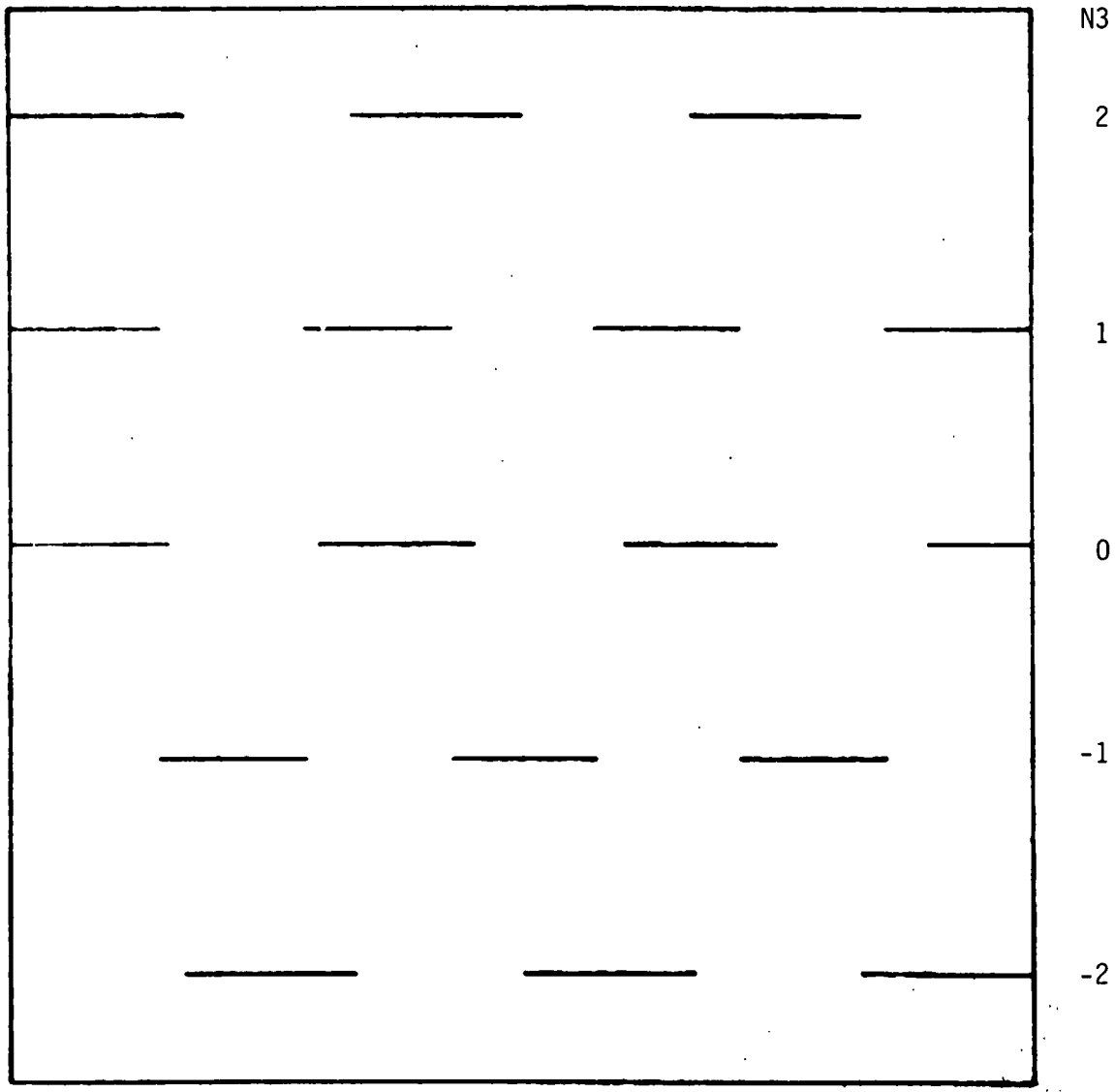
DASHED LINES

```
100 DIM X0(50),Y0(50)
110 N0=37
120 N1=1
130 N2=0.5
140 N3=2
150 N9=32
160 V0=0.8
170 W0=1.5
180 SET DEGREES
190 PAGE
200 VIEWPORT 10,110,0,80
210 WINDOW 0,10,0,15
220 FOR I=1 TO 37
230 X0(I)=COS(10*(I-1))*5+5
240 Y0(I)=SIN(10*(I-1))*5+5
250 NEXT I
260 AXIS 1,1
270 GOSUB 2150
280 HOME
290 END
```



TITLE

DASHED LINES



EXAMPLE OF N3 OPTIONS

TITLE

DASHED LINES

Subroutine 3 - Dashed Line for X and Y Entries

The user provides the following information in the mainline program:

X4 The X entry

Y4 The Y entry

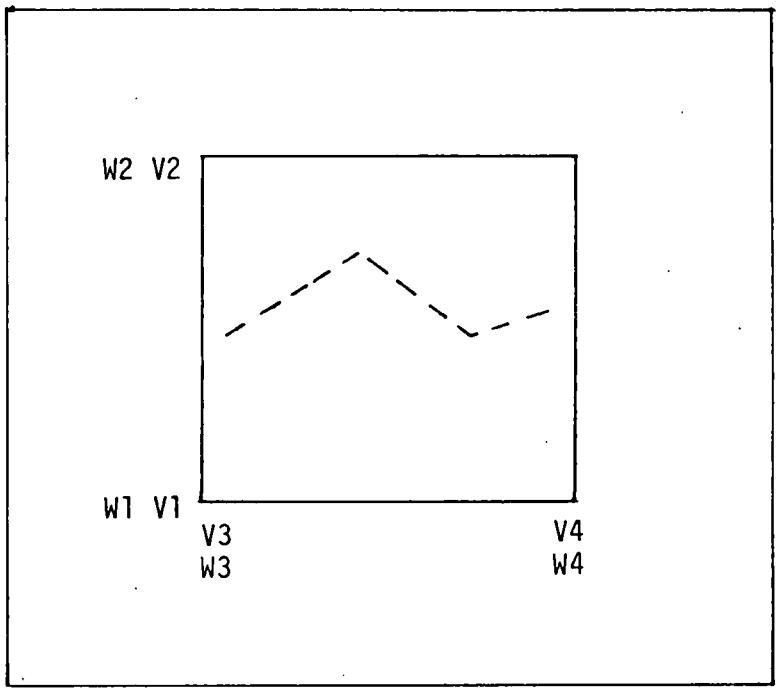
N1 The dash length in terms of the X-axis units if N1>Ø. If N1<Ø, then the magnitude of N1 gives the dash length in terms of the Y-axis units. The actual dash length may be modified slightly to satisfy the end point constraints if N3≠Ø.

N2 The ratio of dash length to dash plus space length. (Ø<N2≤1)

N9 The display address for the plotting device used. For the 4051 screen, N9=32.

VØ The ratio of Y viewport length to X viewport length.

WØ The ratio of Y window length to X window length.



$$V\emptyset = \frac{V2-V1}{V4-V3}$$

$$W\emptyset = \frac{W2-W1}{W4-W3}$$

For the first point, the appropriate initialization must be called.

To start with a dash, GOSUB 3130. To start with a space, GOSUB 3160.

For each successive X,Y pair, GOSUB 3250.

TITLE

DASHED LINES

Operating Hints

If some of the options in this subroutine are not required, then several lines of code may be simplified or eliminated. For example, if the lines always start with a dash, then steps 3140-3160 may be deleted.

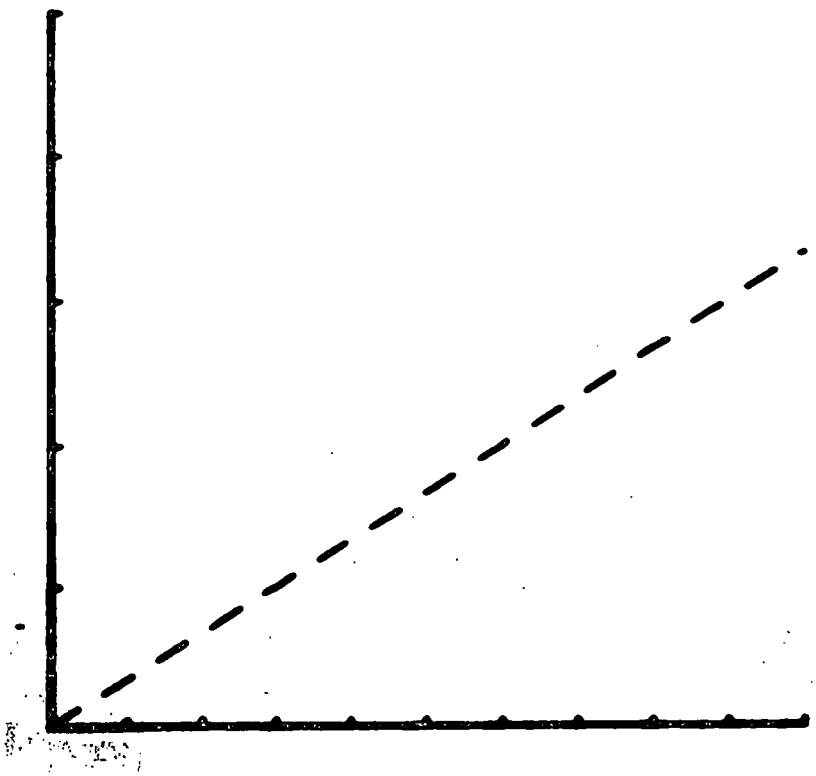
VARIABLE MAP

VARIABLE	TAPE FILE	USAGE
N1		Dash length
N2		Ratio of dash length to dash plus space length
N4-N6, N8		Working variables
N9		Display address (N9=32 for 4051)
VØ		Ratio of Y to X viewport length
WØ		Ratio of Y to X window length
X4		X entry
X5,X6,X8		Working variables
Y4		Y entry
Y5,Y6,Y8		Working variables

TITLE

DASHED LINES

```
100 N1=0.4
110 N2=0.5
120 N9=32
130 V8=1/1.3
140 W8=0.5
150 PAGE
160 VIEWPORT 10,70,0,60
170 WINDOW 0,10,0,5
180 AXIS 1,1
190 X4=0
200 Y4=0
210 GOSUB 3120
220 FOR I=1 TO 10
230 X4=I
240 Y4=I/3
250 GOSUB 3250
260 NEXT I
270 HOME
280 END
```



TITLE

DASHED LINES

All of the subroutines are designed so that the variable names can be changed using an editor program. Also, all REM lines can be deleted.

Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	
Calendar Routines (7 Day Week)	
ORIGINAL DATE: November, 1976 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 8K
AUTHOR: Judy Peterman, Tektronix, Inc., Wilsonville, OR	PERIPHERALS:

ABSTRACT

Files: 1 Program

Statements: 200

This program contains five calendar utility routines based on a seven-day week, Sunday through Saturday. They have been designed specifically for use in programs that calculate and graph financial and other business data, but can be used in any program that involves the collection or display of time-related data. The routine:

1. Gives the data a specific number of time segments before or after a specific date.

2. Gives day number, week number, and day of the week of a specific date based on January 1, 1900.

3. Gives the number of time segments between two specific dates.

4. Verifies a date entry.

5. Unpacks a date.

All routines accommodate five time frames: days, weeks, months, quarters, and years. For example, if you are using days as the time segment in routine #1, 11/17/74 +2 yields 11/19/74; in weeks 11/17/74 +2 yields 12/1/74. The routines will not produce results prior to January 1, 1901.

The routine package comes with examples. Routines and examples require 7.9K bytes to run; the routines alone require 5.3K bytes.

TITLE

CALENDAR ROUTINES (7 DAY WEEK)

DESCRIPTION

All routines accommodate five time frames: days, weeks, months, quarters, and years. The dates are contained in numeric arrays of length 3 by month #, day #, and year #. Each routine will be described in more detail.

1. Date $\pm$ n time segments will go backward or forward in time any specified number of time segments (days, ..., years depending on time frame chosen) from any specified date.

2. Day #, week #, and day of the week returns all three of those values for any specified date. The day # is based on January 1, 1900 being day #1. Week # is based on the week of January 1, 1900 being week #1.

3. Time segments between dates generates a number (positive, negative, or zero) which indicates the number of time segments (based on time frame) between two specified dates.

4. Date entry verify will take a date in string form (the various components of the date must be separated by slashes, "/"), check it for meaningfulness and deposit it in a numeric array. It allows abbreviated dates for some time meshes, and the parameter "END" to be entered in place of month #, or day #.

5. Unpack date routine takes a packed date (all three components of the date put into one floating point number) and puts it into a numeric array. The packed date format is two digits for the year (ones and tens position), two digits for the day number (hundreds and thousands position), and one or two positions for the month number (ten thousands and hundred thousands position). For example, December 5, 1977 is 120577.

TITLE

CALENDAR ROUTINES (7 DAY WEEK)

VARIABLE LIST

Type Legend: A(X) = Numeric array of length X
$ = String variable
S = Simple variable

Variable	Used to Store	Type
B$	Scratch string	$
B2	Time segments between dates: first date	A(3)
B3	Day #, Week #, and day of the week: scratch variable	S
C$	Time mesh string (" days", " weeks", " months", " quarters", " years")	$(9)
CØ	Number of days in the months (= 31, 28, 31, ..., 31)	A(12)
C1	Special purpose array needed to make lines of code work for all time meshes (= 1, 7, 1, 3)	A(4)
C2	Date ± n time segments: initially is the date input by the user and then is returned as the date n time segments away	A(3)
	Day #, week #, and day of the week: the date input by the user	
	Time segments between dates: scratch date	
	Date entry verify: the date output to the user as the equivalent to the date input in string form	
C3	Time segments between dates: second date input by the user	A(3)
C4	Day #, week #, and day of the week: returned as the day # for the date input by the user	S
	Time segments between dates and unpack date: scratch variable	

TITLE

CALENDAR ROUTINES (7 DAY WEEK)

VARIABLE LIST (continued)

Variable	Used to Store	Type
C5	Date ± n time segments: # of time segments (either zero, positive, or negative)	S
	Time segments between dates: scratch variable	
C6	Date ± n time segments, time segments between dates: scratch variable	S
	Unpack date: packed date	
C7	Time segments between dates: scratch variable	S
	Day #, week #, and day of the week: day of the week	
	Date entry verify: position in the date string of the first "/"	
C8	Time segments between dates: the number of time segments	S
	Date entry verify: position of the second "/" in the date string	
C9	Date entry verify: the date in unpacked form	A(3)
E1	Date entry verify: error flag for bad date	S
R$	Date entry verify: string containing date as entered by user	$(24)
T1	Date ± n time segments, time segments between dates, date entry verify: time mesh (1=days, 2=weeks, 3=months, 4=quarters, 5=years)	S

TITLE

CALENDAR ROUTINES (7 DAY WEEK)

EXAMPLE

```
It is in week #4037

                Calendar Routines

Enter time mesh (i.e. 1=daily, 2=weekly, 3=monthly,
                     4=quarterly, 5=yearly)
```

TITLE

CALENDAR ROUTINES (7 DAY WEEK)

Calendar Routines

```
Enter time mesh (i.e. 1=daily, 2=weekly, 3=monthly,
                     4=quarterly, 5=yearly) 2

Enter packed date (i.e. 120577) 41377
Your date unpacked is 4/13/77


Permitted format for "dates":
   MM/dd/yy  (e.g. 11/7/74)

Enter another date 3/2/77

1st date is Apr 13, 1977
2nd date is Mar 2, 1977
Time between dates is -6 weeks

Enter a date 6/25/77
Enter # of weeks (+ or -) -13

Date plus -13 weeks is Mar 26, 1977


Enter date 3/26/77
It is a  Saturday
It is day #28209
It is in week #4030
```

Calendar Routines

Tektronix®
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	
Calendar Routines (5 Day Week)	
ORIGINAL DATE: November, 1976 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 8K
AUTHOR: Judy Peterman, Tektronix, Inc., Wilsonville, OR	PERIPHERALS:

ABSTRACT

Files: 1 Program

Statements: 310

This program contains five calendar utility routines based on the premise that a week is five days, Monday through Friday. The routines are the same as those found in "Calendar Routines (7 Day Week)"; see Program 10 of PROGRAMMING AIDS T1.

1. Date ± n time segments.

2. Date #, week #, and day of the week.

3. Time segments between dates.

4. Date entry verify.

5. Unpack data.

The routines with the examples take 8.3K bytes to run but the routines alone take only 5.6K bytes.

TITLE

CALENDAR ROUTINES (5 DAY WEEK)

DESCRIPTION

All routines accommodate five time frames: days, weeks, months, quarters, and years. The dates are contained in numeric arrays of length 3 by month #, day #, year #. Each routine will be described in more detail.

1. Date $\pm$ n time segments will go backward or forward in time any specified number of time segments (days, ..., years depending on time frame chosen) from any specified date.

2. Day #, week #, and day of the week returns all three of those values for any specified date. The day # is based on January 1, 1900 being day #1. Week # is based on the week of January 1, 1900 being week #1.

3. Time segments between dates generates a number (positive, negative, or zero) which indicates the number of time segments (based on chosen time frame) between two specified dates.

4. Date entry verify will take a date in string form (the various components of the date must be separated by slashes, "/"), check it for meaningfulness and deposit it in a numeric array. It allows abbreviated dates for some time meshes, and the parameter "END" to be entered in place of month # or day #.

5. Unpack date routine takes a packed date (all three components of the date put into one floating point number) and puts it into a numeric array. The packed date format is two digits for the year (ones and tens position), two digits for the day number (hundereds and thousands position), and one or two positions for the month number (ten thousands and hundred thousands position). For example, December 5, 1977 is 120577.

TITLE

CALENDAR ROUTINES (5 DAY WEEK)

VARIABLE LIST

Type Legend: A(X) = Numeric Array of length X
$ = String variable
S = Simple variable

Variable	Used to Store	Type
B$	Scratch String	$(72)
B2	Time segments between dates: first date	A(3)
B3	Day #, week #, and day of the week: Scratch variable	S
C$	Time mesh string (" days", " weeks", " months", " quarters", " years")	$(9)
CØ	Number of days in the months (i.e. = 31, 28, 31, ..., 31)	A(12)
C1	Special purpose array needed to make lines of code work for all time meshes (values are 1, 7, 1, 3)	A(4)
C2	Date ± n time segments: initially is the date input by the user and then is returned as the date n time segments away	A(3)
	Day #, week #, and day of the week: the date input by the user	
	Time segments between dates: scratch date	
	Date entry verify: the date output to the user as the equivalent to the date input in string form	
C3	Time segments between dates: second date input by user	A(3)

TITLE

CALENDAR ROUTINES (5 DAY WEEK)

VARIABLE LIST (continued)

Variable	Used to Store	Type
C4	Day #, week #, and day of the week: returned as the day # for the date input by the user Time segments between dates and unpack date: scratch variable	S
C5	Date ± n time segments: # of time segments (either zero, positive, or negative) Time segments between dates: scratch variable	S
C6	Date ± n time segments, time segments between dates: scratch variable Unpack date: packed date	S
C7	Date ± n time segments, time segments between dates: scratch variable Day #, week #, and day of the week: day of the week Date entry verify: position of the first "/" in the date string	S
C8	Time segments between dates: the number of time segments Date entry verify: position of the second "/" in the date string	S
C9	Unpack date: the date in unpacked form C9(1)=month #, C2(2)=day #, C9(3)=year #	A(3)

TITLE

CALENDAR ROUTINES (5 DAY WEEK)

VARIABLE LIST (continued)

Variable	Used to Store	Type
E1	Date entry verify: error flag for bad date	S
R$	Date entry verify: string containing date as entered by user	$(24)
T1	Date ± n time segments, time segments between dates, date entry verify: time mesh (1=days, 2=weeks, 3=months, 4=quarters, 5=years)	S

TITLE

CALENDAR ROUTINES (5 DAY WEEK)

EXAMPLE

```
                    Calendar Routines

Enter time mesh (i.e. 1=daily, 2=weekly, 3=monthly,
                      4=quarterly, 5=yearly) 2

Enter packed date (i.e. 120577) (not Sat or Sun) 33077
Your date unpacked is 3/30/77


Permitted format for "dates":
   mm/dd/yy  (e.g. 11/7/74)

Enter another date (not Sat or Sun) 4/22/77

1st date is Mar 30, 1977
2nd date is Apr 22, 1977
Time (excluding Sat & Sun) between dates is 3 weeks

Enter a date (not Sat or Sun) 4/25/77
Enter # of weeks (+ or -) 10

Date plus 10 weeks (w/o Sat & Sun) is Jul 4, 1977


Enter date 4/21/77
It is a  Thursday
It is day #28235
It is in week #4034

                    Calendar Routines
```

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	
FORTRAN to BASIC Converter	
ORIGINAL DATE: March, 1980 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 32K
AUTHOR: Mark Mehall, Tektronix, Inc., Wilsonville, Or	PERIPHERALS: 4924 Tape Drive

ABSTRACT

4050R06 Editor ROM

Files: 2 ASCII Program
Requires 2 pre-MARKed files

Statements: 979

This program is designed to convert FORTRAN to 4050 Series BASIC. The program is based on the USA Standard FORTRAN, X3.9-1966.

The FORTRAN statement labels, variables and subroutine names are changed to their BASIC counterparts and remembered for references throughout the program.

The majority of FORTRAN statements are changed into BASIC by this program. The statements that are not directly compatible are made into REMARK's and can be modified using the EDITOR ROM or the 4050 Series Line Editor. The FORTRAN statements: READ, WRITE, FORMAT, IF, GO TO, DO, DIMENSION, CALL, END, RETURN, STOP, SUBROUTINE, and CONTINUE are automatically changed to BASIC. The FORTRAN internal routines are also converted to the corresponding BASIC routines.

The program also prints tables of corresponding FORTRAN statement numbers to BASIC line numbers, FORTRAN variable names to BASIC variables, and FORTRAN subroutine names to BASIC line numbers.

TITLE

FORTRAN to BASIC Converter

PRELIMINARY OPERATIONS

Tape Files Used (all files ASCII)

INTERNAL TAPE

Use a scratch tape. Transfer the FORTRAN to BASIC Converter--PASS ONE program to File 1.

MARK file 2 the size of FORTRAN file being converted.

Transfer the FORTRAN to BASIC Converter--PASS TWO program to file 3.

Insert this tape into the internal tape drive of the 4050.

4924 TAPE DRIVE

File 1: FORTRAN Source to be converted.

File 2: Will contain the converted program (BASIC).

MARK file 2 to the size of file 1 on the 4924 (FORTRAN Source).

TITLE

FORTRAN to BASIC CONVERTER

GENERAL USES

This program aids users in the conversion of FORTRAN programs to 4050 Series BASIC, thus saving expensive development time and allowing access to a large number of previously written routines. General uses include the changing of programs that have been transmitted to the 4050 Series tape from a host computer or the conversion of FORTRAN code that has been stored on tape using the 4050R06 Editor ROM.

OPERATION

To run this program insert a tape containing the FORTRAN program to be converted on file one into the 4924. A scratch file should be marked on file two of the 4924 to be at least as large as the FORTRAN source on file one. After the tape has been inserted into the 4924 and the second file marked, insert the program tape into the internal drive and press 'AUTOLOAD'.

This program is a two pass process: the first pass reads the FORTRAN source from the 4924 Tape Drive and uses a scratch file on the internal drive (this file is already marked). The second pass reads the scratch file from the internal drive and creates a file containing BASIC on file two of the 4924. The second file on the 4924 should be marked by the user (see above). Both passes print information on the screen. The first pass prints the FORTRAN statements with BASIC line numbers and BASIC variables, the second pass prints the BASIC statements. After the conversion process is complete, the program prints three tables: FORTRAN statement numbers to BASIC line numbers, FORTRAN variables names to BASIC variables, and FORTRAN subroutine names to BASIC line numbers (for GOSUB's).

TITLE

FORTRAN to BASIC CONVERTER

FORTRAN/4050 SERIES BASIC COMPARISON

The following is a comparison between FORTRAN and 4050 Series BASIC. This should help with the conversion of statements that are not directly compatible with BASIC.

SPECIAL CHARACTERS

FORTRAN		BASIC
+	PLUS	+
-	MINUS	-
*	ASTERISK	*
**	EXPONENTATION	^
/	SLASH	/
=	EQUAL	=
,	COMMA	,
(	OPEN PAREN	(
)	CLOSE PAREN	)
$	DOLLAR SIGN	$
' or "	QUOTE	"

Note: The special characters are the same for both FORTRAN and BASIC with the exception of exponentation (** vs. ^) and the quote (' or " vs. ").

TITLE

FORTRAN to BASIC CONVERTER

INTERNAL ROUTINES

FORTRAN		BASIC
ABS/IABS	ABSOLUTE VALUE	ABS
FLOAT	INTERGER TO REAL	(None)
IFIX	REAL TO INTEGER	INT
SIGN/ISIGN	TRANSFER OF SIGN	(FNS)
DIM/IDIM	POSITIVE DIFFERENCE	(FND)
AMOD/MOD	MODULAR ARITHMETIC	(FNM)
AINT/INT	REAL TO INTEGER	INT
AMAX0/MAX0	MAXIMUM	MAX
AMIN0/MIN0	MINIMUN	MIN
EXP	E TO THE POWER	EXP
ALOG	NATURAL LOG	LOG
ALOG10	COMMON LOG	LGT
ATAN	ARCTANGENT	ATN
SIN	SINE	SIN
COS	COSINE	COS
SQRT	SQUARE ROOT	SQR
RAND	RANDOM NUMBER	(RND)

Notes: FORTRAN AND BASIC often differ in the syntax of the functions the (FNx) of BASIC may be used to emulate the operation. FORTRAN allows multiple variables in the MIN and MAX functions, eg. D = MIN(A,B,C)
BASIC would use D = A MIN B MIN C
FORTRAN also has complex and hyperbolic routines, these are not converted by this program.

RELATIONAL OPERATORS

FORTRAN		BASIC
.EQ.	EQUAL	=
.NE.	NOT EQUAL	<>
.LT.	LESS THAN	<
.GT.	GREATER THAN	>
.GE.	GREATER/EQUAL	=>
.LE.	LESS/EQUAL	<=
.AND.	LOGICAL AND	AND
.OR.	LOGICAL OR	OR
.NOT.	LOGICAL NOT	NOT
.TRUE.	TRUE	1
.FALSE.	FALSE	0

TITLE

FORTRAN to BASIC CONVERTER

VARIABLES

FORTRAN uses variable names of up to six characters, the first of which must be a letter. Basic uses a single letter or a letter and a single digit. FORTRAN uses names that begin with the letters 'I' through 'N' as integer variables, BASIC maintains integers and floating point variables automatically. BASIC also supports character variables as a letter followed by a '$'. FORTRAN's manipulation of character data is weak (at least in ANSI 1966) and varies depending on the computer and complier. Most applications using character data can be easily converted to BASIC. This program does not attempt to identify and convert variables use to manipulate character data.

ARRAYS

FORTRAN supports arrays of three or more dimensions, 4050 Series BASIC handles two dimensions.

ACCURACY and RANGE

FORTRAN/COMPUTER	BASIC/4050 SERIES
IBM 1130: Std. Prec: 7 Digits Dbl. Prec:10 Digits Exp: -39 to +38	Prec: 14 Digits Exp: -307 to +307
IBM 370: Std. Prec: 7 Digits Dbl. Prec:16 Digits Exp: -78 to +75	
CDC 7000: Std. Prec:14 Digits Dbl. Prec:29 Digits Exp: -293 to +322	

TITLE

FORTRAN to BASIC CONVERTER

KEYWORDS

FORTRAN	BASIC
ASSIGN	REMARK
BLOCK DATA	REMARK
CALL	GOSUB
CHARACTER	REMARK
COMMON	REMARK
CONTINUE	REMARK
DATA	(READ/DATA)
DECODE	REMARK
DEFINE	(DEF/GOSUB)
DIMENSION	DIM
DO	FOR/NEXT
DOUBLE PRECISION	REMARK
ENCODE	REMARK
END	END
ENTRY	REMARK +
EQUIVALENCE	REMARK
ERR	(none)
EXTERNAL	REMARK
FORMAT	IMAGE
FUNCTION	REMARK +
GO TO	GO TO
IF	IF/GO TO
IMPLICIT	REMARK
INTEGER	REMARK
LOGICAL	REMARK
PAUSE	(INPUT/WAIT)
READ	INPUT
REAL	REMARK
RETURN	RETURN
STOP	STOP
SUBROUTINE	REMARK +
WRITE	PRINT

Note: The BASIC statements in parenthesis are commands that can be manually substituted for the FORTRAN statement. The REMARK statements with a '+' are for FORTRAN statements that, when converted, generate other BASIC code or references.

TITLE

FORTRAN to BASIC CONVERTER

NOTES ON USAGE

The following steps should be useful in the conversion process:

STEP 1:
Using the EDITOR ROM (4050R06):

1. Make FORTRAN variables the same in Main and Subroutines, if possible. Otherwise use separate variables.
2. Remove carriage control characters from FORMAT statements and replace with linefeeds.

STEP 2: Run the FORTRAN to BASIC Conversion Program.

STEP 3:
Using the EDITOR ROM:

1. Write the FNx statements (ISIGN,AMOD,etc.)
2. Convert the Statement Functions.
3. Add system SUBROUTINES, if necessary.

This program does not guarantee a perfectly running BASIC program from any FORTRAN program. It does provide a starting point for the conversion effort. Some programs will convert to BASIC easily (see example one). However, since FORTRAN has different constructs than BASIC, some programs may be difficult to change. This program should be helpful in the FORTRAN to BASIC conversion of any program.

TITLE

FORTRAN to BASIC CONVERTER

EXCEPTIONS

The following are NOT considered by this program (at this time):

Auxiliary I/O (BACKSPACE, ENDFILE, REWIND)
ASSIGNed GO TO's
BLOCK DATA
Character Variables (Treated as regular variables)
COMMON variables (All variables are common in BASIC)
COMPLEX variables
DATA statements
DOUBLE PRECISION
EQUIVALENCE
Hollerith Data (Allowed only in FORMAT statements)
Implied DO loops
STATEMENT Functions
END and ERR Conditions on I/O Statements (Ignored)

RESTRICTIONS

The following are restrictions on the FORTRAN input:

1000 Different FORTRAN Statement Numbers (*)
50 Different Subroutines Names (*)
260 Different FORTRAN Variable Names
6500 Card Images (Approximate)

Note: The items with a (*) may be changed by the user (if memory permits).

TITLE

FORTRAN to BASIC CONVERTER

VARIABLES USED

A$	Variable Used to Hold Next Card Image
A	Variable Used to Clear SRQ from 4924
B$	Table of BASIC Line Numbers Used in a FOR-NEXT Loop
B	Variable Used for the BASIC Line Number
B0	Work Variable for BASIC Line Number from FORTRAN
C$	Variable Used to Hold FORTRAN Card Image
D	Device Number for Output from Analysis Section
D$	Table of BASIC Variables Used in a FOR/NEXT Loop
D0	Variable Used to Convert Floating Point FORMAT to IMAGE
F$	Table of FORTRAN SUBROUTINE Names (used with L)
F0	Variable Used as a Flag to Process New Statements
I$	Table of Internal FORTRAN Routines (Used with J$)
I0	Variable Used as an Index into P
J$	Table of Basic Functions (Used with I$)
K$	Table of FORTRAN Keywords
L	Table of BASIC Line Numbers (Used with F$)
L0	Variable Used in the FORMAT to IMAGE Section
N$	Work Variable
N	Work Variable for BASIC Line NUMBER
N0	Work Variable Used to count '(' in IF
N1	Work Variable Used to count ')' in IF
O$	Table of FORTRAN Logical Operators
P$	Table of Basic Logical Operators
P	Table of Pointers into T$
P0	Work Variable Used as a Pointer in a String
P1	Work Variable Used as a Pointer in a String
P2	Work Variable Used as a Pointer in a String
P3	Work Variable Used as a Pointer in a String
P4	Variable Used to determine Statement Processing
Q$	Work Variable Used to pad Strings for Searches
R$	Work Variable Used with R in IF Section
R	Work Variable Used to Calculate next Line Number in IF
S$	Work Variable
S0	Variable Used as a Flag for SUBROUTINE's END
T$	Table of FORTRAN Statement Numbers (6 char. each)
T	Array of BASIC Line Numbers corresp. to T$
T0	Work Variable Used as a Pointer
T1	Work Variable Used with T0
U$	Work Variable Used to pad Strings
U	Undefined (or System) FORTRAN SUBROUTINE Line Number
V$	Table of FORTRAN Variables

TITLE

FORTRAN to BASIC CONVERTER

EXAMPLE CONVERSIONS

The following are examples of output from the conversion program. Both the FORTRAN source and corresponding BASIC programs are listed.

FORTRAN Example One

```
77    FORMAT(I4,2X,2F5.2,T23,I2)
88    FORMAT(I4,F7.2,2F8.2,T79,I2)
C
      READ(1,77)NUMEMP,HOURS,RATE,KODE
C
      REGPAY=HOURS*RATE
C
C
C
C     ***BONUS IS BASED ON YEARS OF EMPLOYMENT
C           AS INDICATED BY KODE
C           KODE      YEARS
C             1     MORE THAN FIVE
C             2     1 TO 5
C             3     LESS THAN 1
C
      GOTO(55,65,75),KODE
55    BONUS=REGPAY*2.00
      TOTPAY=REGPAY+BONUS
      WRITE(3,88)NUMEMP,REGPAY,BONUS,TOTPAY,KODE
1     STOP
C
65    BONUS=REGPAY
      TOTPAY=REGPAY+BONUS
      WRITE(3,88)NUMEMP,REGPAY,BONUS,TOTPAY,KODE
2     STOP
C
75    BONUS=REGPAY/2.00
      TOTPAY=REGPAY+BONUS
      WRITE(3,88)NUMEMP,REGPAY,BONUS,TOTPAY,KODE
C
3     STOP
      END
```

TITLE

FORTRAN to BASIC CONVERTER

BASIC Example One

```
100 IMAGE 4D,2X,2(2D.2D),23T,2D
110 IMAGE 4D,4D.2D,2(5D.2D),79T,2D
120 REM
130 INPUT N0,H0,R0,K0
140 REM
150 R1=H0*R0
160 REM
170 REM
180 REM
190 REM       ***BONUS IS BASED ON YEARS OF EMPLOYMENT
200 REM             AS INDICATED BY KODE
210 REM             KODE      YEARS
220 REM               1      MORE THAN FIVE
230 REM               2      1 TO 5
240 REM               3      LESS THAN 1
250 REM
260 GOTO K0 OF 270,320,370
270 B0=R1*2.00
280 T0=R1+B0
290 PRINT USING 110:N0,R1,B0,T0,K0
300 STOP
310 REM
320 B0=R1
330 T0=R1+B0
340 PRINT USING 110:N0,R1,B0,T0,K0
350 STOP
360 REM
370 B0=R1/2.00
380 T0=R1+B0
390 PRINT USING 110:N0,R1,B0,T0,K0
400 REM
410 STOP
420 END
```

TITLE

FORTRAN to BASIC CONVERTER

FORTRAN STATEMENT NUMBERS TO BASIC LINE NUMBERS

FORTRAN	BASIC
ROUTINE: (MAIN)	
77	100
88	110
55	270
1	300
65	320
2	350
75	370

FORTRAN VARIABLES TO BASIC VARIABLES

FORTRAN	BASIC
BONUS	B0
HOURS	H0
KODE	K0
NUMEMP	N0
RATE	R0
REGPAY	R1
TOTPAY	T0

TITLE

FORTRAN to BASIC CONVERTER

FORTRAN Example Two

```
      I=0
      S=3.141593
      ROOT=1.7
1     ROOT=(ROOT**2+S)/(2.0*ROOT)
      I=I+1
      TEST=ROOT**2-S
      IF (TEST) 8,5,9
4     FORMAT(' PI**.5= ',F8.6/'0',I4,' ITERATIONS')
5     WRITE(3,4) ROOT,I
      STOP
8     TEST=-1.0*TEST
9     IF (TEST-.000005) 5,5,1
      END
```

BASIC Example Two

```
100 I=0
110 S=3.141593
120 R0=1.7
130 R0=(R0↑2+S)/(2.0*R0)
140 I=I+1
150 T0=R0↑2-S
160 GO TO SGN (T0)+2 OF  200,180,210
170 IMAGE " PI**.5= ",1D.6D/"0",4D," ITERATIONS"
180 PRINT USING 170: R0,I
190 STOP
200 T0=-1.0*T0
210 GO TO SGN (T0-.000005)+2 OF  180,180,130
220 END
```

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE

Flow Diagrammer (tape version)

ORIGINAL DATE	REVISION DATE
December, 1977	

EQUIPMENT AND OPTIONS REQUIRED

16K

AUTHOR

Keith S. Reid-Green
Educational Testing Service
Princeton, NJ

PERIPHERALS

4662 Plotter

ABSTRACT

Files: 5 ASCII Porgram
Requires dedicated tape

Statements: 917

The program is used to design, store, recall and modify flow diagrams for use in program and system documentation.

A diagram consists of a heading, 10 different box types, their connecting lines and labeling. Boxes and lines may be solid or dotted and may be arranged up to 4 across and 9 deep on a page.

The program consists of 5 phases:

1. Main menu and function keys

2. Enter boxes

3. Connect, insert, delete boxes

4. Enter box data and heading

5. Store or retrieve diagrams

The first 62 files of a tape must be dedicated to this program. Files 1 through 5 contain the program; files 6 through 42 contain information about the current diagram; and files 43 through 62 store up to 20 diagrams.

TITLE

Flow Diagrammer

PRELIMINARY OPERATING INSTRUCTIONS

Flow Diagrammer must occupy its own tape.

Step 1. Using one of the tape transfer procedures outlined on page iii, transfer files 18 through 22 from the PROGRAMMING AIDS T1 master tape to the Flow Diagrammer tape.

Step 2. Insert Flow Diagrammer tape into the 4050 tape drive.

```
FIND 6
MARK 1, 1024
FIND 7
MARK 36, 768
FIND 43
MARK 20, 3072
```

The Flow Diagrammer program tape is now ready to use. PRESS AUTOLOAD.

TITLE

FLOW DIAGRAMMER

TABLE OF CONTENTS

TITLE

FLOW DIAGRAMMER

Description: The program consists of five phases:

PHASE	FUNCTION
1.	Main menu and function keys
2.	Enter boxes
3.	Connect, insert, delete boxes
4.	Enter box data and heading
5.	Store or retrieve diagrams

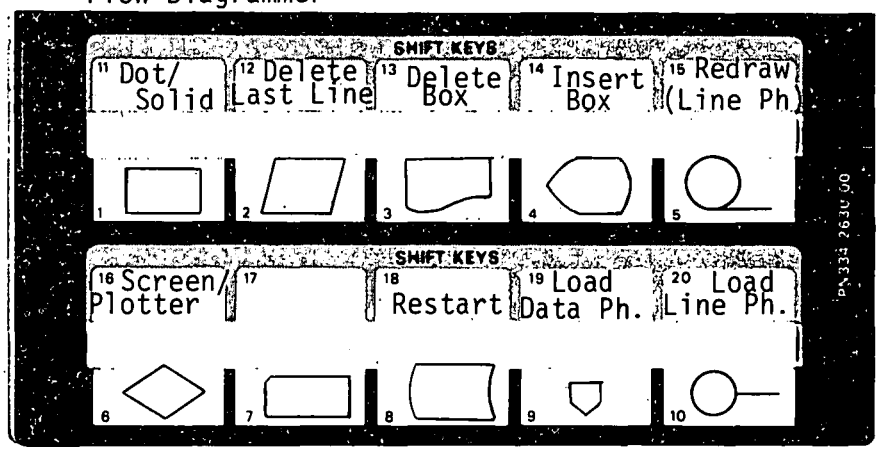
These phases are stored on files 1 - 5 of the magnetic tape. Files 6 through 42 contain information about the current diagram; file 6 contains the diagram heading, or title, the box definition array Z and the number of connectors Z1. Files 43 through 62 are used to store diagrams.

The user should arrive at the terminal with a rough-flow diagram drawn on the diagramming matrix form. (Page 107).

TITLE

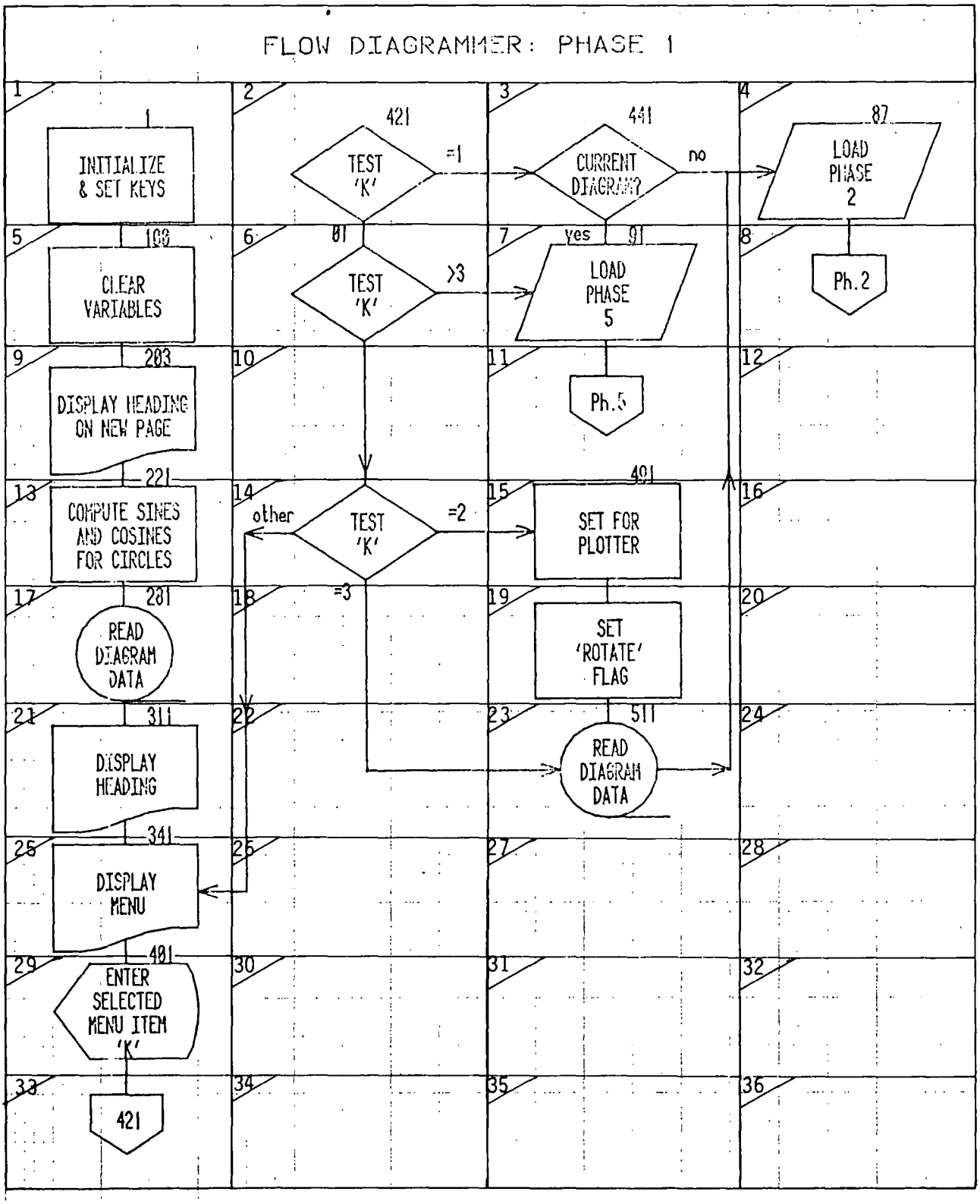
FLOW DIAGRAMMER

TITLE Flow Diagrammer TAPE # FILE #



TITLE

FLOW DIAGRAMMER



TITLE

FLOW DIAGRAMMER

To begin the program, press 'AUTO LOAD'. If the magnetic tape does not move immediately, press 'BREAK' twice, then press 'AUTO LOAD'. After a few seconds, the words 'FLOW DIAGRAMMER' will appear near the top of the screen and the tape will move. Then either the message 'There is no current diagram' or 'Current diagram is (title)' will appear, followed by the menu:-

```
                FLOW DIAGRAMMER


There is no current diagram


Enter:

  1 to start a new diagram
  2 to redraw current diagram
  3 to change current diagram
  4 to retrieve a stored diagram
  5 to destroy part or all of stored data:
```

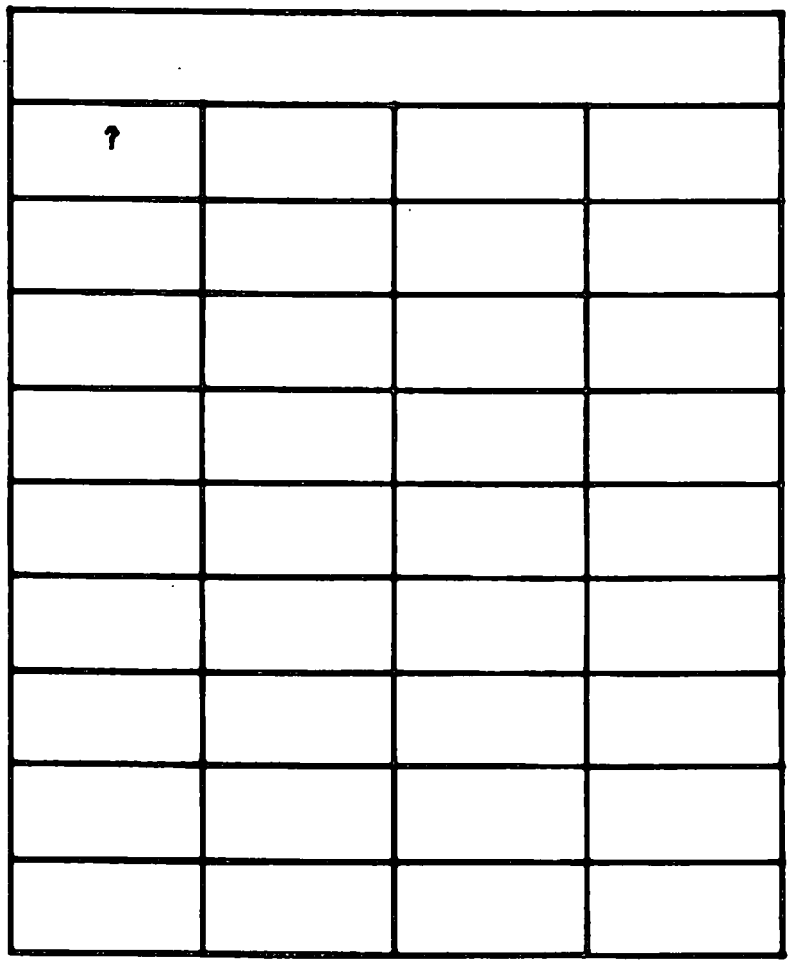
If there is no current diagram, selecting menu items 2 or 3 makes no sense. Selecting item 1 will initiate phase 2, selecting 4 or 5 will initiate phase 5.

If a current diagram exists, selecting 1, 4 or 5 will initiate phase 5, selecting 2 will cause the current diagram to be plotted, and selecting 3 will initiate phase 3. If 2 is selected, a piece of 11 x 16 inch paper must be in place on the plotter and the plotter otherwise prepared for action.

TITLE

FLOW DIAGRAMMER

Starting a new diagram: Selecting item 1 from the main menu causes either phase 5 or phase 2 to be loaded from tape. Phase 5 will be loaded if there is a diagram, so that it may be saved or deleted. Otherwise, phase 2 is loaded and the diagram matrix is drawn on the screen.

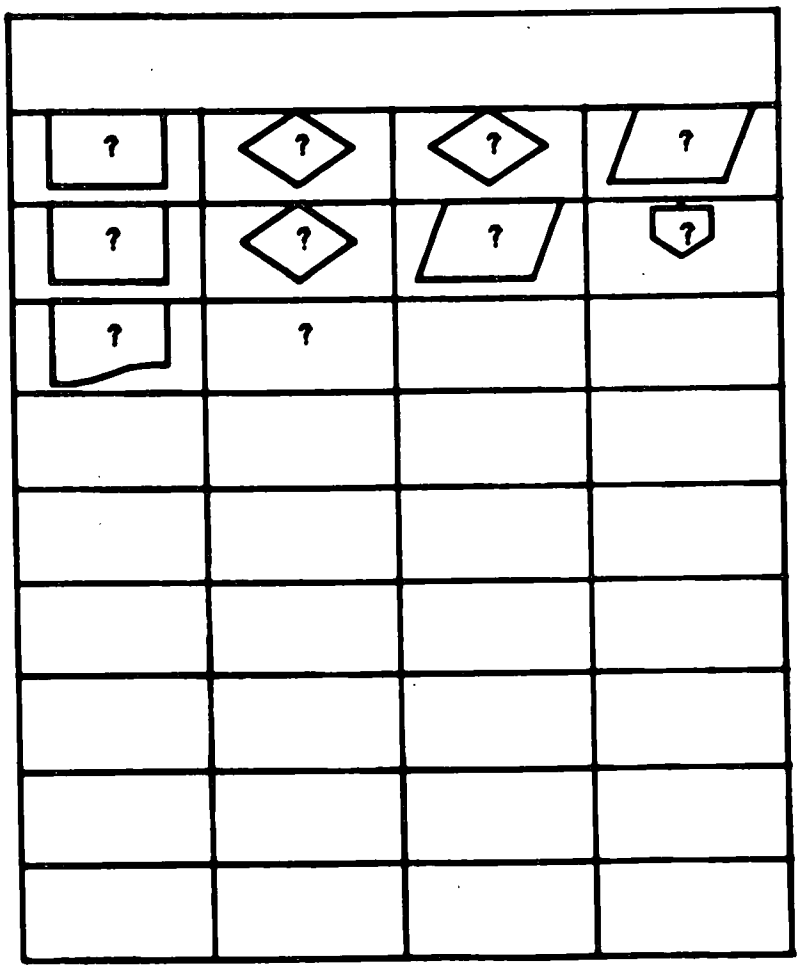


TITLE

FLOW DIAGRAMMER

The question mark moves from section to section as box types are selected by User Definable Keys. The question mark may be moved without selecting a box type by pressing 'return'. When a UDK is pressed, it is necessary to wait for the box to appear on the screen before pressing another UDK.

Shown below is the result of pressing UDKs 1, 6, 6, 2, 1, 6, 2, 9 and 3. The next section is to be left blank, so 'return' will be pressed and the question mark will appear in the following section.

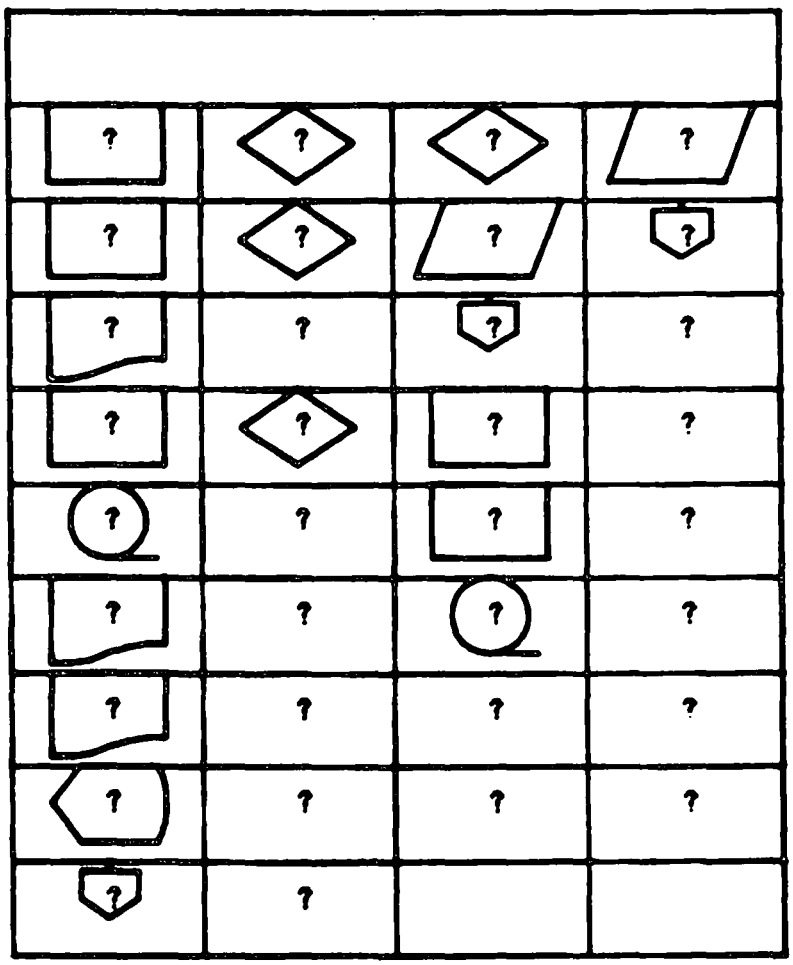


TITLE

FLOW DIAGRAMMER

In phase 2, User Definable Keys 1-10, 11, 16, 18 and 20 are meaningful. Key 11 is used to change the line drawing mode from solid to dotted lines or vice versa. Key 16 will allow the screen output to appear on the plotter, but should not be used in phase 2. Key 18 allows return to phase 1 in order to restart. Key 20 will be pressed to enter phase 3 after completing the assignment of boxes to sections. If a box is assigned to the last section or if 'return' is pressed when the question mark reaches the last section, phase 3 is loaded.

Note that keys 12-15 are <u>not</u> active during phase 2. Errors in the assignment of boxes must be corrected in phase 3. Shown below is the final assignment of boxes for the sample diagram. Key 20 should now be pressed.

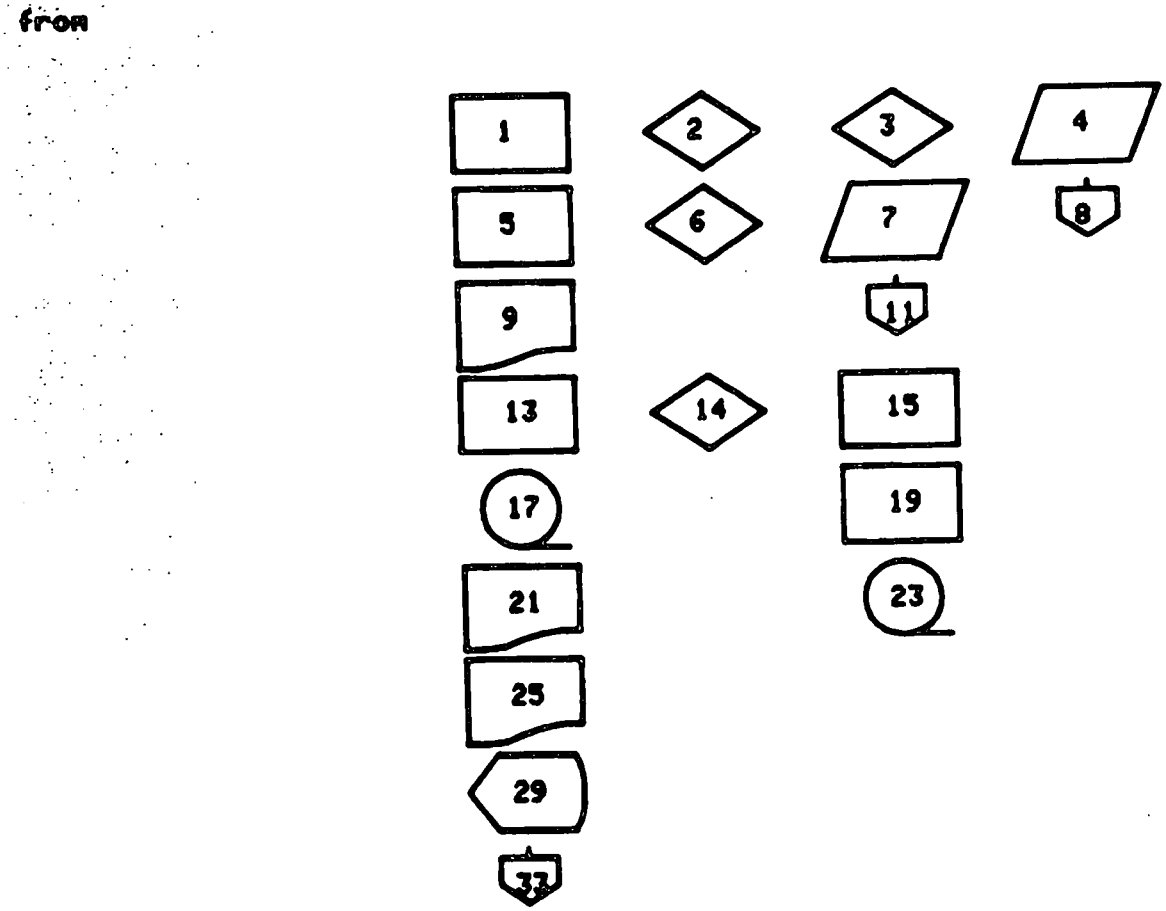


TITLE

FLOW DIAGRAMMER

Connecting boxes and correcting errors: When phase 3 is loaded, the screen is cleared and the defined boxes are displayed, showing the assigned box numbers. All User Definable Keys are currently meaningful except 17, (not used) and 20. Consequently, boxes may be added or removed at any time during this phase. If a box has been connected to other boxes and is then removed, all connections to and from the box are also removed.

Shown below is the initial phase 3 display.



TITLE

FLOW DIAGRAMMER

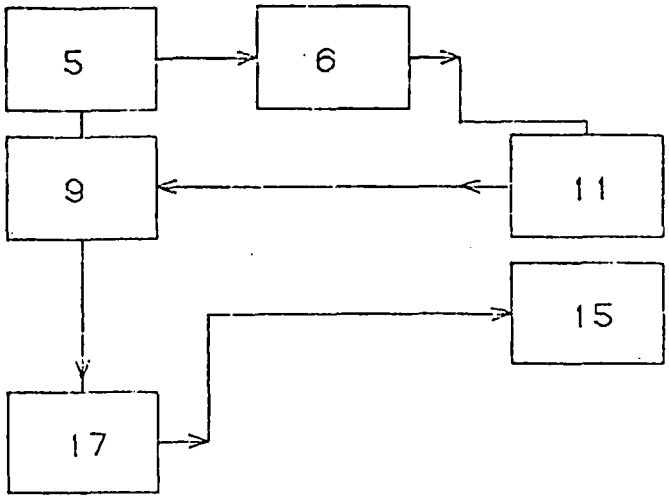
The terminal is now awaiting input. For each pair of boxes to be connected, the user enters the 'from' box number and presses 'return'. The word 'side' is displayed, and the user enters 'r' for right-hand side, 'b' for bottom, or 'l' for left-hand side, and presses 'return'. The same procedure is repeated for the 'to' box, except that 'r', 'l' or 't' for top are the permissible sides. Then the type of line is input when the word 'type' is displayed.

There are three line types; 1, 2 or 3. Type 1 connects boxes by drawing in the up or down direction from box to box, then in the 'across' direction to complete the connection. Type 2 draws half way across, then up or down, (with an arrow in the middle of this line) then the other half way across. Type 3 draws across then up/down. Types 4, 5 and 6 are dotted line equivalents of types 1, 2 and 3, respectively. Examples are shown on page 10; note particularly the connections from boxes 6 to 11 and 17 to 15.

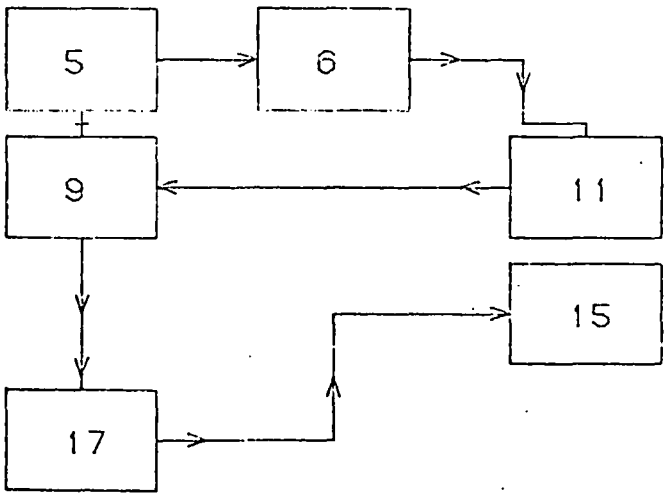
TITLE

FLOW DIAGRAMMER

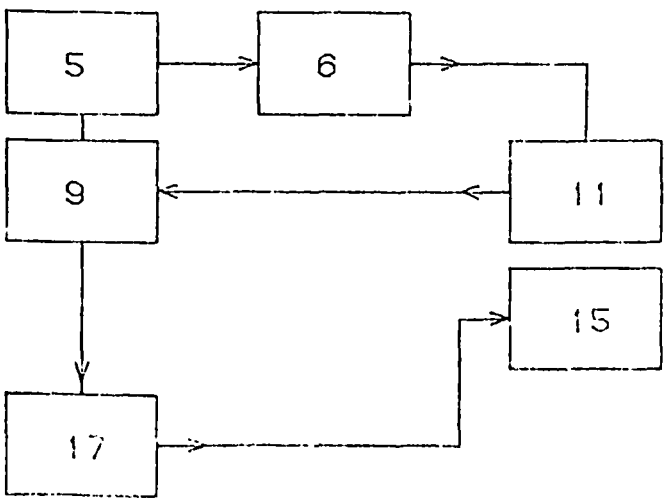
TYPE 1



TYPE 2



TYPE 3

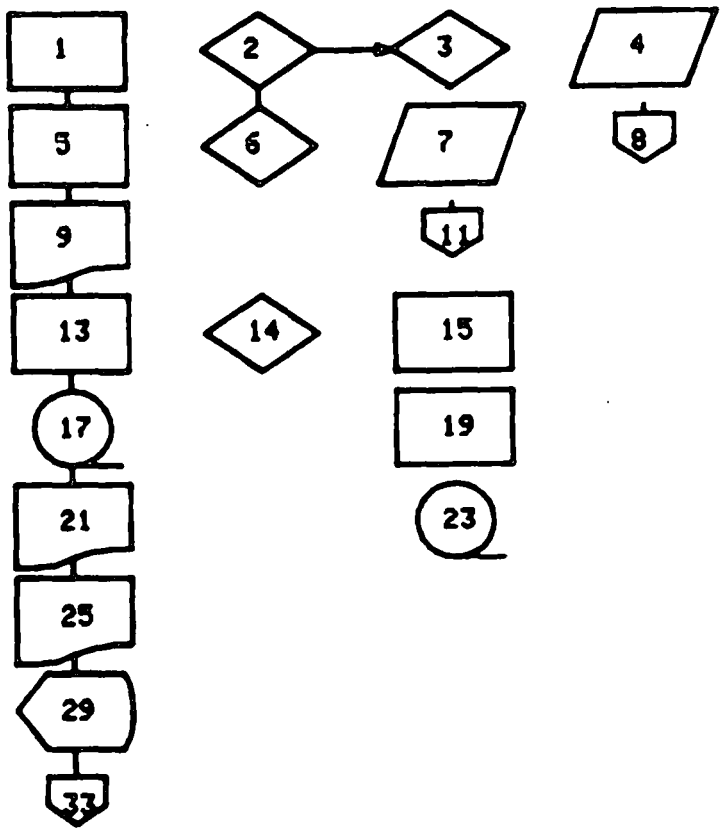


TITLE

FLOW DIAGRAMMER

In the following illustration, the user has entered eleven commands to connect boxes and is approaching the end of a 'page' on the screen. Pressing the 'redraw' key, (UDK 15) will cause the screen to be erased, the diagram redrawn and 'from' to appear at the top of the screen. Connecting may continue.

```
from 1    side B
to   5    side T
type 1
from 5    side B
to   9    side T
type 1
from 9    side B
to   13   side T
type 1
from 13   side B
to   17   side T
type 1
from 17   side B
to   21   side T
type 1
from 21   side B
to   25   side T
type 1
from 25   side B
to   29   side T
type 1
from 29   side B
to   33   side T
type 1
from 2    side B
to   6    side T
type 1
from 2    side R
to   3    side L
type 1
from
```

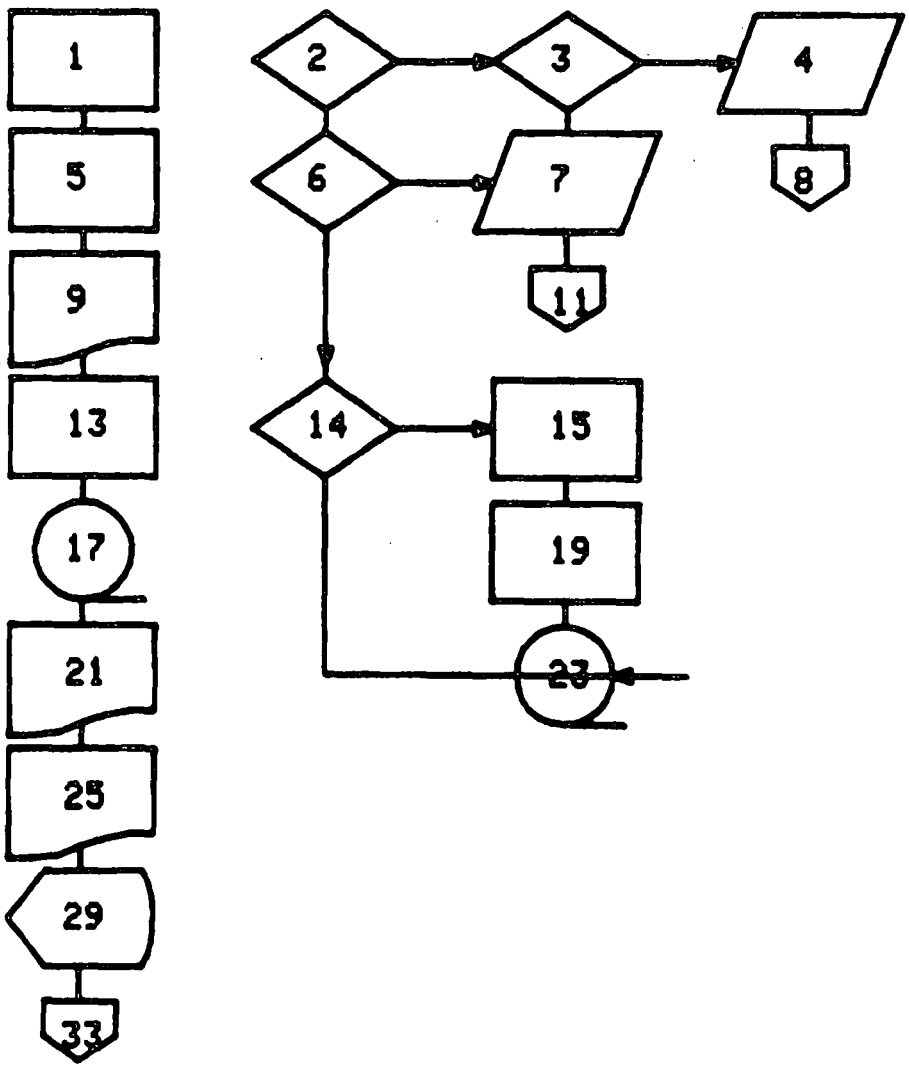


TITLE

Flow Diagrammer

After having entered several additional commands, the user has made a mistake, by requesting that the bottom of box 14 be connected to the right-hand side of box 23. To correct such mistakes, UDK #12, "DELETE LAST LINE" is pressed.

```
from 6     side R
to   7     side L
type 1
from 6     side B
to   14    side T
type 1
from 7     side B
to   11    side T
type 1
from 3     side R
to   4     side L
type 1
from 3     side B
to   7     side T
type 1
from 4     side B
to   8     side T
type 1
from 14    side R
to   15    side L
type 1
from 15    side B
to   19    side T
type 1
from 19    side B
to   23    side T
type 1
from 14    side B
to   23    side R
type 1
```

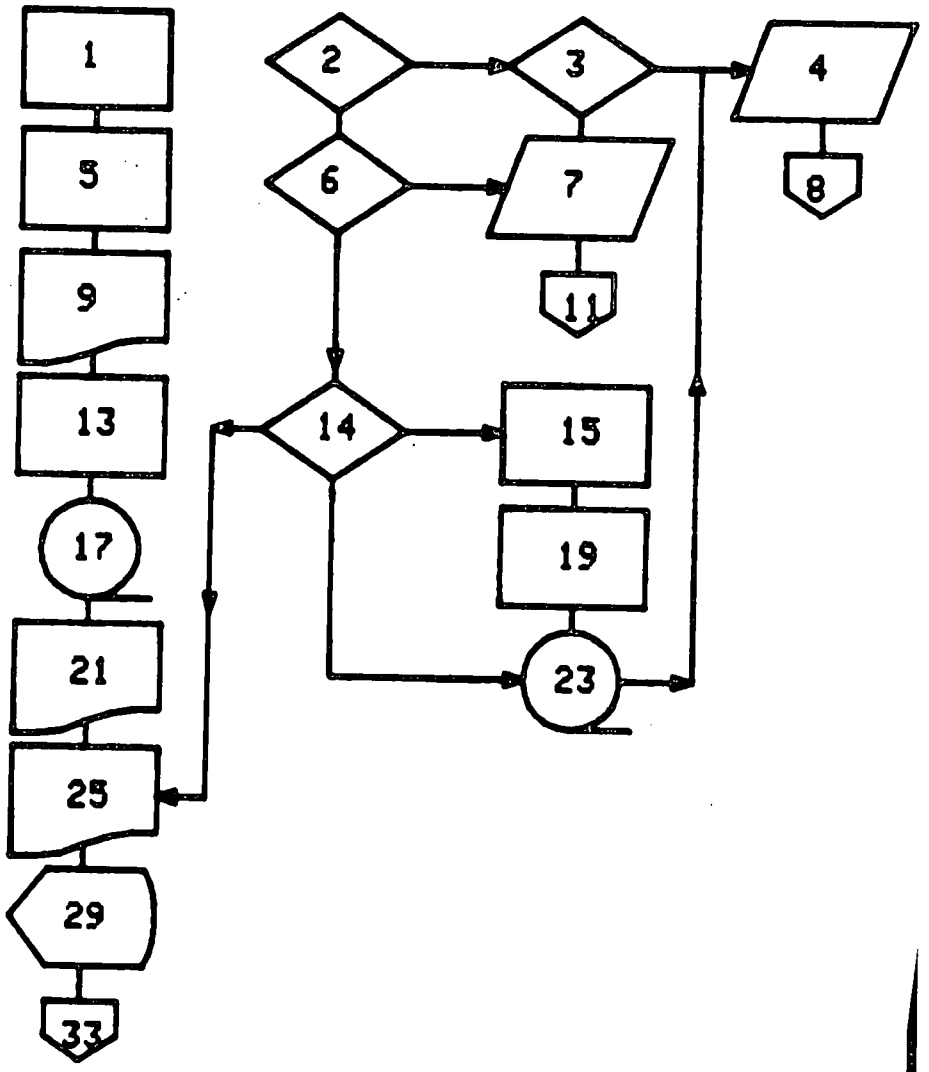


TITLE

FLOW DIAGRAMMER

Shown below is the completely connected diagram.

```
from 14   side B
to   23   side L
type 1
from 23   side R
to   4    side L
type 2
from 14   side L
to   25   side R
type 2
from
```



TITLE

FLOW DIAGRAMMER

The user now has two choices; to plot the diagram or to load phase 4, the data entry phase. It is prefereable at this point to press UDK 16, then UDK 15. This will cause the diagram to be output to the 4662 Plotter as shown on page 119.

TITLE

FLOW DIAGRAMMER

Now load phase 4 by pressing UDK 19. When phase 4 is loaded, the following menu appears:

```
Enter:
 1) to enter heading
 2) to enter box data
 3) to plot box data
```

First, enter '1' and press 'return'. The message 'Enter heading' appears. Enter a heading of 50 or fewer characters and press 'return'.

```
Enter:
 1) to enter heading
 2) to enter box data
 3) to plot box data
1

Enter heading: FLOW DIAGRAMMER: PHASE 1
```

The menu will return to the screen, now enter '2' and press 'return'.

```
Enter:
 1) to enter heading
 2) to enter box data
 3) to plot box data
2
```

TITLE

FLOW DIAGRAMMER

Then enter a box number, up to four lines of information to be entered into the box (up to 20 characters per line) and information, if any, to appear outside the box (up to 8 characters per line.) Press 'return' to omit a line. Automatic centering of in-box data will take place. Do not add extraneous spaces after a line since they will be counted as characters in the centering algorithm. If it necessary to enter data on, say, lines 1, 3 and 4, enter a space in line 2 before pressing 'return'.

Examples of data entry for boxes 1 and 14 are shown below:

```
box number: 1

Enter line 1: INITIALIZE
      line 2: & SET KEYS
      line 3:
      line 4:

      above box: 1
      on right connector:
      on bottom connector:
      on left connector:


box number: 14

Enter line 1: TEST
      line 2: 'K'
      line 3:
      line 4:

      above box:
      on right connector: =2
      on bottom connector: =3
      on left connector: other
```

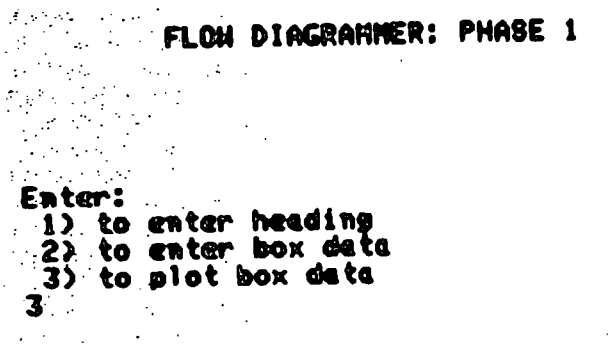
TITLE

FLOW DIAGRAMMER

After entering all box data, select menu item 3 of phase 4, 'plot box data', and the heading and annotations will be output to the plotter. (See page 19)

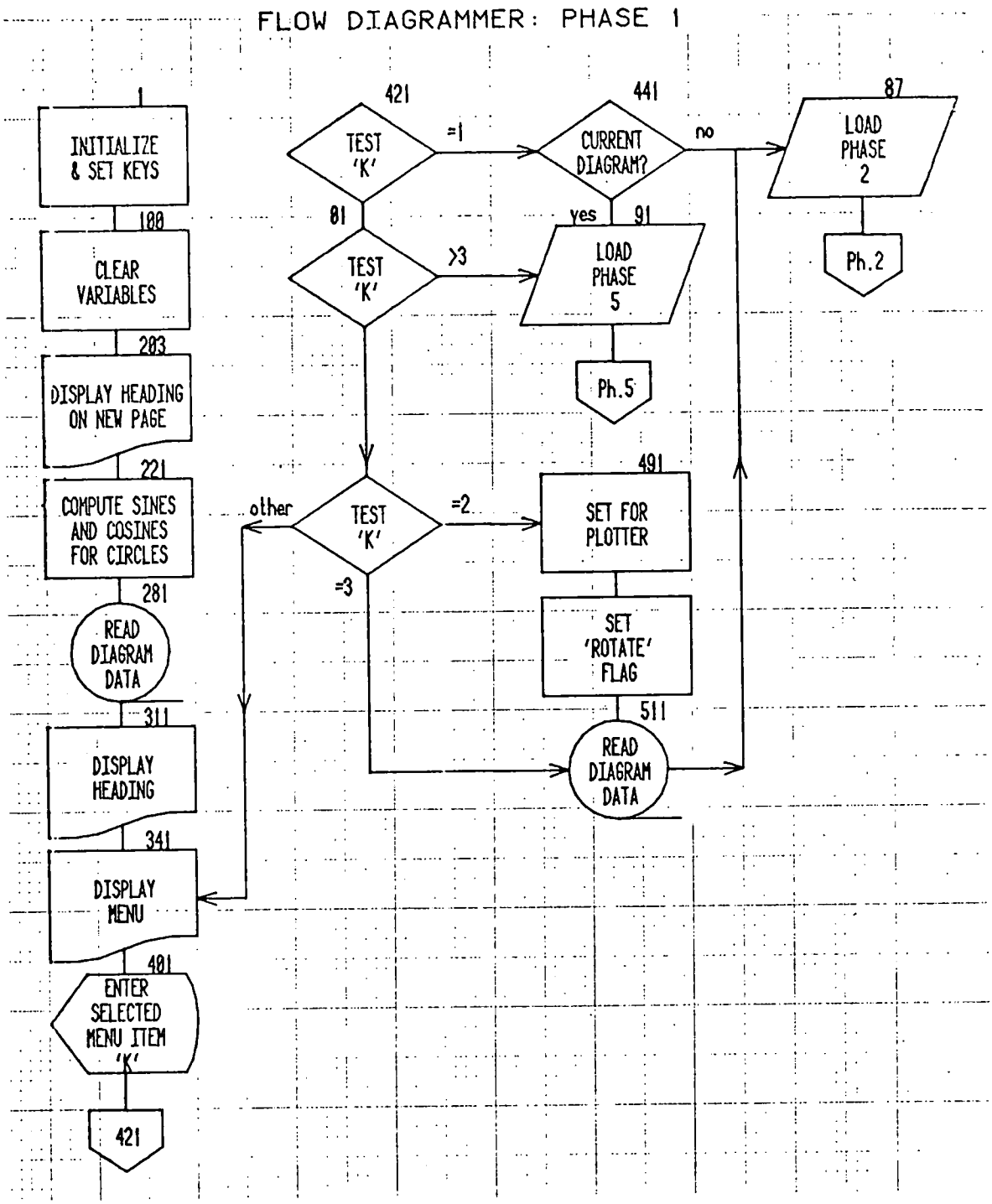
Phase 1 will then be reloaded and the main menu will be displayed. Corrections may be made at this point.

Note: Be sure the plotter has paper.



TITLE

FLOW DIAGRAMMER



TITLE

FLOW DIAGRAMMER

After the diagram is finished, phase 1 is reloaded and the main menu reappears:-

```
                    F L O W   D I A G R A M M E R


Current diagram is FLOW DIAGRAMMER: PHASE 1


Enter:

  1 to start a new diagram
  2 to redraw current diagram
  3 to change current diagram
  4 to retrieve a stored diagram
  5 to destroy part or all of stored data:
```

The user may select any item; item 2 will cause another copy of the current diagram to be drawn, item 3 will permit correction of any errors, (the current diagram will reappear on the screen and phase 3 will be active,) or item 1 will be selected if a new diagram is to be drawn.

TITLE

FLOW DIAGRAMMER

Selecting item 4 will generate the following:-

```
Current diagram: FLOW DIAGRAMMER: PHASE 1

Stored diagrams:
 1 FLOW DIAGRAMMER: PHASE 1
 2 ROTATED LINE ROUTINE
 3 DRAW BOXES FROM FUNCTION KEYS

WARNING--The current diagram will be destroyed.
Press 'return' to proceed or key 18 to save current diagram.
```

and after the user presses 'return' or UDK 18 the main menu will be redisplayed. If item 4 is selected when no current diagram exists, the message is:-

```
No current diagram

Stored diagrams:
 1 FLOW DIAGRAMMER: PHASE 1
 2 ROTATED LINE ROUTINE
 3 DRAW BOXES FROM FUNCTION KEYS

Enter number of diagram to retrieve:
```

Item 5 is an acceptable choice at this time, since its selection has no effect on the current program.

TITLE

FLOW DIAGRAMMER

If item 1 is selected while a current diagram is active, phase 5 is loaded and the following message appears:-

```
Current diagram: FLOW DIAGRAMMER: PHASE 1
Stored diagrams:
...none
Enter 1 to store current diagram,0 to destroy:
```

The user enters '1' and presses 'return' to store the current diagram. The 37 files used to store the heading, lines and box data for the current diagram are read and consolidated into a single long file. If the box data, including separators, (\ and ~, which must not be used in diagrams,) exceeds 1000 characters, the message

```
Diagram too big to store
STOP IN LINE 641 PRIOR TO LINE 651
```

is displayed. To destroy the diagram and return to the main menu, enter 'RUN 651' and press 'return'.

TITLE

FLOW DIAGRAMMER

If the diagram is not too big, the message

```
Enter 0 to save on available space,
Enter number to replace stored diagram:
```

is displayed. Normally, the user will enter '0' and press 'return'. However, if the current diagram was previously retrieved for modification, it is preferable to return it to the file from which it was retrieved, in order to destroy the previous version of the diagram.

If item 5 is selected, the user intends to destroy one or more stored diagrams. This need only be done when twenty diagrams (the maximum) are stored and a current diagram must be saved. The message

```
No current diagram

Stored diagrams:
 1 FLOW DIAGRAMMER: PHASE 1
 2 ROTATED LINE ROUTINE
 3 DRAW BOXES FROM FUNCTION KEYS

Enter 0 to retain all stored diagrams,
      diagram number to delete a specific diagram,
      or 99 to destroy all stored diagrams:
```

appears. After the user selects the appropriate action, the main menu will be redisplayed.

TITLE

FLOW DIAGRAMMER

Data Tape Structure:

File 1	ASCII Program	3000
Files 2-5	ASCII Program	4800 each
File 6	Binary Data	1024
Files 7-42	Binary Data	768 each
Files 43-62	Binary Data	3072 ea

Internal Data Storage:

Variable	Used to Store...	Type
A,B,C,D	Endpoints of lines	Simple
C5	Cosines	Array (25)
H$	Headings	String (50)
K	Last function key	Simple
L$	Diagram	Array (1000)
M	Device address	Simple
R	Rotate flag	Simple
S	Dotted line flag	Simple
S5	Sines	Array (25)
T	Box types	Array (4,10)
Z	Connectors	Array (50)
Z1	Number of connectors	Simple

TITLE

FLOW DIAGRAMMER

Appendix A

Flow Diagramming Matrix Form

TITLE

FLOW DIAGRAMMER

FLOW DIAGRAMMING MATRIX

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32
33	34	35	36

TITLE

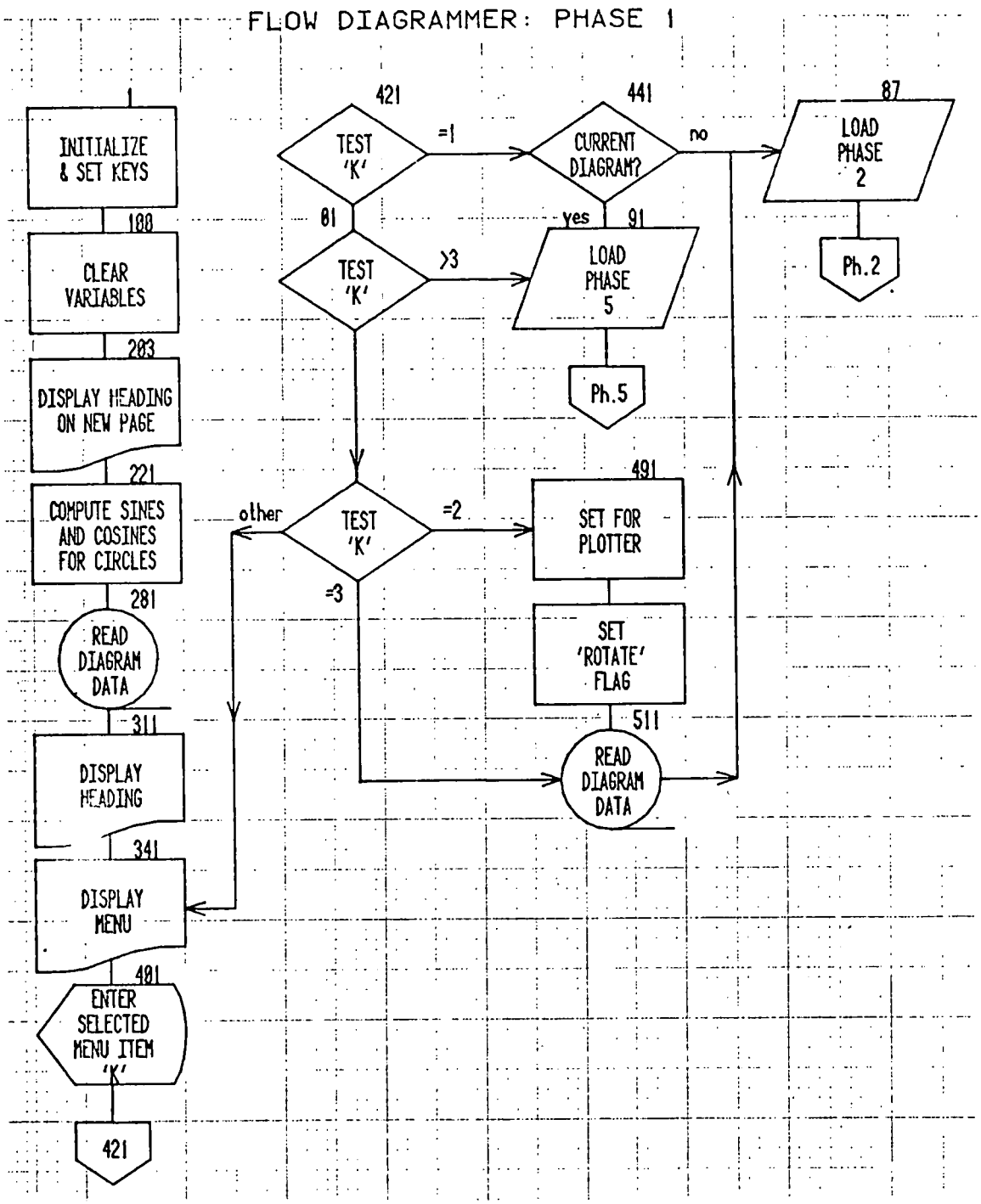
FLOW DIAGRAMMER

Appendix B

Flow Diagrams

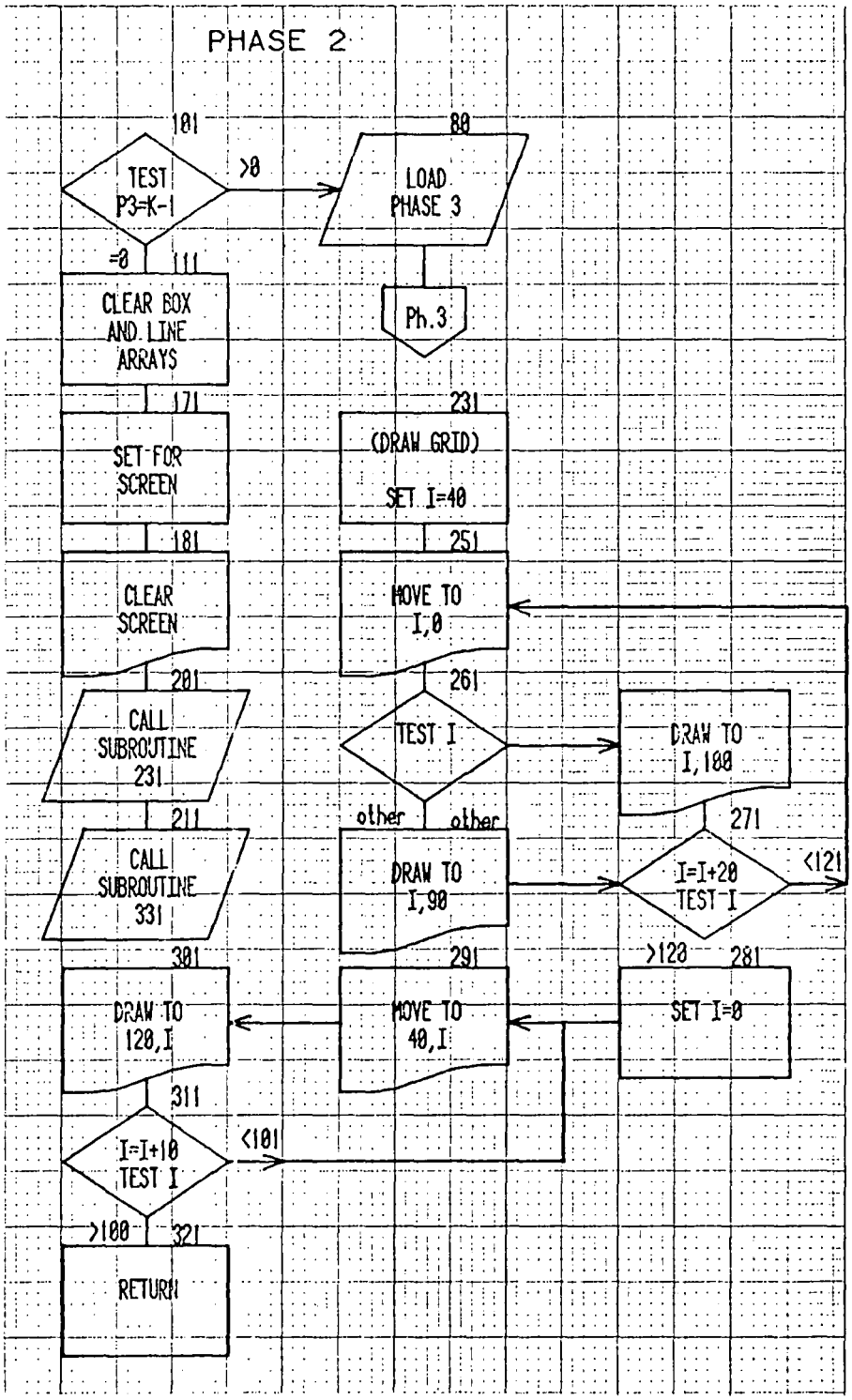
TITLE

FLOW DIAGRAMMER



TITLE

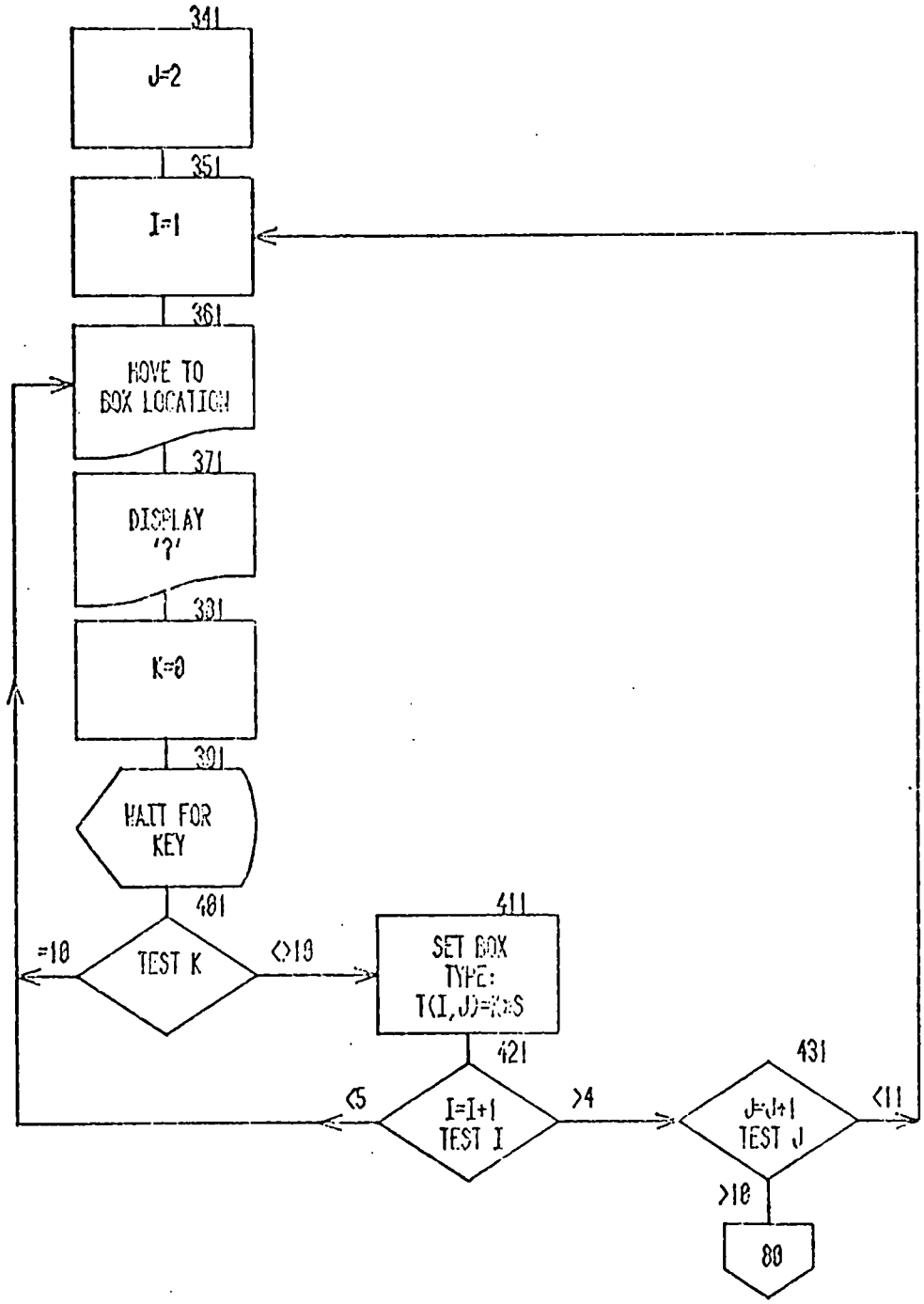
FLOW DIAGRAMMER



TITLE

FLOW DIAGRAMMER

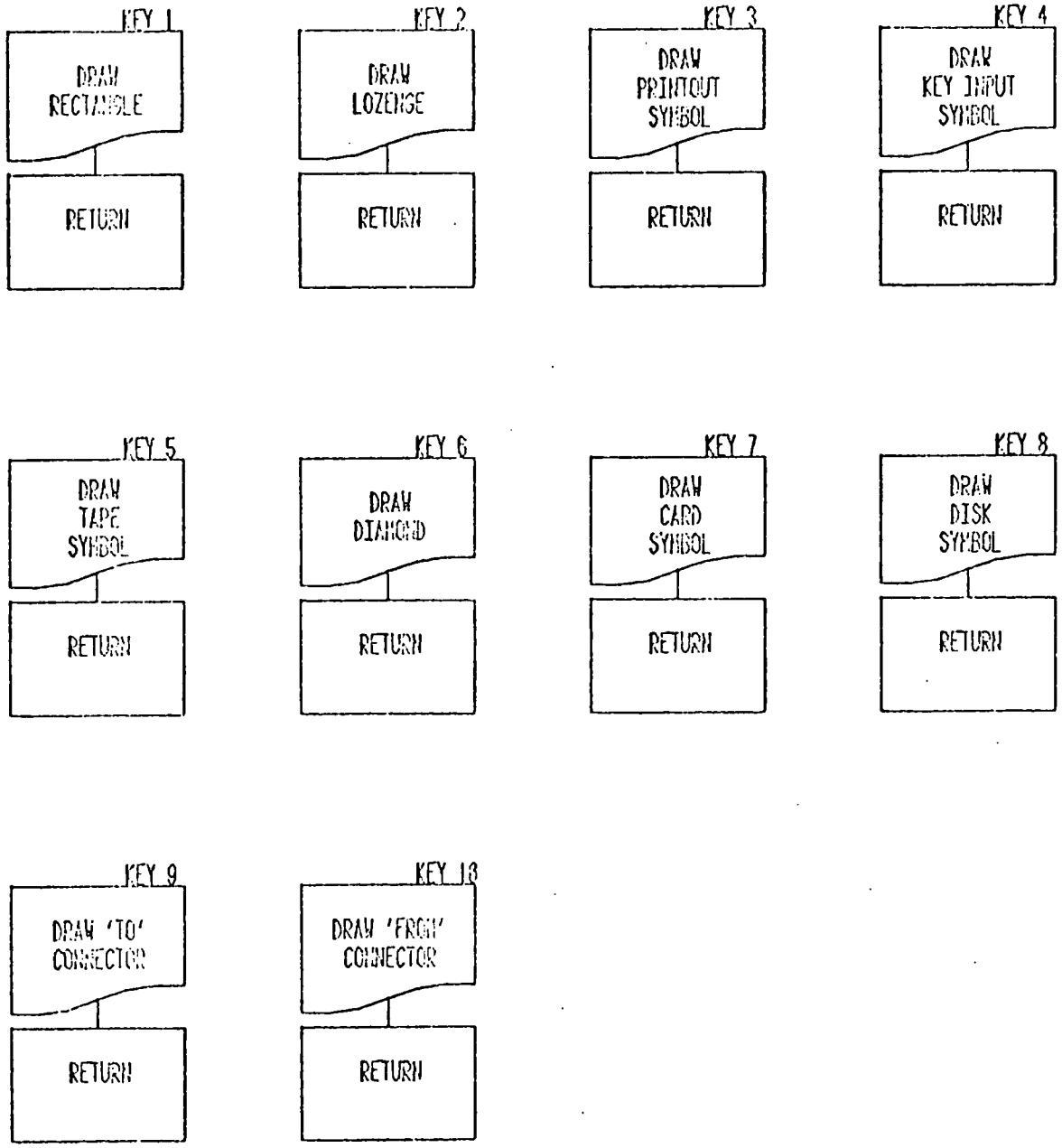
SELECT BOXES FROM FUNCTION KEYS



TITLE

FLOW DIAGRAMMER

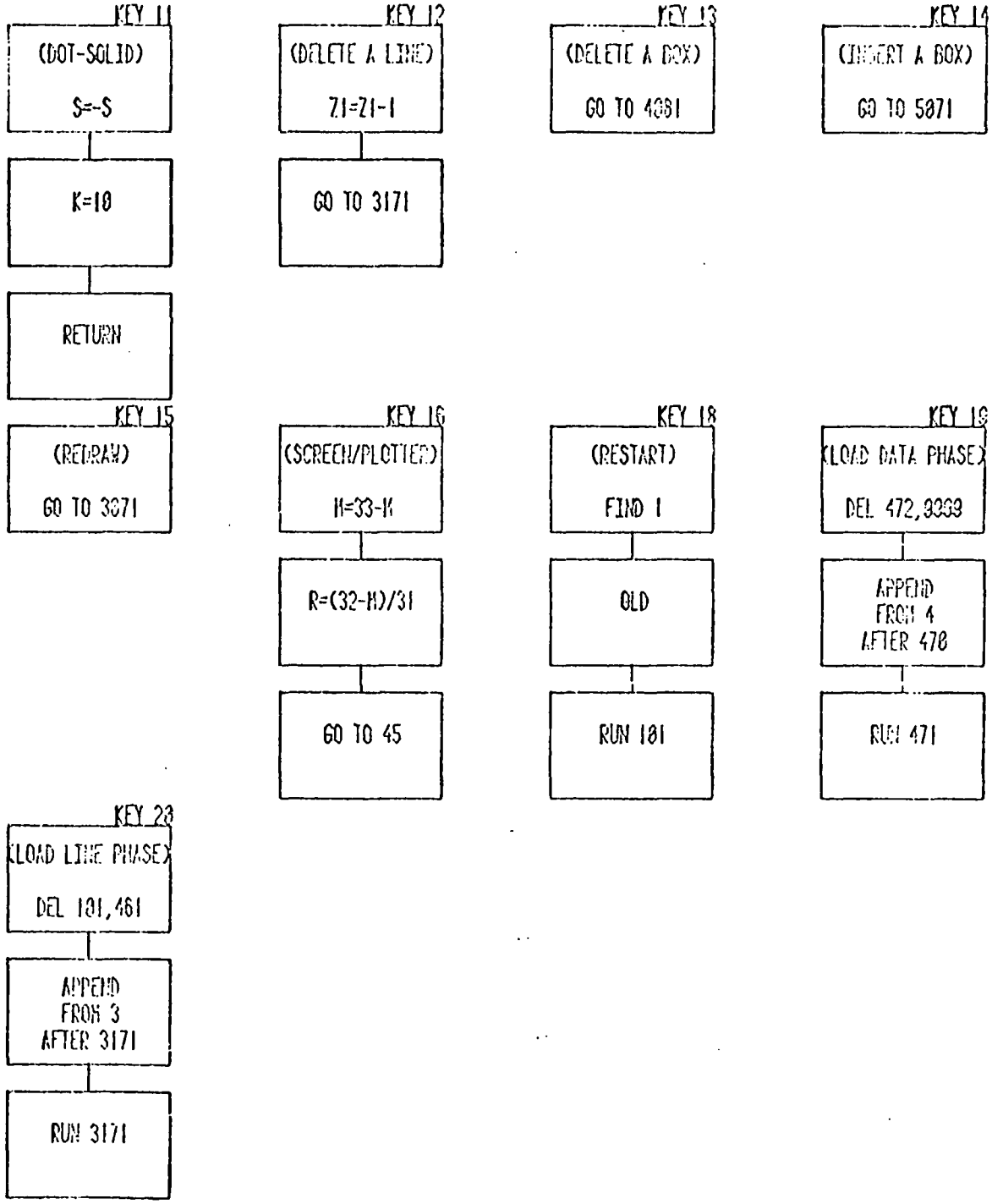
DRAW BOXES FROM FUNCTION KEYS 1-10



TITLE

FLOW DIAGRAMMER

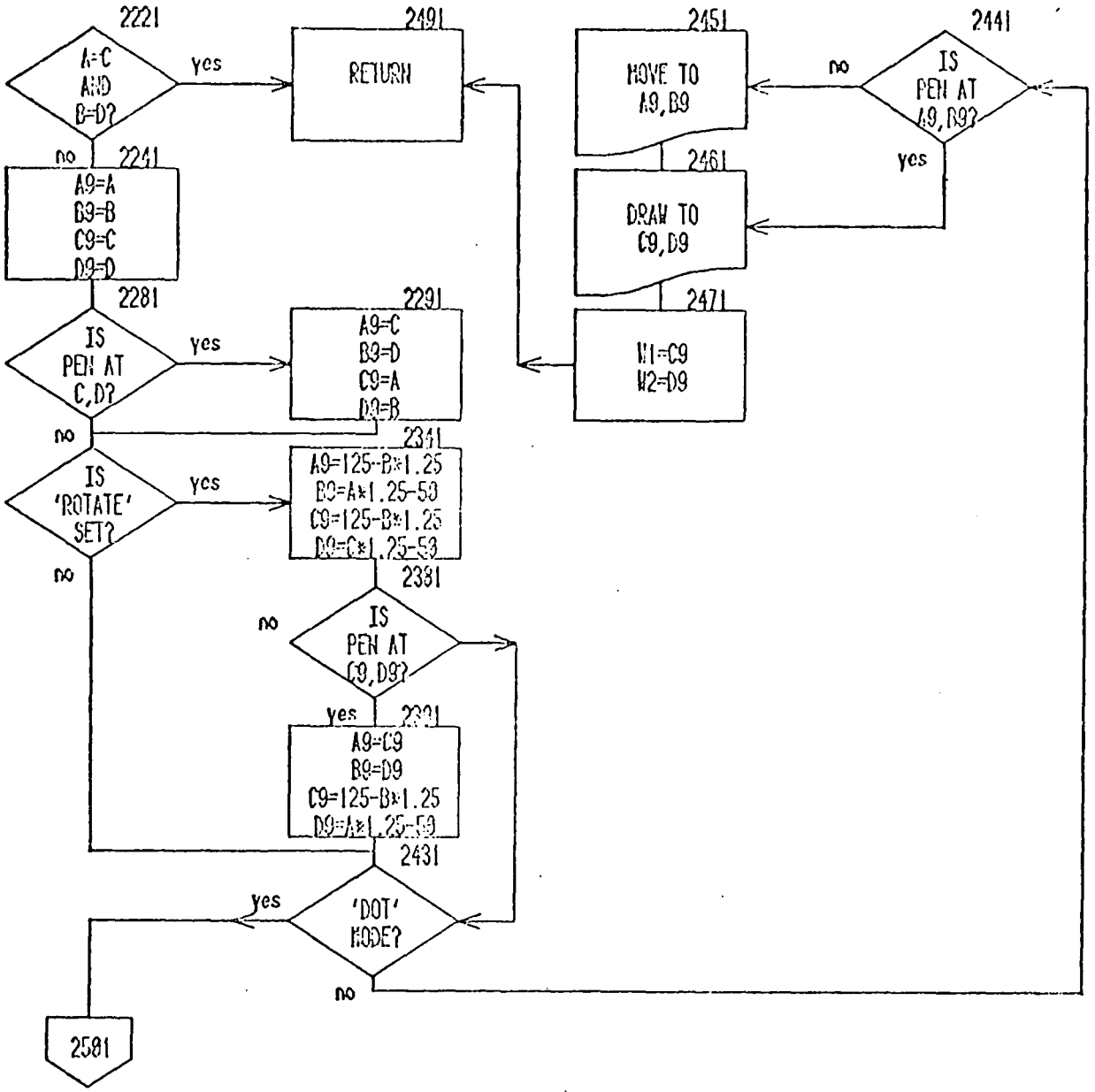
FUNCTION KEYS 11-20



TITLE

FLOW DIAGRAMMER

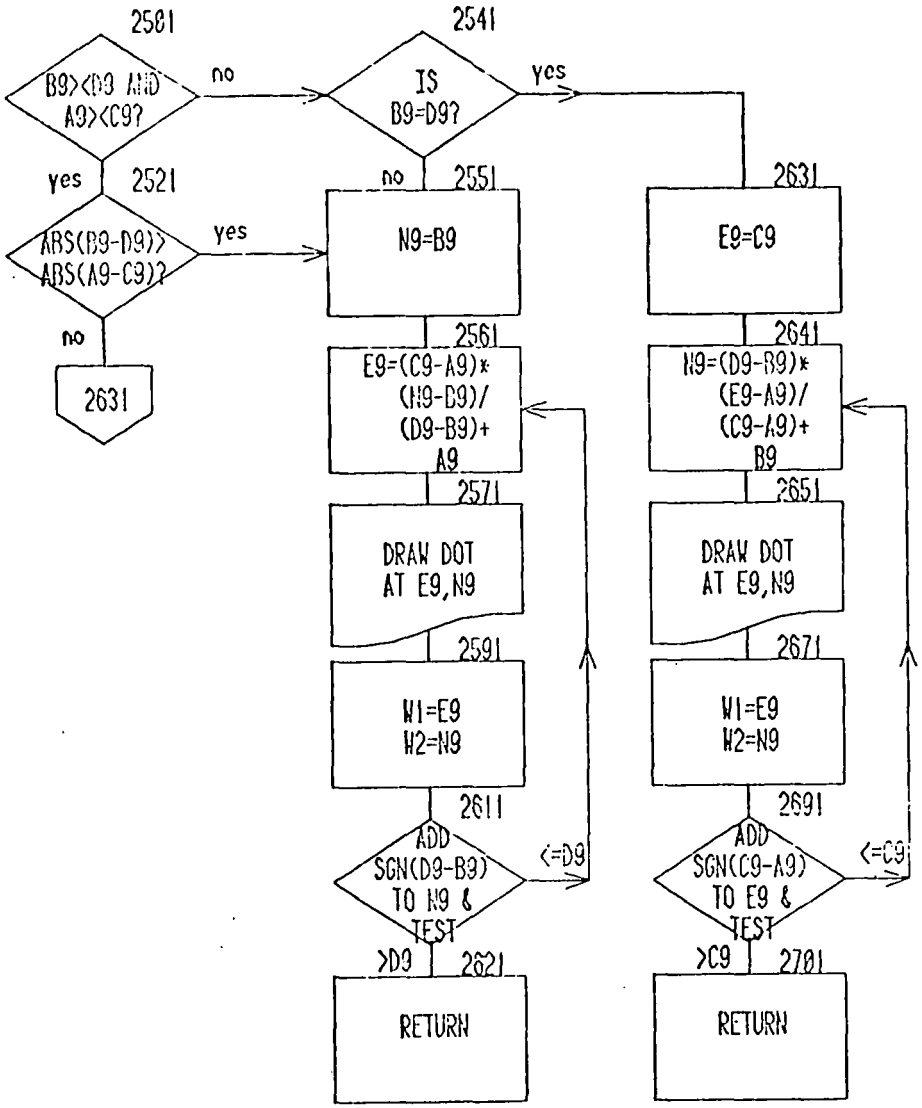
ROTATED LINE ROUTINE



TITLE

FLOW DIAGRAMMER

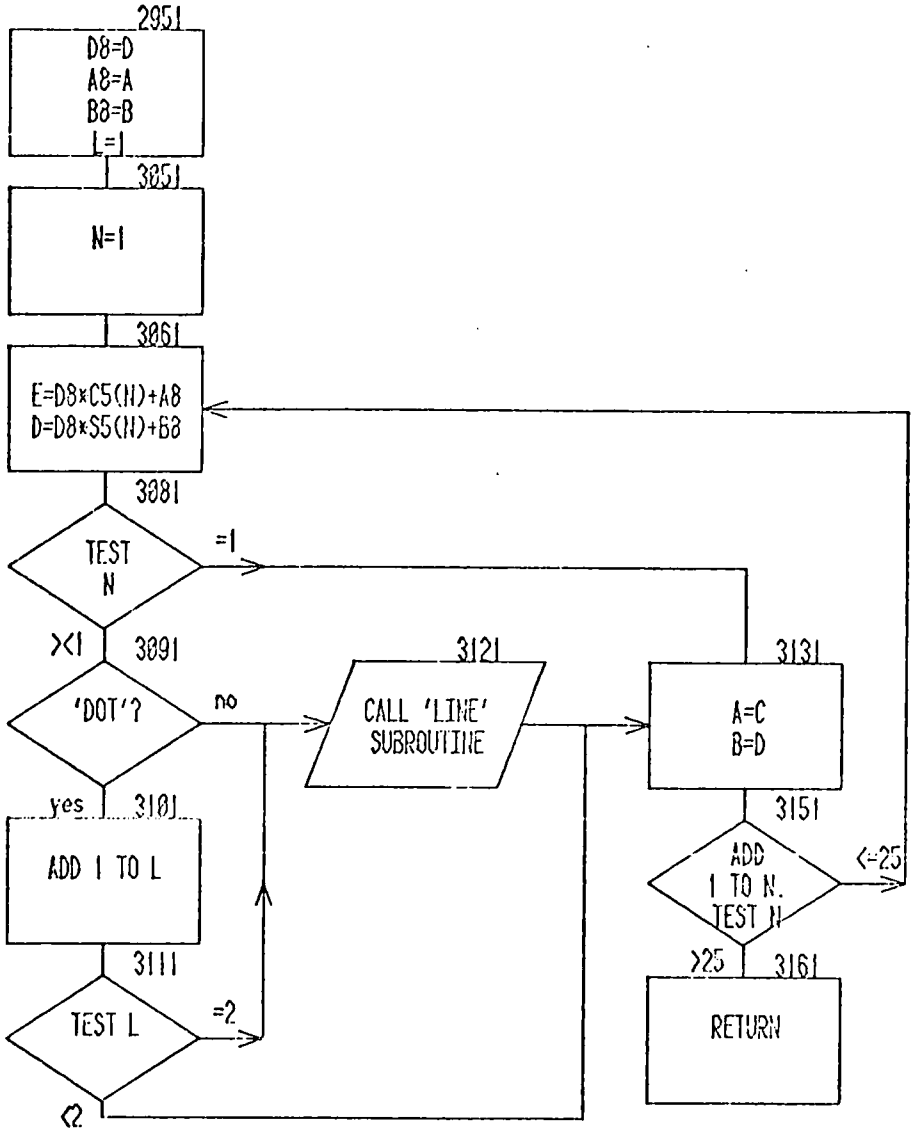
Rotated Line Routine (Continued)



TITLE

FLOW DIAGRAMMER

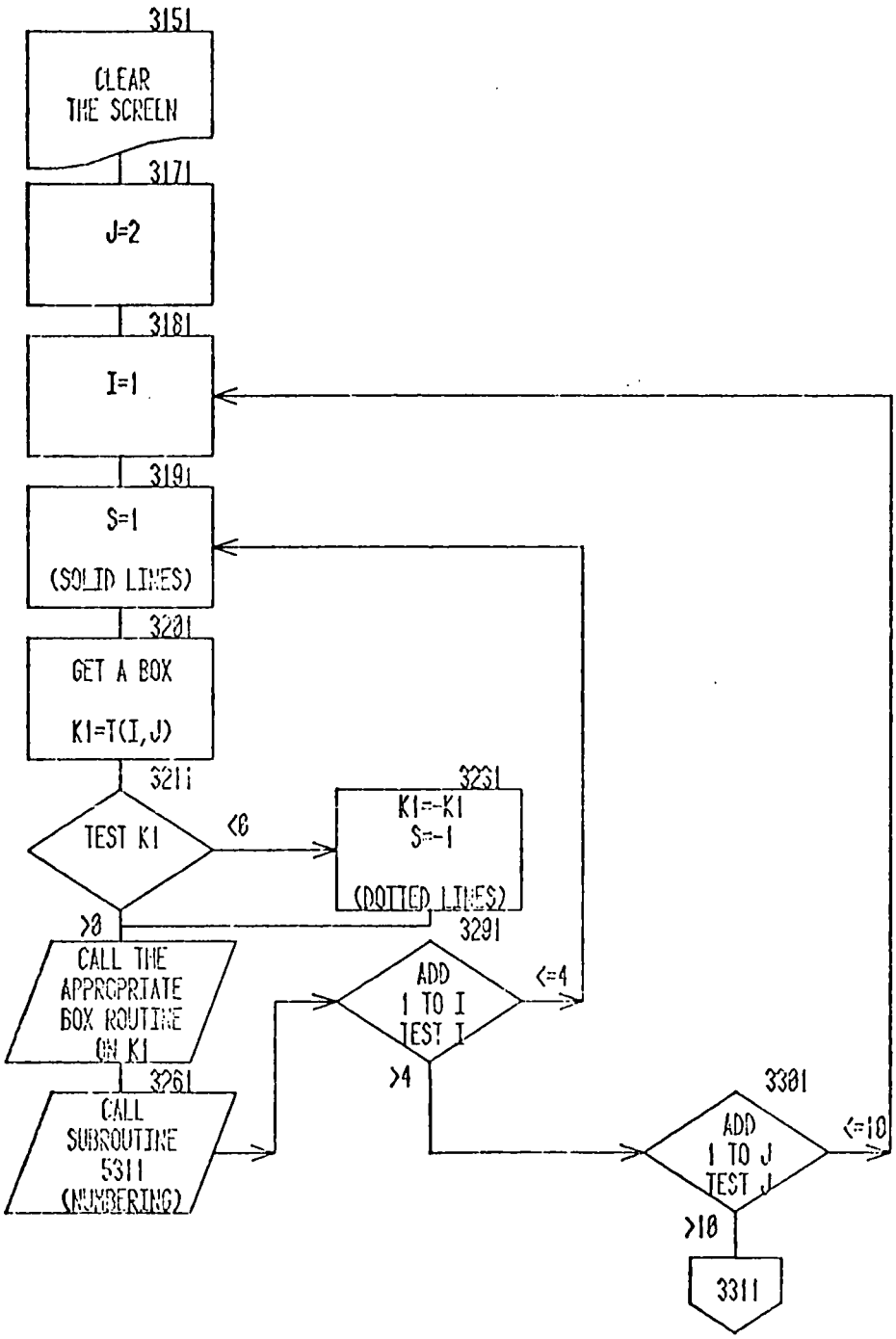
CIRCLE ROUTINE



TITLE

FLOW DIAGRAMMER

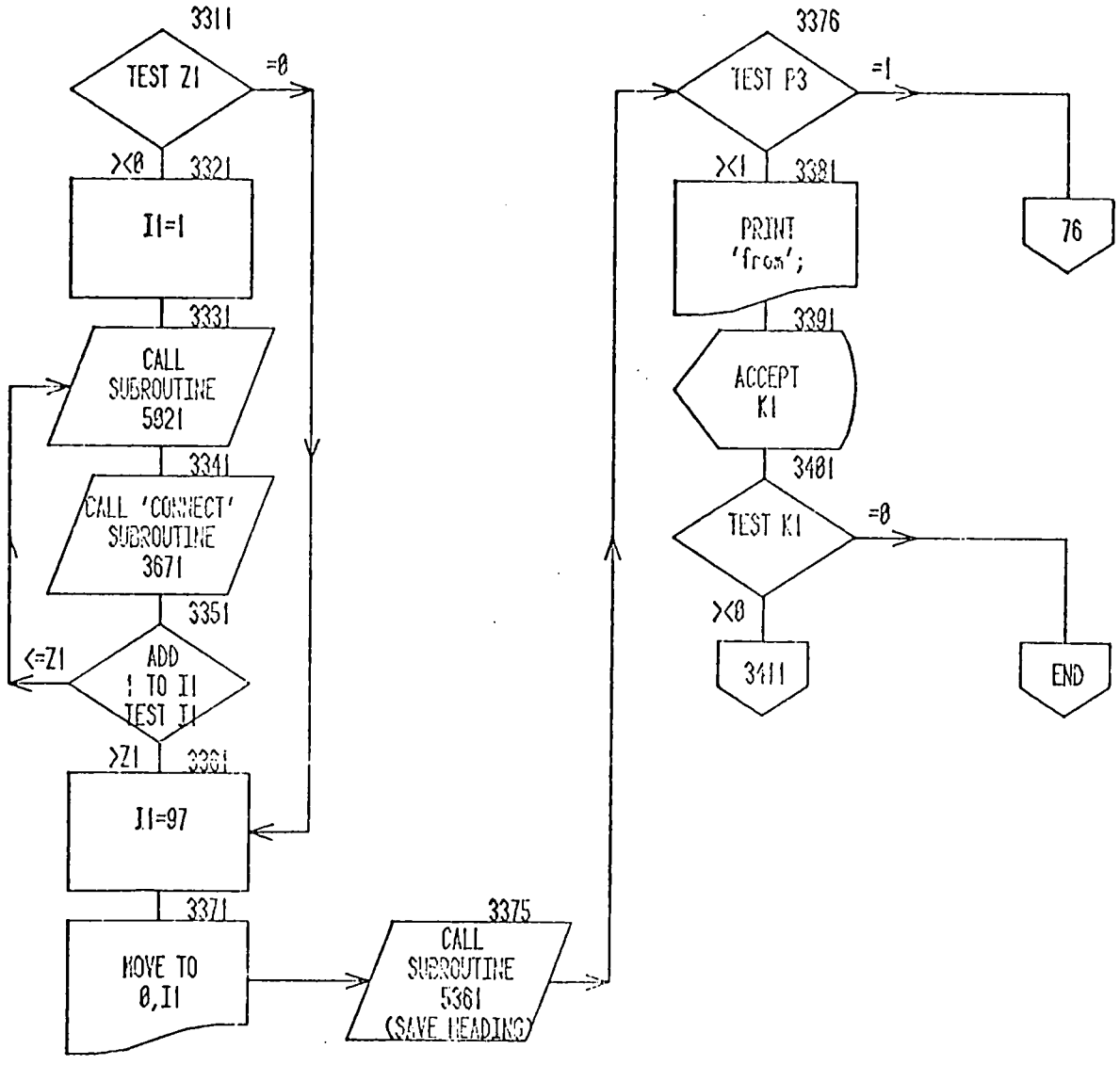
PHASE 3: DISPLAY BOXES



TITLE

FLOW DIAGRAMMER

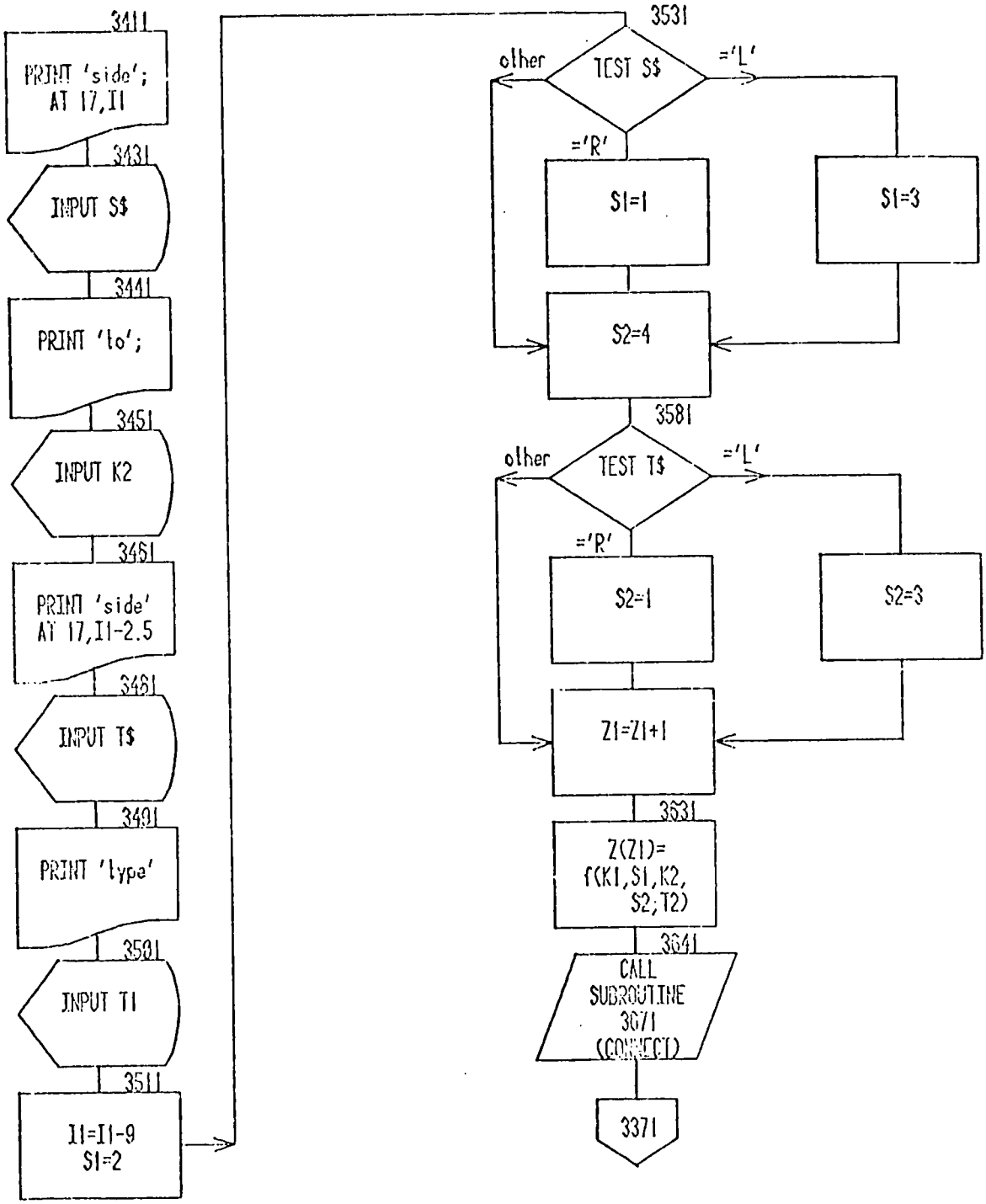
CONNECT BOXES



TITLE

FLOW DIAGRAMMER

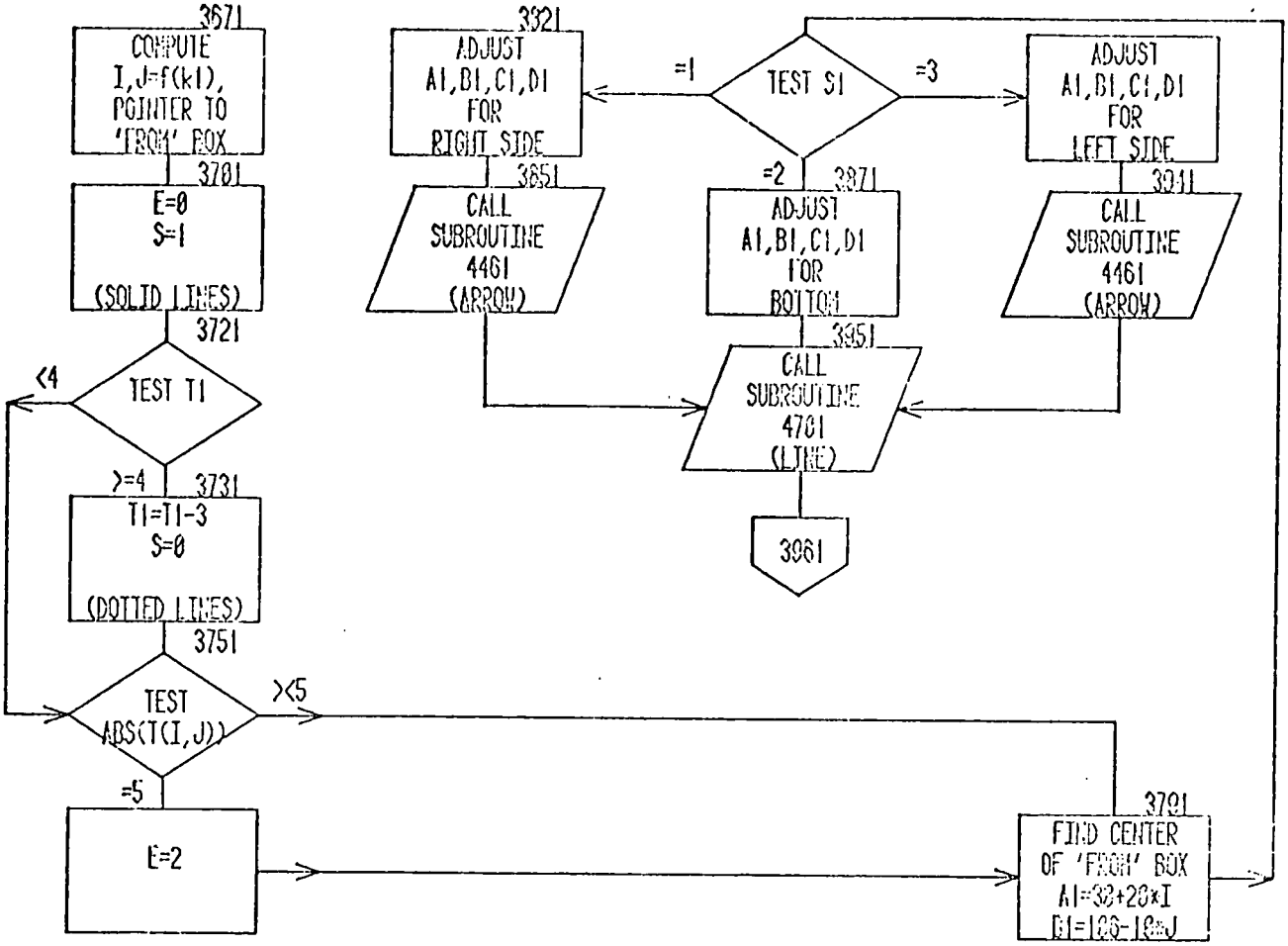
Connect Boxes (Continued)



TITLE

FLOW DIAGRAMMER

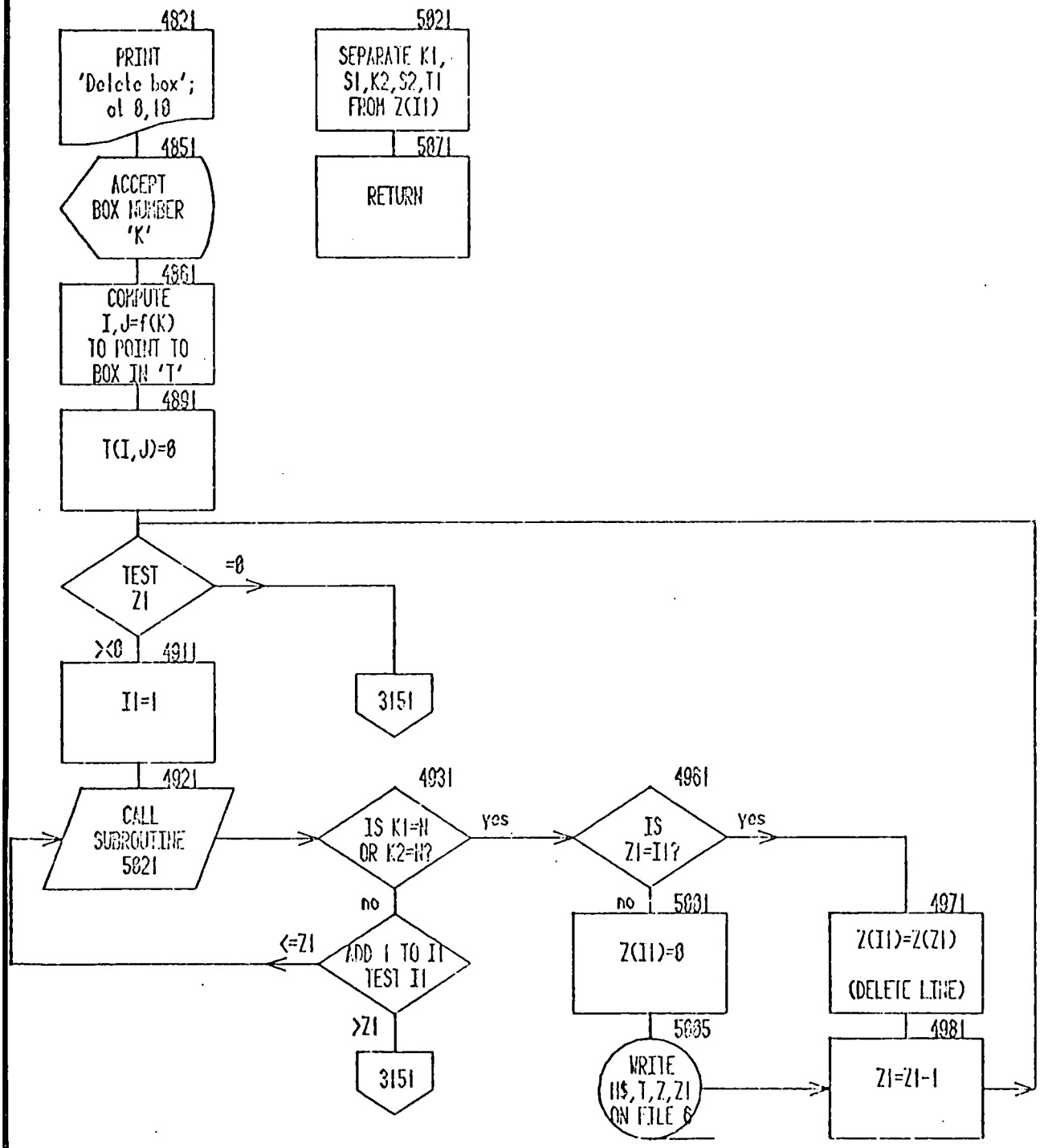
DRAW CONNECTING LINES



TITLE

FLOW DIAGRAMMER

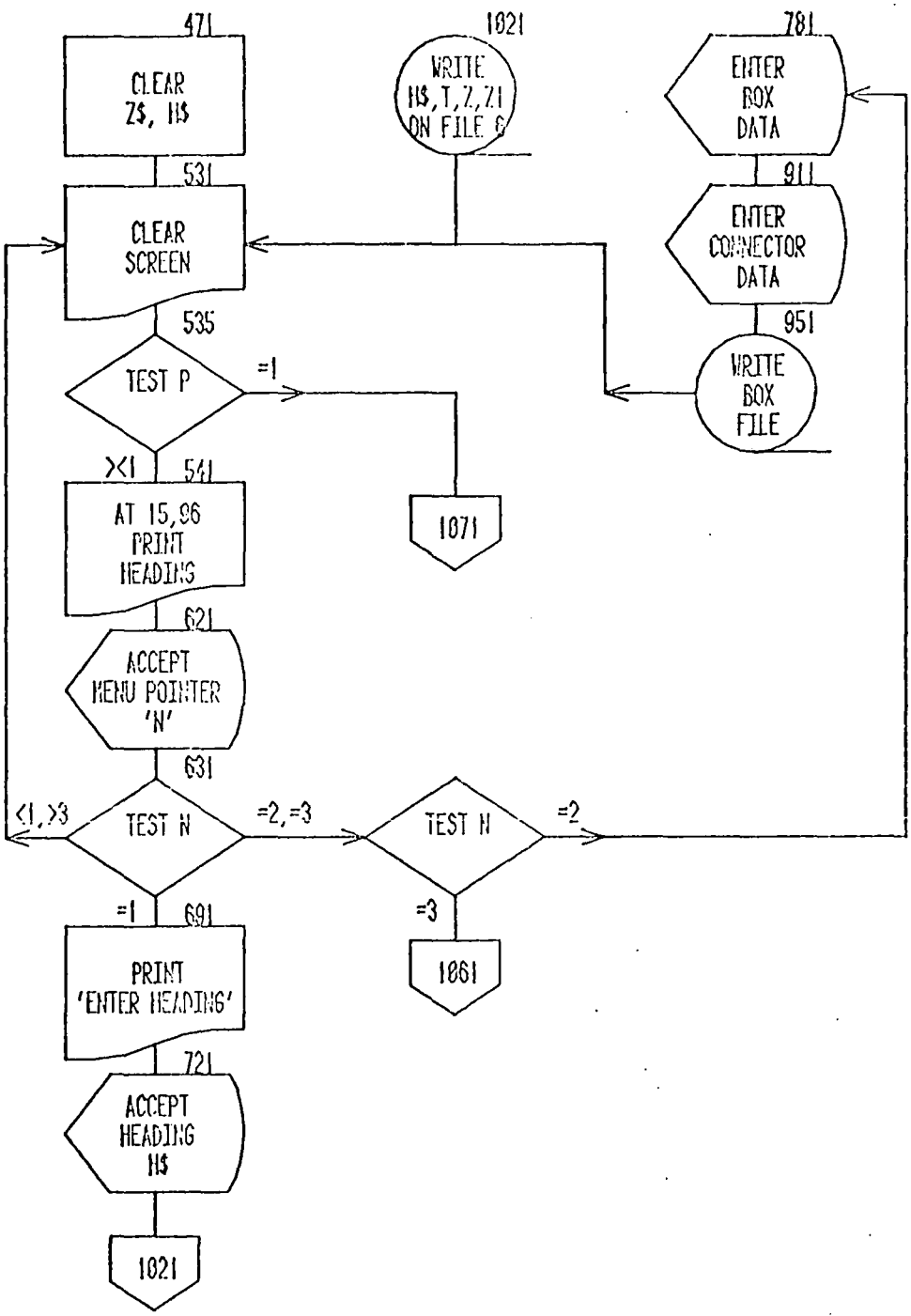
DELETE A BOX



TITLE

FLOW DIAGRAMMER

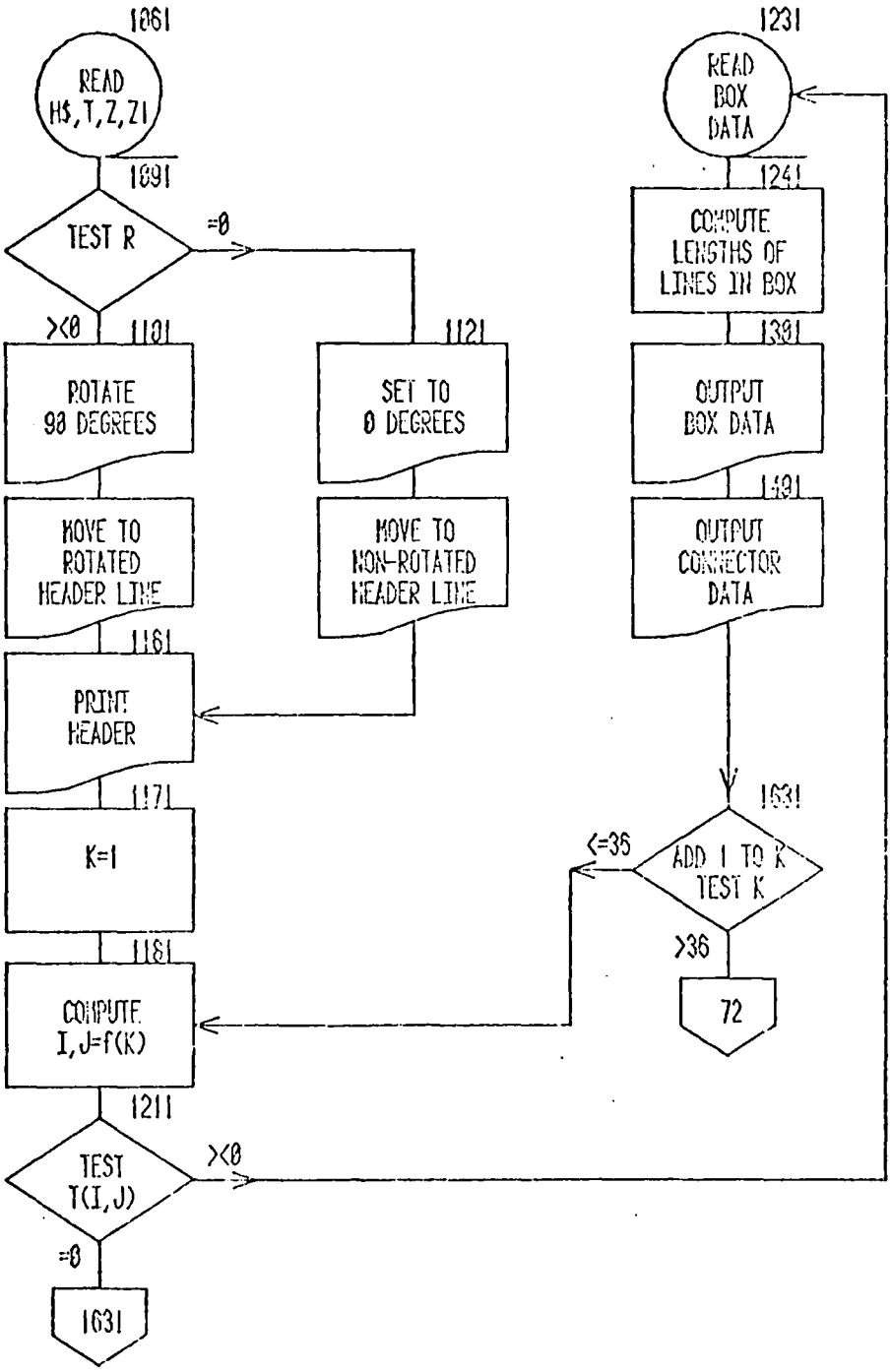
PHASE 4



TITLE

FLOW DIAGRAMMER

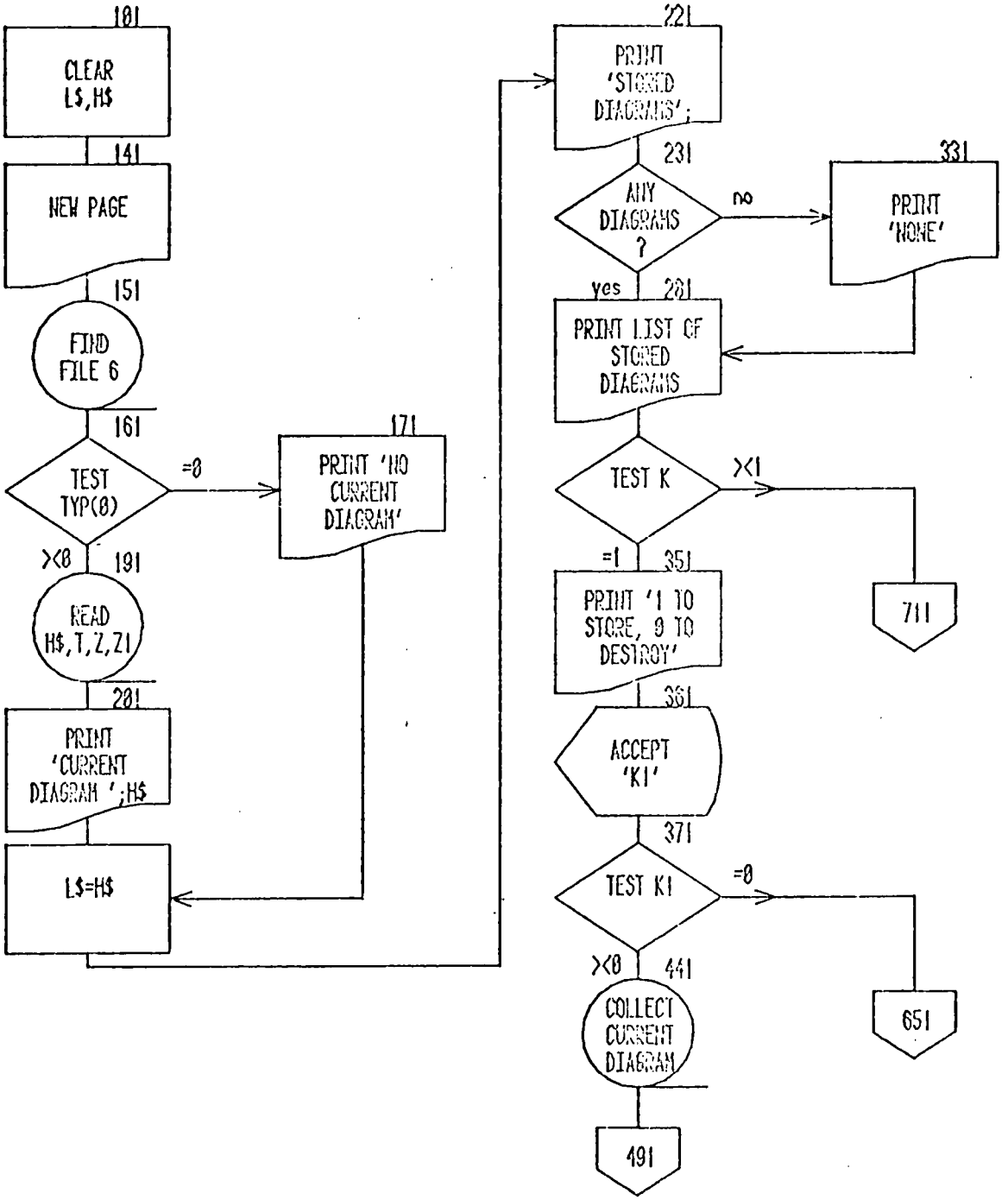
PLOT BOX DATA



TITLE

FLOW DIAGRAMMER

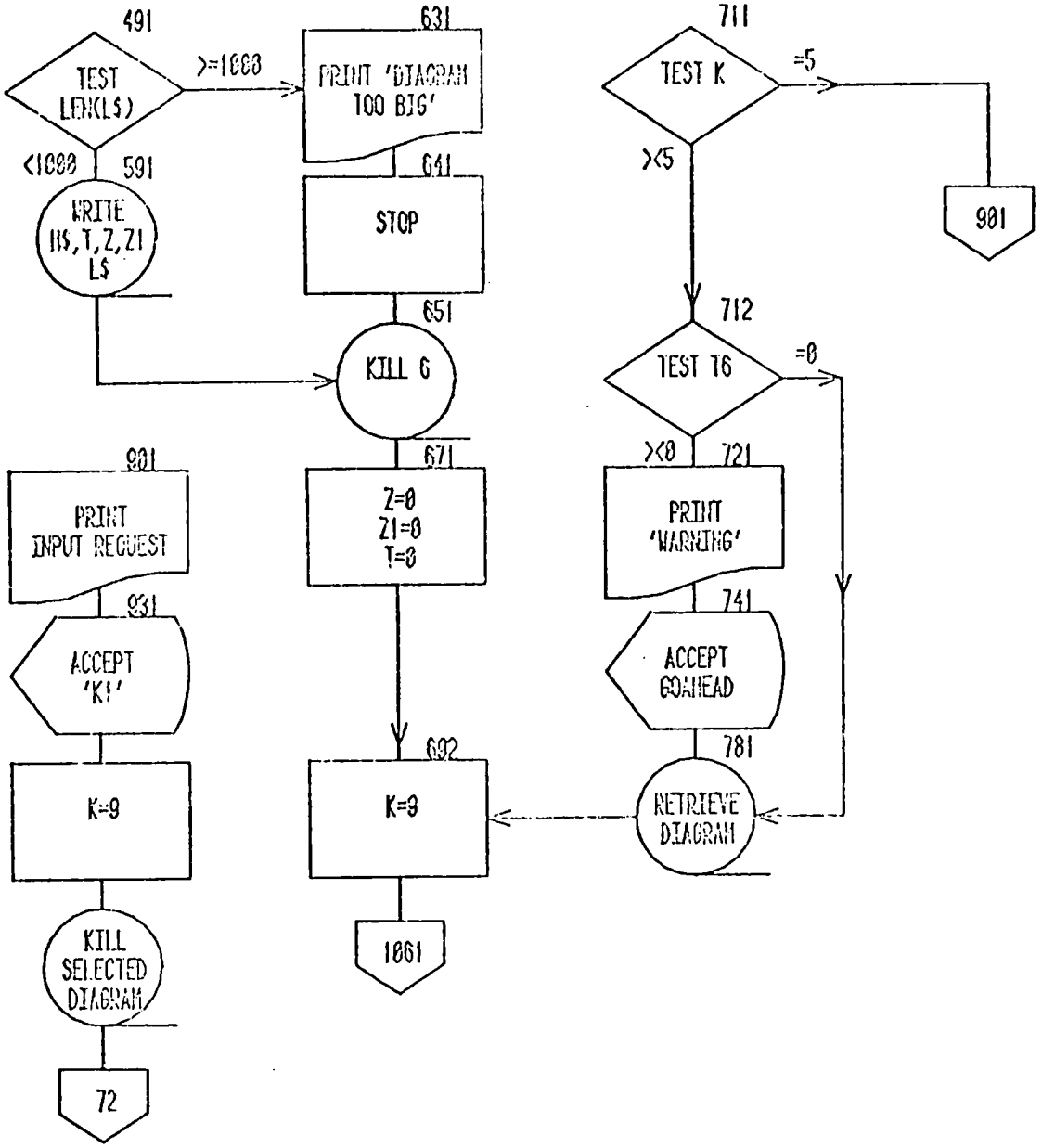
PHASE 5



TITLE

FLOW DIAGRAMMER

Phase 5 (Continued)



Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE

Flow Diagrammer (Disk)

ORIGINAL DATE	REVISION DATE
November, 1977	November, 1980

AUTHOR Keith S, Reid-Green
Education Testin Service
Princeton, NJ

EQUIPMENT AND OPTIONS REQUIRED

32K

PERIPHERALS

4907 File Manager

4662 Plotter

ABSTRACT

Revised by: Leland C. Sheppard
Sheppard Software Co.
Sunnyvale, CA

Files: 1 Program
Requires Dedicated Data Disk

Statements: 923

The program is used to design, store, recall and modify flow diagrams for use in program and system documentation.

A diagram consists of a heading, 10 different box types, their connecting lines and labeling. Boxes and lines may be solid or dotted and may be arranged up to four across and nine deep on a page.

The program consists of six phases:

1. Create Data Disk
2. Main Menu
3. Enter Boxes
4. Connect, Insert, Delete Boxes
5. Enter Box Data and Heading
6. Store/Retrieve, Destroy Diagrams

A disk will hold 200 diagrams with up to 2000 characters of "box data" per diagram.

TITLE

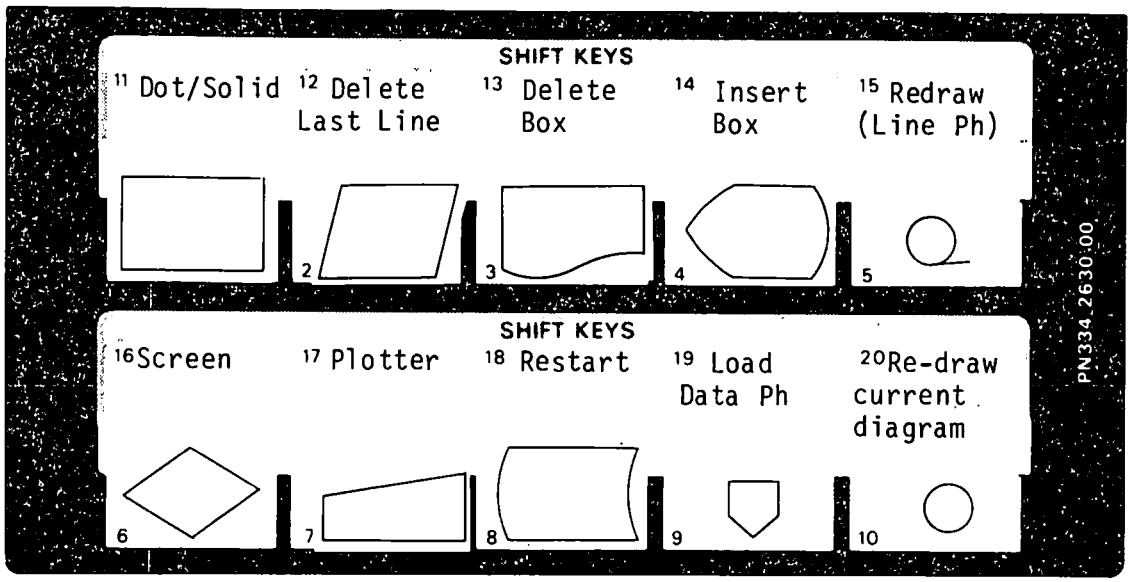
Flow Diagrammer

TABLE OF CONTENTS

TITLE

Flow Diagrammer

TITLE TAPE # FILE #



TITLE

Flow Diagrammer

DESCRIPTION

The program consists of six phases:

1. Creation of data disk
2. Main menu
3. Enter boxes
4. Connect, insert, delete boxes
5. Enter box data and heading
6. Store/retrieve/destroy diagrams

These phases are stored in one large program file.

Three files will occupy the data disk. NOTE: The program file cannot reside on the data disk; there isn't room.

When you begin a flow diagram, you should arrive at the 4050 with a rough flow diagram drawn on the diagramming matrix form.

The plotter address is set for 1. Change statement 68 if your plotter address is different.

FLOW DIAGRAMMING MATRIX

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32
33	34	35	36

TITLE

Flow Diagrammer

PHASE 1

CREATE DATA DISK

Load the program into 4050 memory.

Insert a disk into the 4907 File Manager (you may use any disk unit).

RUN the program.

The program will prompt you for the date and time if it hasn't been set on the 4907, and will format the disk and MOUNT it.

It will create and format the three binary data files. NOTE: If your disk has already been formatted, and data files created, it will bypass this stage.

Data File Structure

@FLOW/WORK

37 records, 874 bytes each
1st record: H$, T, Z, Z1
2-37: Z$

Contains the information on the current diagram.

@FLOW/DIRECTORY

200 records, 55 bytes each
all: H$

Contains the title of each stored diagram.

@FLOW/DIAGRAMS

200 records, 2824 bytes each
all: T, Z, Z1, L$

Contains the rest of the diagram data. 200 diagram capacity with 2000 characters of box data (maximum) per diagram.

H$ = Title String (51)
T = Box Types Array (4,10)
Z = Connectors Array (50)
Z1 = Number of connectors Simple
L$ = Diagram Array (2001)
Z$ - box data String (126)

TITLE

Flow Diagrammer

PHASE 2

MAIN MENU

Once your data disk has been initialized, you will be directed to the second phase:

```
There is no current diagram

Choices:

  0 to end program
  1 to start a new diagram
  2 to redraw current diagram
  3 to change current diagram
  4 to retrieve a stored diagram
  5 to destroy part or all of stored data
  6 to store current diagram
Enter:
```

If there is no current diagram, selecting menu items 2, 3 or 6 makes no sense. Selecting item 1 will initiate phase 3 (enter boxes), selecting 4 or 5 will initiate phase 6 (store or retrieve diagrams).

If a current diagram exists, selecting 1, 4, 5 or 6 will initiate phase 6, giving you a chance to save the current diagram if desired. Selecting 2 will cause the current diagram to be plotted (a piece of 11 x 16 inch paper must be in place on the plotter and the plotter prepared for action). Selecting 3 will initiate phase 4 (connect, insert, delete boxes).

TITLE

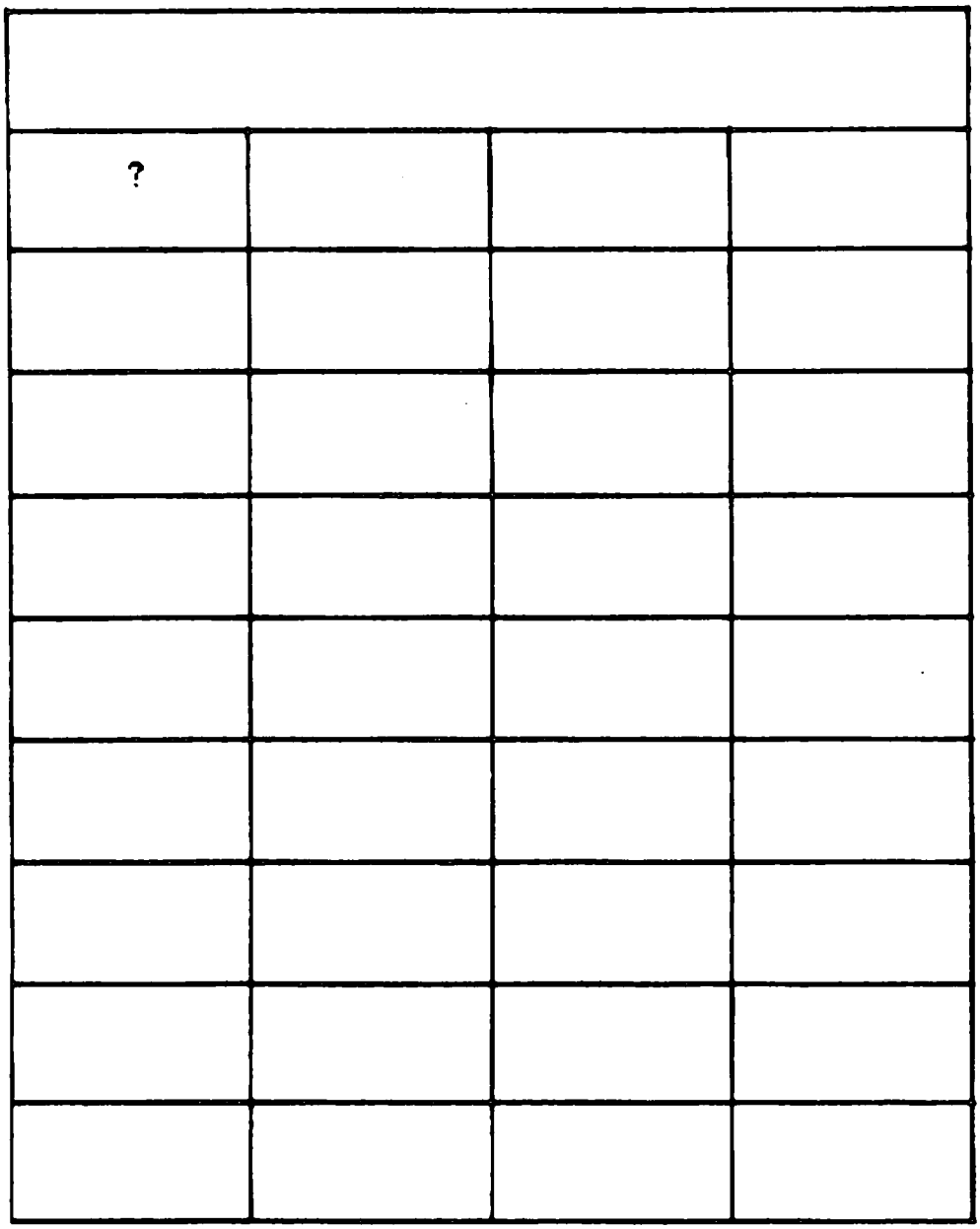
Flow Diagrammer

PHASE 3

ENTER BOXES (Starting a new diagram--no current diagram)

Select item 1 from the main menu. A diagram matrix will be drawn on the screen.

Refer to your rough diagram and input the boxes as prompted.



TITLE

Flow Diagrammer

In this phase (Phase 3), User-Definable Keys 1-11, 15-18 are meaningful.

UDKs 1-10 enter and plot your box selections.

UDK 11 changes the line style from solid to dotted and vice cersa.

UDK 17 will direct screen output to the plotter; however, it shouldn't be used in phase 3.

UDK 18 returns to phase 1 to restart. (All entered diagram data will be lost.)

UDK 15 enters phase 4 after completing the assignment of boxes to sections. (If the last section of the matrix is completed by assigning a box or pressing RETURN, phase 4 will automatically be loaded.)

UDKs 12-14 and 20 are not active during phase 3 and should not be pressed. Errors in the assignment of boxes must be corrected in phase 4.

TITLE

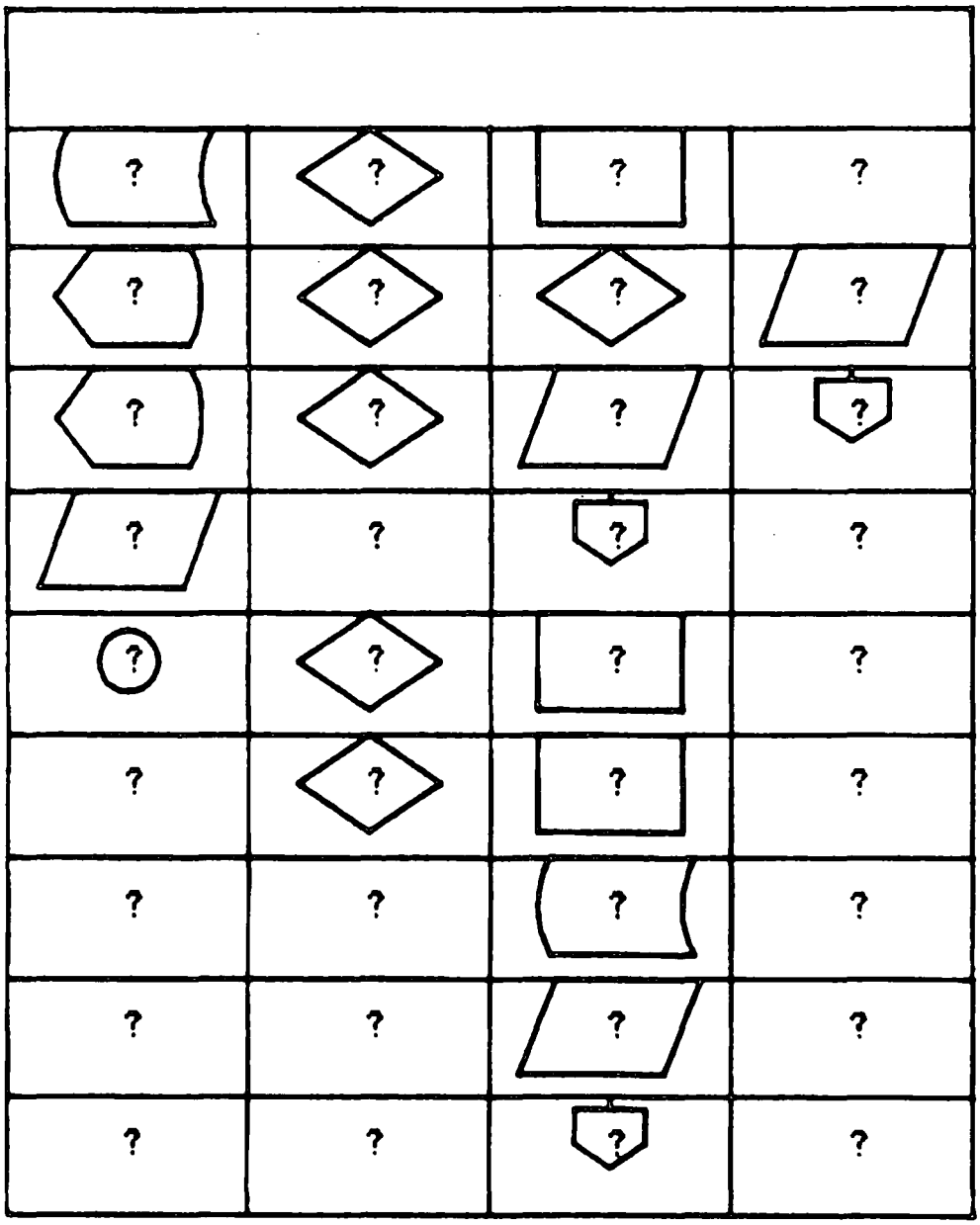
Flow Diagrammer

The question mark moves from section to section as you select box types by pressing User-Definable Keys 1-10. The question mark may be moved without selecting a box type by pressing RETURN.

NOTE: When you press a UDK, wait for the box to be drawn on the screen before pressing another UDK.

Shown below is the final assignment of boxes for the sample diagram. Boxes 1-3 arethe result of pressing UDK 3, 6, and 1. A RETURN was pressed to get to square 5 without entry in square 4. And so on, until the diagram was complete.

Since RETURN was pressed when the question mark was in the last square, the program automatically loaded Phase 4 (CONNECT, INSERT, DELETE BOXES). If the diagram had been complete before the last square, you would have pressed UDK 15 to enter Phase 4.



TITLE

Flow Diagrammer

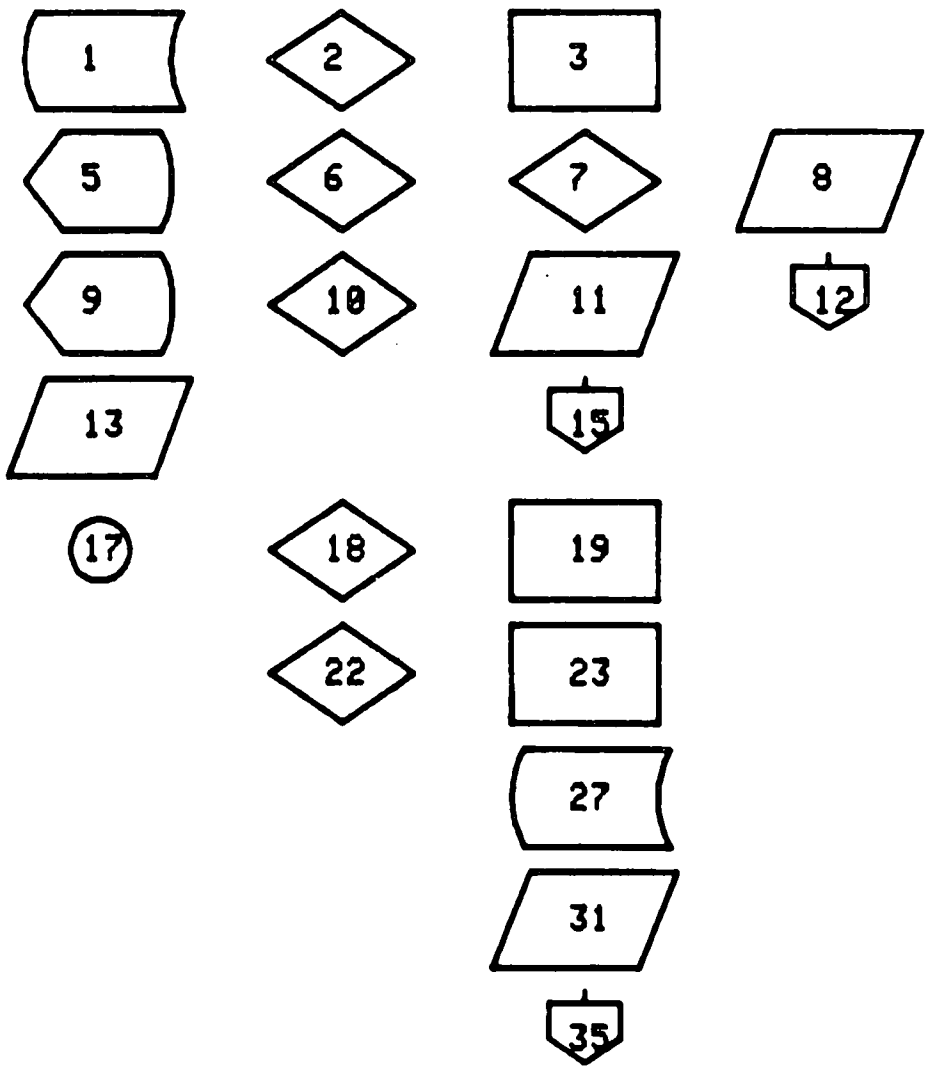
PHASE 4

CONNECT, INSERT, DELETE BOXES

When phase 4 is loaded, the screen is cleared and the defined boxes are displayed with their assigned numbers. All UDKs are currently meaningful except 20. Consequently, boxes may be added or removed at any time during this phase. If a box has been connected to other boxes and is then removed, all connections to and from the box are also removed.

Shown below is the initial phase 4 display.

from



TITLE

Flow Diagrammer

Connecting Boxes

For each pair of boxes to be connected, enter the 'from' box number and press RETURN. When the word 'side' is displayed; enter R for right-hand side, B for bottom or L for left-hand side, and press RETURN.

The same procedure is repeated for the 'to' box, except R, L or T for top are the permissible sides.

Input the type of line desired when the word 'type' is displayed.

Line Types

Three line types are available: 1, 2 or 3 (and their dashed alternates: 4,5,6).

Type 1 connects boxes by drawing in the up or down direction from box to box, then in the 'across' direction to complete the connection.

Type 2 draws half-way across, then up or down, (with an arrow in the middle of this line) then the other half-way across.

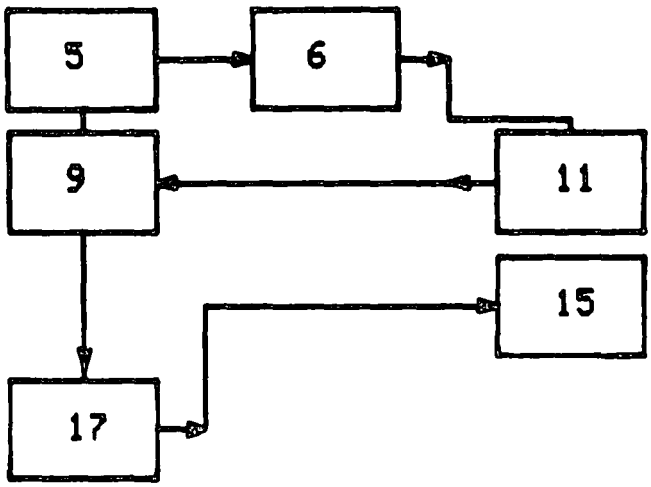
Type 3 draws across, then up/down.

Examples are shown on the next page. Note particularly the connections from boxes 6 to 11 and 17 to 15.

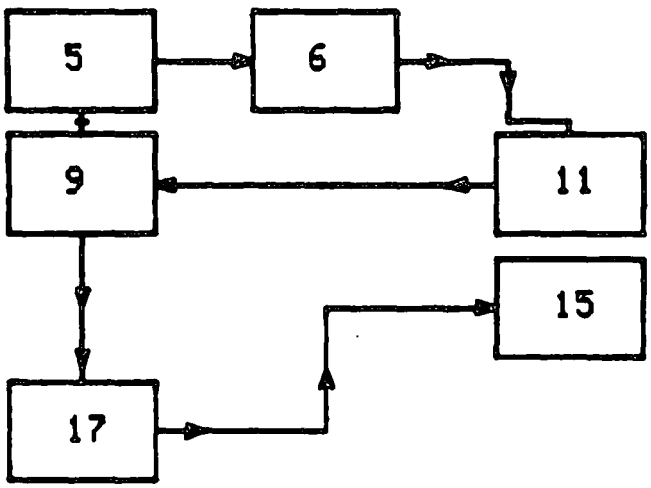
TITLE

Flow Diagrammer

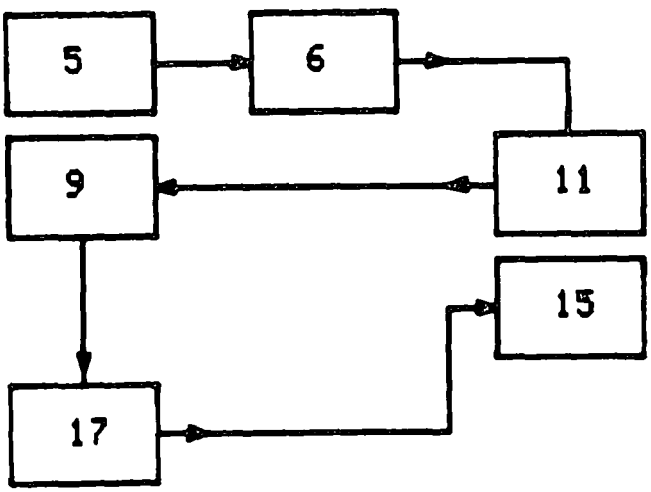
Line
Type 1



Line
Type 2



Line
Type 3

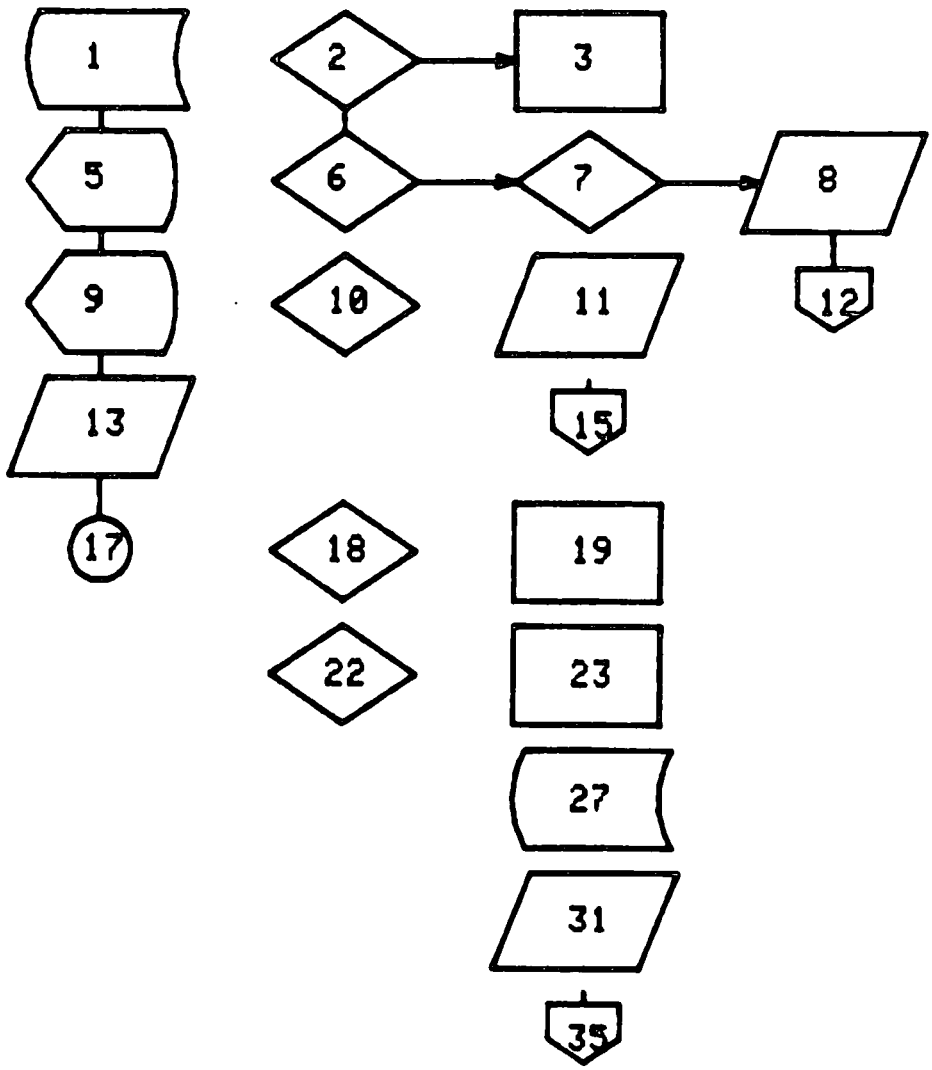


TITLE

Flow Diagrammer

In the following illustration, nine commands connect boxes and the screen is almost full. When RETURN is pressed after the 10th command, the screen will be erased, the diagram redrawn and 'from' will appear at the top of the screen. Connection may then continue.

```
from 1    side b
to   5    side t
type 1

from 5    side b
to   9    side t
type 1

from 9    side b
to   13   side t
type 1

from 13   side b
to   17   side t
type 1

from 2    side r
to   3    side l
type 1

from 2    side b
to   6    side t
type 1

from 6    side r
to   7    side l
type 1

from 7    side r
to   8    side l
type 1

from 8    side b
to   12   side t
type 1

from 7    side b
to   11   side t
type 1
```

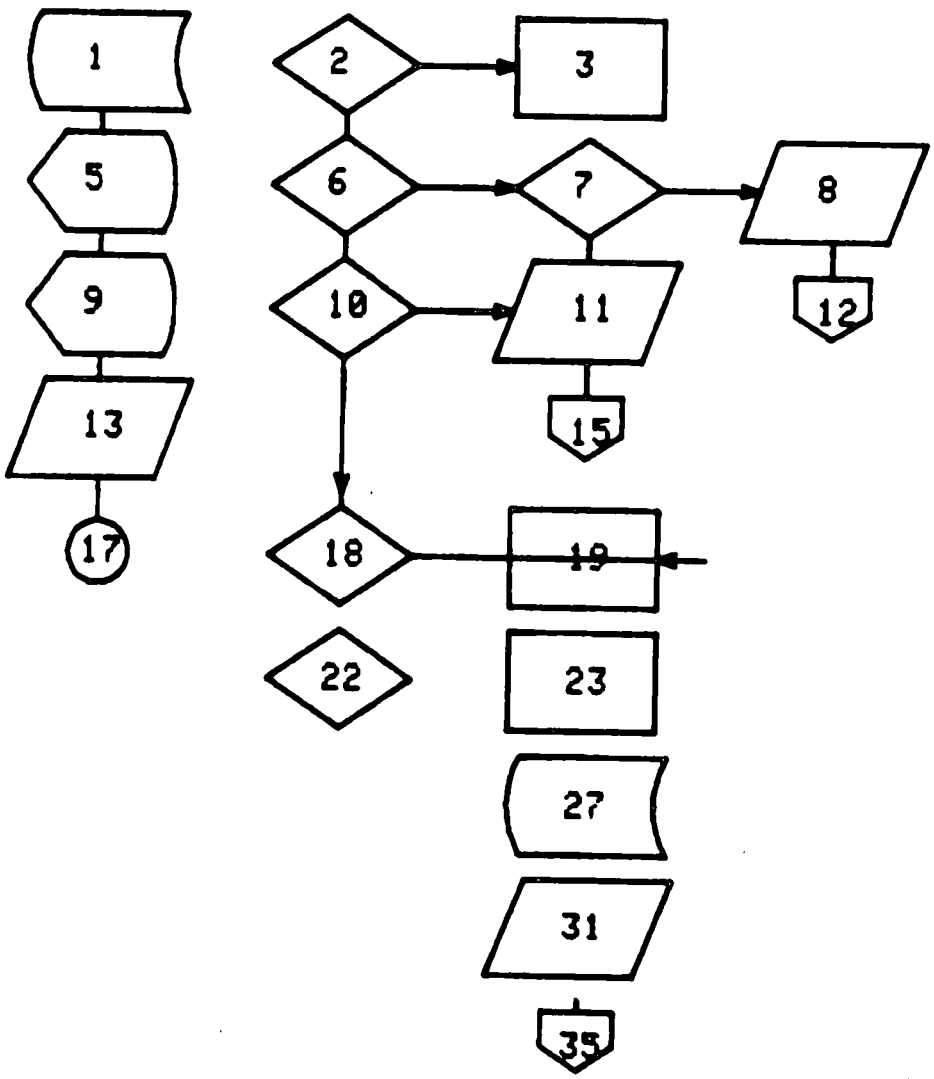


TITLE

Flow Diagrammer

A mistake is made by requesting that the right side of box 18 be connected to the right side of box 19. To correct such mistakes, press UDK 12 (Delete Last Line). The screen will be paged and the diagram without the last line re-displayed.

```
from 6    side B
to   10   side T
type 1
from 10   side R
to   11   side L
type 1
from 11   side B
to   15   side T
type 1
from 10   side B
to   18   side T
type 1
from 18   side R
to   19   side R
type 1
from
```



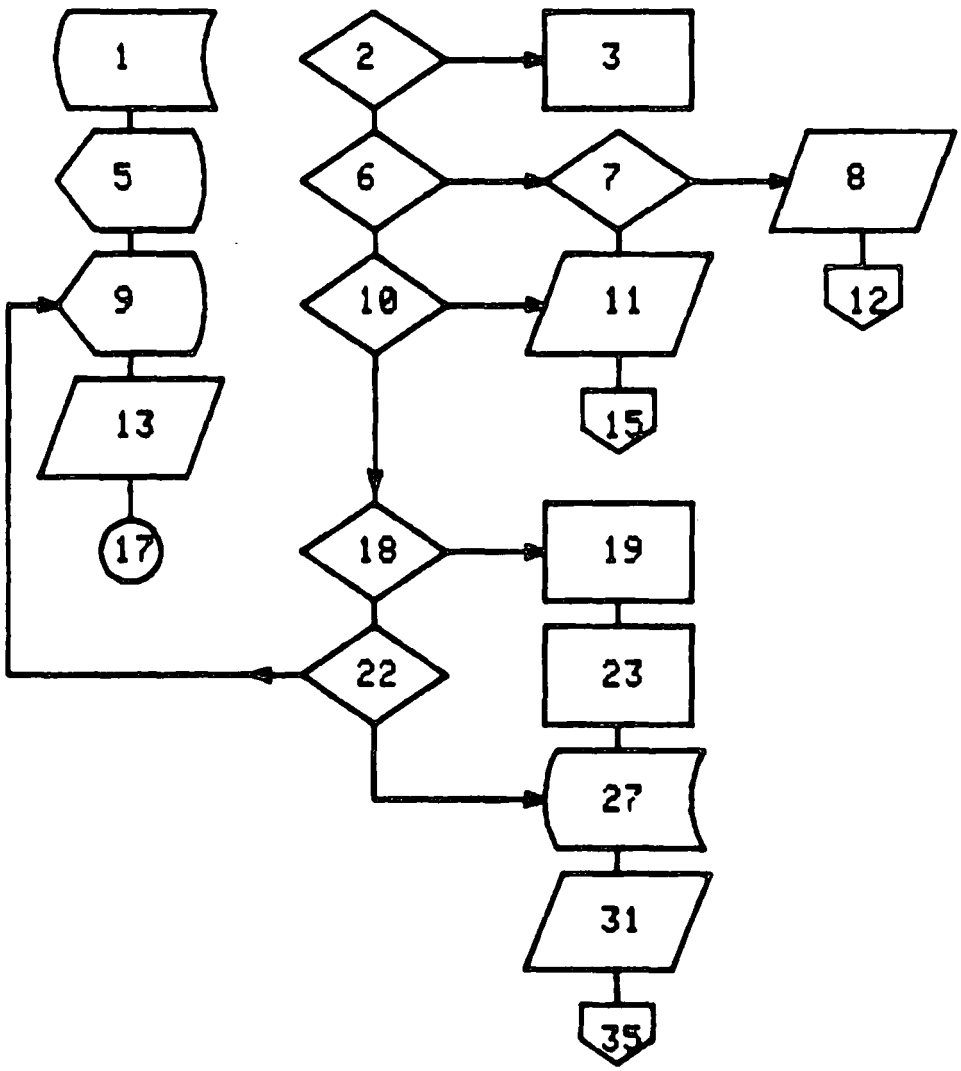
TITLE

Flow Diagrammer

Line connecting may continue. Shown below is the complete connected diagram.

You now have two choices: to plot the diagram or to load phase 5. At this point it is prefereable to press UDK 17 (direct output to the plotter) then UDK 15 (redraw). The diagram will be output to the 4662 Plotter.

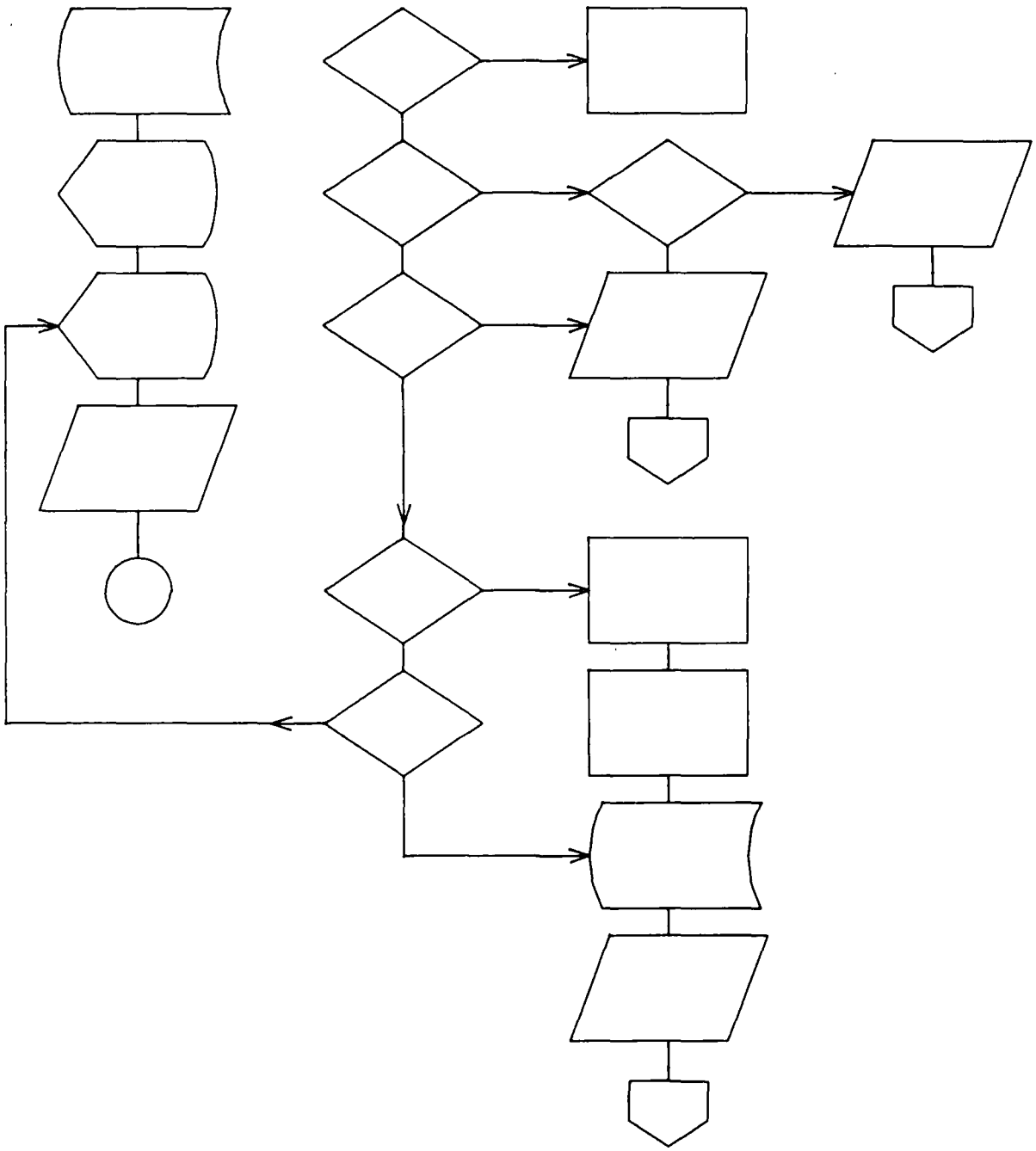
```
from 18   side R
to   19   side L
type 1
from 19   side B
to   23   side T
type 1
from 23   side B
to   27   side T
type 1
from 27   side B
to   31   side T
type 1
from 31   side B
to   35   side T
type 1
from 22   side B
to   27   side L
type 1
from 22   side L
to   9    side L
type 3
from 18   side B
to   22   side T
type 1
from
```



TITLE

Flow Diagrammer

Pressing UDK 17 and UDK 15 plots the connected symbols on the plotter.



TITLE

Flow Diagrammer

PHASE 5

ENTER BOX DATA AND HEADING

Now load phase 5 by pressing UDK 19. The following menu will be displayed. Enter 1 to input the heading. (You must enter a heading to save your data.)

```
Enter:
 1) to enter heading
 2) to enter box data
 3) to plot box data
 4) return to main menu

CAUTION:  You must enter a heading to save your data
1


Enter heading: FLOW DIAGRAMMER: PHASE 2
```

At this point, the heading, box types, connectors and number of connectors are written to the work file on disk.

The menu will return to the screen; enter '2' and press RETURN.

```
        FLOW DIAGRAMMER: PHASE 2


Enter:
 1) to enter heading
 2) to enter box data
 3) to plot box data
 4) return to main menu

CAUTION:  You must enter a heading to save your data
2
```

TITLE

Flow Diagrammer

Enter a box number. Up to four lines of information may be entered into the box (up to 20 characters per line), and information, if any, to appear outside the box (up to 8 characters per line).

Press RETURN to omit a line. Automatic centering of in-box data will take place. Do not add extraneous spaces after a line since they will be counted as characters in the center algorithm. If it's necessary to enter data on, say, lines 1, 3 and 4, enter a space in line 2 before pressing RETURN.

```
box number: 1

Enter line 1: READ
      line 2: DIAGRAM
      line 3: DATA
      line 4:

      above box: 790
      on right connector:
      on bottom connector:
      on left connector:
```

After entering all box data, select menu item 3 'plot box data" and the heading and annotation will be output to the plotter.

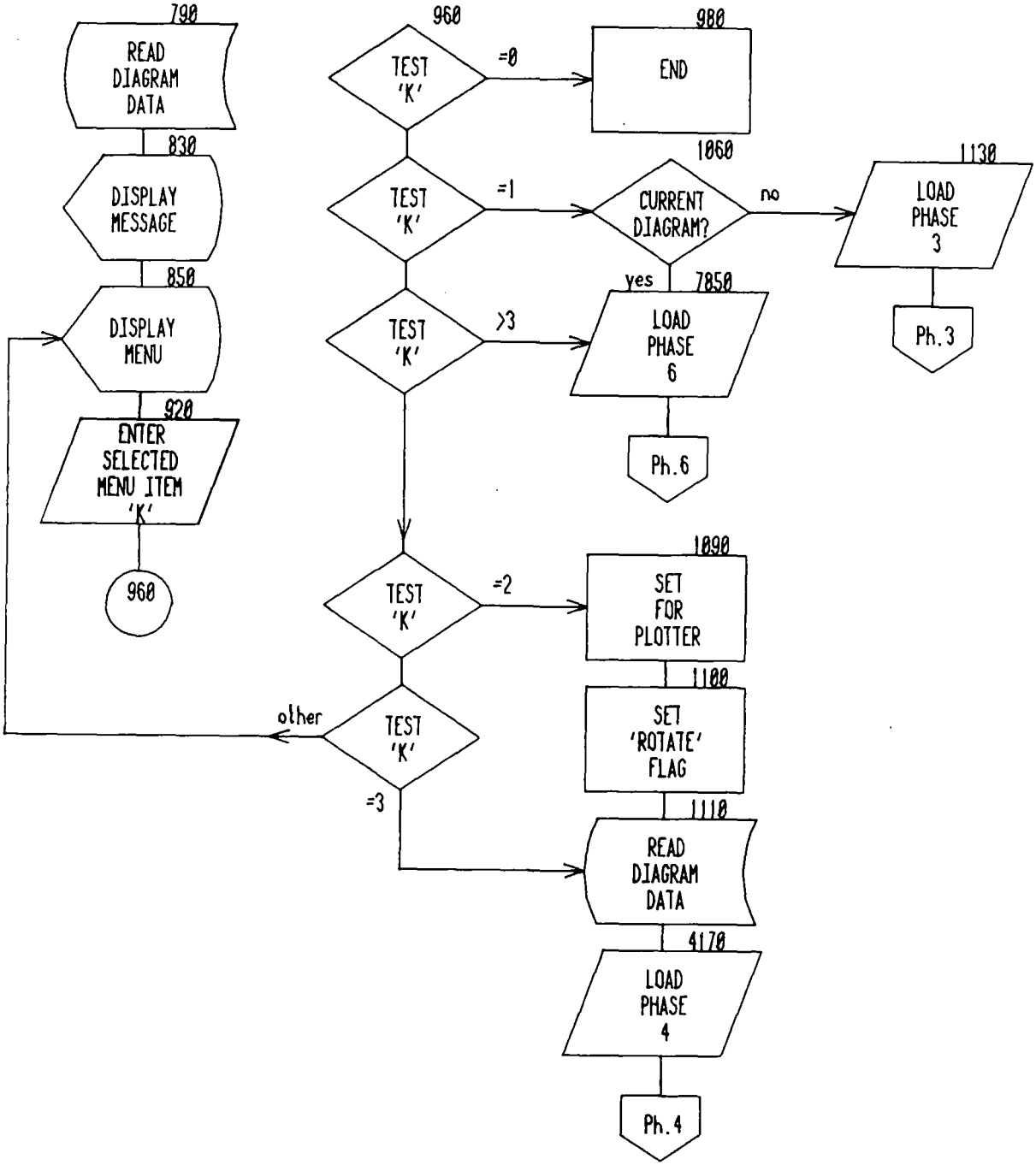
Phase 2 will then be reloaded and the main menu redisplayed.

Corrections may be made at this point, by choosing main menu item 3 which will load Phase 4.

TITLE

Flow Diagrammer

FLOW DIAGRAMMER: PHASE 2



TITLE

Flow Diagrammer

PHASE 6

STORE/RETRIEVE/DESTROY DIAGRAMS

In this phase, you may store/destroy the current diagram (if any) and store/destroy any stored diagrams.

If a current diagram exists and you enter phase 6 from main menu item 1 or menu item 4, you will be given a chance to save the current diagram before retrieving or beginning another.

TITLE

Flow Diagrammer

INTERNAL DATA STORAGE

Variable	Used to Store	Type
A,B,C,D	Endpoints of lines	Simple
C5	Cosines	Array (25)
H$	Flow diagram heading	String (51)
K	Last function key	Simple
L$	Diagram	Array (2001)
M	Device address	Simple
R	Rotate flag	Simple
S	Dotted line flag	Simple
S5	Sines	Array (25)
T	Box types	Array (4,10)
Z	Connectors	Array (50)
Z1	Number of connectors	Simple

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE		
Segemented Data Base and Windowing Routines		
ORIGINAL DATE	REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED
January, 1978		32K
AUTHOR	Tektronix, Inc. Wilsonville, OR	PERIPHERALS
Leslie L. Brabetz		

ABSTRACT

Files: 3 ASCII Program
2 Binary Data
Statements: 701

A series of articles in TEKniques (Vol. 1 No. 10, and Vol. 2 Nos. 1 and 2) described the theory and operation of creating a segmented data base from a large serial data base for fast windowing. Five files included in this program illustrate the mechanics of carrying out segmentation and windowing.

One routine allows definition of rectangular data windows. A master file may be read in and the vectors which begin and end or intersect the data window are stored in a segment file. The coordinates of intersection with the boundaries are calculated and stored in the segment file. The master data file must be in the form of arrays, with the number of coordinate pairs, N, followed by the coordinate arrays, X,Y.

$$N, X_1, X_2, \ldots, X_n\ Y_1, Y_2, \ldots, Y_n$$

Output segment files are created with the same format. To apply this routine to a user's data will require some revision of the program I/O and segment definition.

A small routine is included which generates the two data files. A third routine similar to the first is included. However, it directs the output to the display rather than a segment file and input files are read from the tape rather than the disk.

TITLE

Segmented Data Base and
Windowing Routines

4907 FILE MANAGER IS A POWERFUL GRAPHICS AID

By Les Brabetz and Gary P. Laroff

The last issue of TEKniques (Vol. 1 No. 9) introduced the 4907 intelligent flexible disc mass storage unit for the 4051. A SORTing program was also included in that issue. This month begins a series of articles on powerful graphics handling routines made possible by the large direct access storage capability and multiple-level file-by-name library structure supported by the 4907. Just as numerical records can be accessed in any order on the disc, so can graphic entities be stored and retrieved.

Consider a large data base, perhaps the high resolution map of the United States in Fig. 1. It might be desirable to plot the entire map. In this case all of the coordinates in the data base are required and can be read sequentially. This is as easy and convenient to do with data stored on magnetic tape as it is with disc storage methods. But what if only a subset of the map is desired? How is it possible to quickly locate those coordinates that will be plotted in the desired map area, and discard those that fall outside the boundaries? The file library system available on the 4907 is designed to offer solutions to such graphic applications.

The data base consists of about 3000 X,Y pairs and takes about four minutes to display. The objective is to store the data so that any map area can be chosen and quickly displayed. The data base designs are described in this article. Succeeding articles will present a program to construct a segmented data base from an existing sequential data base and another program that selects the segments to display.

The map as originally digitized can be displayed with a number of DRAW commands and MOVE commands. The data is stored as a series of coordinates that can be displayed with a MOVE to the first coordinate pair and an array DRAW to the remaining coordinates. Because each of these data sections consists of a different number of coordinates, the number of coordinate pairs is also stored. The data format is:

$$N, X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_n$$

where N is the number of coordinate pairs. The data base could be read and displayed from either tape or disc with similar programming:

FROM TAPE

```
100 READ @33:N
110 DELETE X,Y
120 DIM X(N),Y(N)
130 READ @33:X,Y
```

FROM 4907

```
100 READ #1:N
110 DELETE X,Y
120 DIM X(N),Y(N)
130 READ #1:X,Y
```

These arrays can then be displayed by a MOVE to the first point and a DRAW for the entire array:

```
140 MOVE X(1),Y(1)
150 DRAW X,Y
```

All of the arrays are READ and displayed in this manner. The data base consists of 30 arrays containing about 3000 coordinate pairs.

This data base is not optimized for displaying subsections of the map. With this filing method, four minutes are required to display any map section because the 4051 must READ every data pair and "clip" all graphics that would appear outside the designated window.

A far superior technique would be to divide the map into many smaller sections. The program could then look at the desired plot boundaries and choose those map sections required for the display. The random file access feature of the 4907 makes this quite practical. For the

TITLE

Segmented Data Base and Windowing Routine

Fig. 1. Master file of digitized map with arbitrary grid values.

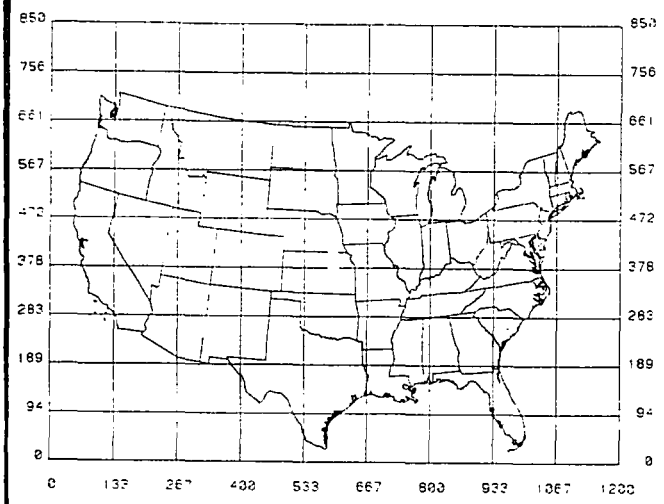
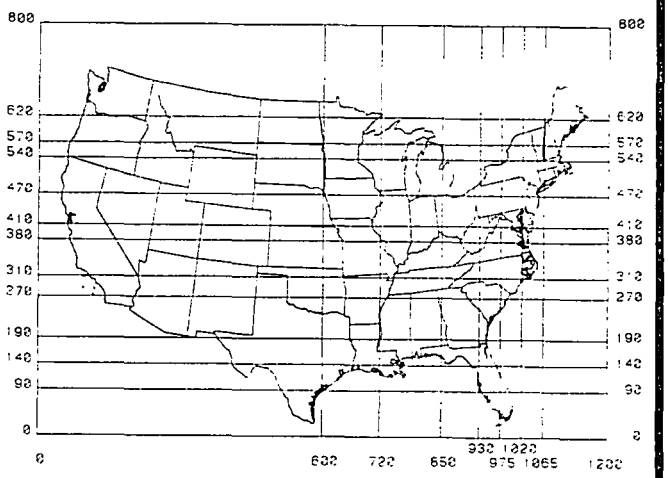


Fig. 2. Map data base illustrating segment division lines to produce 96 map segments.



purposes of this description, 96 segments created a data base that could quickly display any section of the original map. Fig. 2 shows the map as segmented into eight divisions horizontally and twelve vertically. An attempt was made to select a segment definition which would create an equal distribution of coordinates between the file segments. The fine detail required by the eastern coastline and minimum detail required for western state boundaries caused the density of graphic coordinates to be far greater toward the eastern part of the map. Smaller segment definition was required in some areas in order to attain reasonably equal vector densities in each segment.

Each segment is a rectangle and must contain the coordinates of intersection of any map lines with its borders. This is a substantial task; the program that builds this segmented data base will be discussed in the next issue.

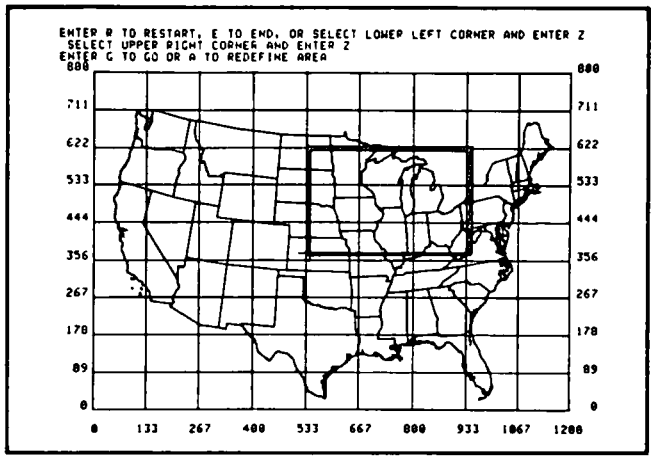
Once the segmented data base is completed (as 4907 files USMAP/SEG1 through USMAP/SEG96), a simple program can be used to access the files, then quickly display any segment of the map. In this manner, one can rapidly access and display any section of a large data base (such as a map) and then use custom routines to add such information as county boundaries, major highways or weather information.

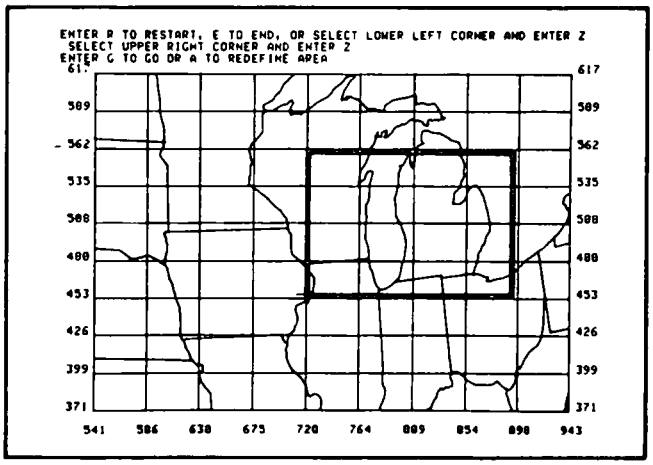
TITLE

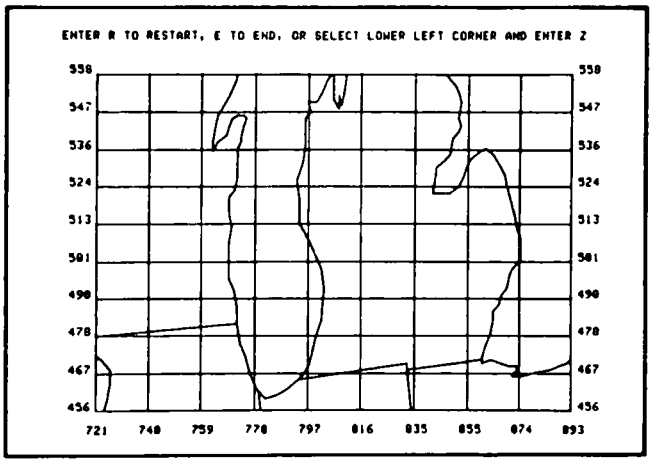
Segmented Data Base and
Windowing Routines

4907: Segmented Data Base Provides Fast Graphics Access

By Les Brabetz







(This is the second of three articles on using the new 4907 FILE MANAGER flexible disc mass storage unit as a powerful graphics aid.)

The article on segmented data bases in the previous issue of TEKniques (Vol 1 No. 10), described the use of a segmented graphics data base. The techniques discussed permit rapid display of portions of the data base rather than displaying the entire data base to view a smaller portion. Only those files (or segments of the data base) that are required to complete the display are read into memory and displayed on the screen. The ability of the 4907 FILE MANAGER to randomly access files allows the effective use of this data base technique. A file is created for each segment of a coordinate grid that overlays the entire data base. A rectangular grid was defined which varied in segment size to provide a reasonably equal density of vectors per segment. The segments generated with this grid defined some empty files, but was a much better distribution of vectors than a uniform grid.

This article describes construction of the segment files from the master data base file. Once the grid coordinates

Fig. 1. Default map of the United States with area selected for "zooming" (heavy rectangle).

Fig. 2. Map segment selected in Fig. 1 as displayed from segmented data base. Second "zoom" requested of Lake Michigan (heavy rectangle).

Fig. 3. Lake Michigan area as requested in Fig. 2 and displayed from segmented data base.

TITLE

Segmented Data Base and Windowing Routines

are defined, there are two possible ways of building the segment files. The first method is to digitize a portion of the data base into each segment file. Any vector which crosses the segment boundaries must end exactly on the boundary, and must end at the correct angle to intercept the vector approaching from the adjacent segment; this is quite difficult. The main requirement of the segmented data base is to provide vector continuity across segment boundaries. The second method, which was used to build the demonstration data base of the U.S. Map, is to take a large data base and select vectors to transfer to separate files. The problem of boundary vector continuity is overcome by calculating an intercept point on the boundary, and adding this point to the segment file.

The original data base of the U.S. Map consists of approximately 3000 X-Y coordinate pairs. Building a segment file requires examining each point and determining which one of the following four conditions fits that point:

1. The new point and the last point are within the defined area.
2. The new point and the last point are outside the defined area.
3. The point is entering or exiting the area relative to the last point.
4. The last point was outside and the new point is outside, but the vector drawn between the points passes through the segment area.

Conditions one and two are the simplest case as the point is either stored in the segment file or discarded. Condition three requires the slope and direction of the vector to be calculated. Once these are known, the boundary intercept point may be determined. Condition four is the most elaborate solution as it requires determining the points of boundary intersection for both entrance and exit.

A rectangular definition of the segment was chosen to ease the calculation of the boundary intercept point. Once direction and slope of the vector is determined, one boundary coordinate is calculated and stored.

Determining the Boundary Intersection

Refer to the example in Fig. 4. Point P2 exceeds Y3, the maximum value of the Y range of the segment definition. A set of coordinate axes are placed on the point P1 and the quadrant of operation determined by the point relationship of P1 and P2. For this example, quadrant I is defined. The slope of the vector connecting P1 and P2 is used to determine which boundary will be intercepted.

If the slope of the vector connecting P1 and P2 is greater than the slope of the vector connecting P1 and P4, Y3 is used for a known boundary value. If the P1 to P2 slope were less than the P1 to P4 slope, then X4 would be used for the boundary calculation. X3 is calculated from the slope and the Y3 intercept value:

X3 = slope * distance + origin
= ((Y2 - Y1)/(X2-X1) * (Y3 - Y1)) + X1

The boundary point P3 is added to the array and output to the segment file. Once the slope of the vector is determined and the boundary value known, the intercept coorinates are easily determined.

The coordinate pairs are read from the master data base file in the form of an array, and are output to the segment file in the same fashion. The output array is stored in memory until the segment boundary is crossed or until reading of the input array has been completed. If the entry array of vectors is outside of the defined area, no transfer occurs between the input and output files. The coordinate transfer begins when a point is inside the area and ends when the array point crosses the boundary.

After the first attempts were made at using this file transfer technique, two enhancements were formulated. The first was to generate four quadrant files from the main data base and then to use them for the segment file creation. This reduced the amount of time to create the segments by reducing the amount of data to scan and test against the segment area. The second enhancement was to

TITLE

Segmented Data Base and
Windowing Routines

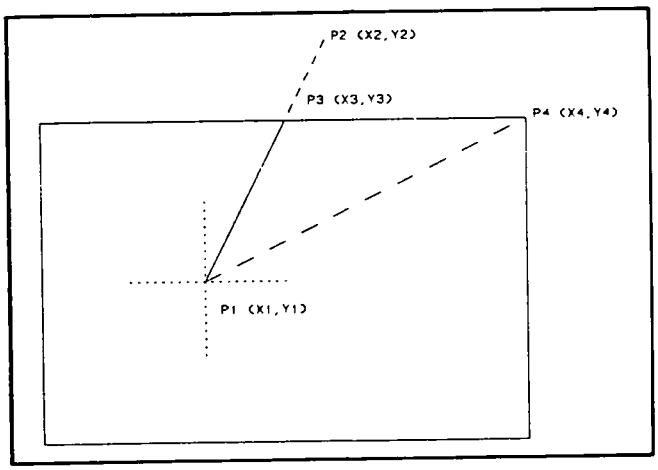


Fig. 4. Example of boundary intercept calculation for condition 3, vector exiting boundary area.

attempt an equal distribution of vectors among many small segments. Careful definition of the grid for the segments is important in both creation time and display time.

Approximately four hours were required to generate the segmented data base. No operator intervention is required as input files are selected and output files defined by the segment definition program. Once created, segments can be chosen and displayed in seconds. The program used for windowing will be documented and placed in the 4051 Applications Library.

The next article regarding the segmented data base will describe the selection of the segment files for building displays. Tekniques

TITLE

Segmented Data Base and Windowing Routines

4907: Graphic Displays from Segmented Data Base

by Les Brabetz & Gary P. Laroff

(This is the third of a three-part article on the new 4907 FILE MANAGER intelligent flexible disc mass storage unit as a powerful graphics aid. The first two articles, in TEKniques Vol. 1 No. 10 and Vol 2 No. 1, described the concept and creation of a segmented data base using the 4907 FILE MANAGER. This article will complete the series by providing background summaries from the first two articles, then will describe segment file selection for creation of a display.)

A segmented data base stores a large, highly detailed graphics display in the form of X-Y coordinate arrays. The storage format is the number of coordinate pairs in an array followed by the coordinate pairs.

$$N, X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_n$$

Initially, a data base exists. It is either computed and stored or digitized and stored. Such a data base consists of some number of arrays of DRAW commands. These could be continuous sections of a map. Islands off the coast would be separate arrays because a MOVE is required to get across the water. A good example of a typical data base is the map of the United States discussed in the first article in this series (TEKniques Vol. 1 No. 10) and reproduced here as Fig. 1. The disadvantage of such a typical data base is that it can only be accessed serially; that is, to see the information stored at the end of the data base, one needs to READ all of the earlier information.

In order to access the graphic data quickly, the data base is segmented into a number of rectangular sections, each of which is stored on the 4907 disc as a separate file. The program that does this was discussed in the last issue of TEKniques (Vol. 2 No. 1). The trick is to compute and store all points where the map crosses one of the straight "segment boundaries.". Any segment may now be called and the data read as quickly as any other.

In summary, each segment or portion of the display consists of a file containing, in theory, an equal number of coordinate pairs. Actually the number of pairs in an array may vary widely. Display or vector generation from the array is performed by MOVing to the first coordinate pair and DRAwing through the remainder of the array. Creation of these segment files is performed by scanning the master data base and selecting those vectors that lie within the defined portion of the display.

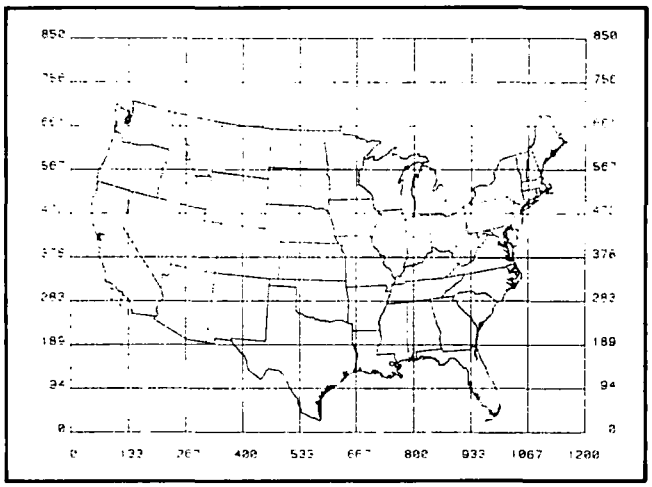
Once the segment files have been generated, a means of selecting and displaying the necessary files to build up a display is required. Each segment file is considered a separate entity, independent of the other files in the data base. These building blocks are selected and fit together to fill a defined area. The division of the map of the United States into 96 segments is shown in Fig. 2.

So now that we know how to create a segmented data base, how do we pick the data segments? Being numerical data, the segments have X and Y coordinates associated with each corner. If we then number the segments, we will be able to write a program that will convert minimum and maximum values of X and Y into segment numbers. More precisely, if we DRAW a grid with coordinates over our master data display, we can then enter minimum and maximum values for X and Y (defining the lower left and upper right corners of the rectangular segment, respectively) and have our program look up all of the graphic segments that make up the rectangle. Our looked-up segments will usually form a slightly larger section of the data base than requested.

TITLE

Segmented Data Base and Windowing Routines

Fig. 1. Master file of digitized map with arbitrary grid values.



Our program would then set the graphic WINDOW to the requested rectangle, READ the appropriate numbered segments and display them. Easier said than done? Not really. Let's go a little deeper.

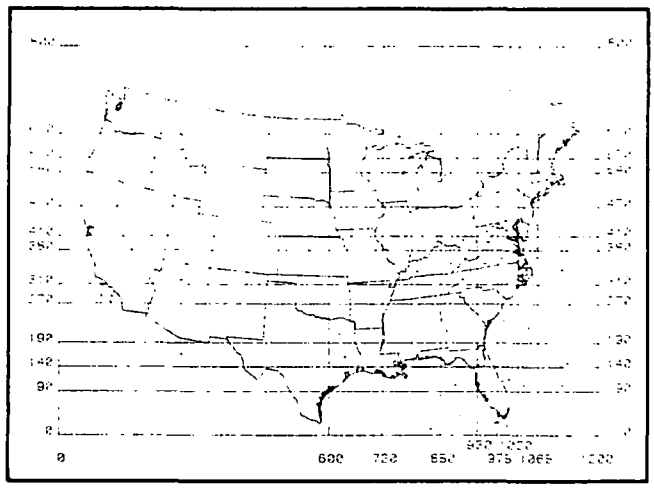
Selecting the Display Area

The 4907 FILE MANAGER demonstration program, which displays the U.S. Map graphics data base, uses the graphics cursor to select the area to display. First the master data base file of the entire map is displayed. The user is requested at this point to select a rectangular area of interest. The graphics cursor is enabled through the POINTER command and the 4952 Option 2 Joystick positions the cursor. When the user positions the graphics cursor to the lower left corner of the desired rectangle, a key is struck on the 4051 keyboard to indicate selection of the location. This returns the X-Y coordinate of the location (in user defined window units) and the character struck. This process is repeated for the upper right corner. The upper and lower limits of the X and Y ranges are now defined. This is only one method of selecting the area. These coordinates could be entered from the keyboard, defined by an operating program, or any other appropriate technique to define the numerical range of interest.

Determining the Segments

The previously defined X and Y coordinates which delineate all the segment boundaries now reside in two arrays within the 4051 memory. Selecting the segment files is performed by comparing the range of interest with the segment boundary values and choosing the segment boundary values which encompass the range of the desired display. Most of the time the segments to be displayed exceed the defined display area, and the internal clipping features of the 4051 display (WINDOW and VIEWPORT) are used to limit the vectors being displayed.

Fig. 2. Map data base illustrating segment division lines and numbering scheme for identification of segments.



A simple example will illustrate selection of actual segments. Each segment file is named to correspond with a location in the matrix in Fig. 3, e.g., SEG1 through SEG12. The data range to be displayed will be:

$X_{min} = 150$ $Y_{min} = 50$
$X_{max} = 250$ $Y_{max} = 250$

TITLE

Segmented Data Base and Windowing Routines

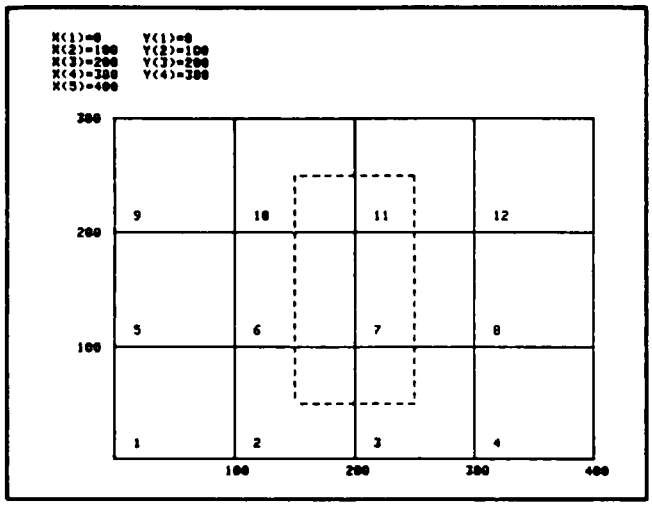


Fig. 3. Example of segment numbering scheme and selection of segments required for a display.

The segment files are chosen from the data base in the following fashion. First the X range is determined by comparing the minimum value with the array values. X(2) is selected as the segment boundary which is less than or equal to the minimum. X(4) is selected as the maximum segment boundary by containing a value greater than or equal to the maximum value. Y(1) and Y(4) are selected in the same fashion. We now know the X range of segments to be 2 to 3 and the Y range as segments 1 through 3.

Calling the Segments

Segments are selected by a row and column method with the row determined by the Y segment range and the column by the X segment range. Since 4907 FILE MANAGER segment files are selected by name, rather than by number, string variables containing the segment numbers will be constructed. First we will generate a master string, A$, which will contain all of our required segment numbers. Segment numbers will be separated within A$ by a space. SEGMENT FILE NUMBER = (ROW - 1) * 4 + COLUMN.

```
1000 REM SEGMENT FILE SELECTION
1010 A$=""
1020 FOR I=2 TO 3
1030 FOR J=1 TO 3
1040 A=(J-1)*4+I
1050 B$=STR$(A)
1060 A$=A$&B$
1070 NEXT J
1080 NEXT I
1090 END

PRINT A$
 2 6 10 3 7 11
```

When the above section of code is executed the string A$ contains the segment file numbers required to complete the display. The string values are separated by spaces which are used as delimiters when selecting the individual files. For the demonstration data base "SEG" was prefixed to the segment numbers. Thus segment 1 is called "SEG1" and segment 12 is called "SEG12." The files are selected by this process:

```
2000 REM  SEGMENT FILE SELECTION AND DISPLAY
2010 C1=2
2020 A$=A$&" "
2030 C=POS(A$," ",C1)
2040 D$=SEG(A$,C1,C-C1)
2050 D$="SEG"&D$
2060 C1=C+1
2070 REM  OPEN AND DISPLAY SEGMENT FILE
2080 GOSUB 3000
2090 IF C1<LEN(A$) THEN 2030
2100 REM END OF DISPLAY
2110 END
3000 PRINT D$;"  ";
3010 RETURN

RUN
SEG2  SEG6  SEG10  SEG3  SEG7  SEG11
```

Line 2020 appends a space to the end of A$ so that we can pick off the last segment number. Variables C and C1 are used as pointers to the spaces present in A$. C takes on the value of the position of the next space in the string with C1 containing the position of the last space plus one position. D$ in line 2040 becomes the segment file number located between spaces. To complete construction of the filename, "SEG" is appended to the segment number in line 2050. The filenames are printed to the screen in this sample program. Normally, these filenames would be used with an "OPEN" command to the 4907. The contents would be read and displayed to the screen. The end of file (EOF) condition would be used to end the data input and "CLOSE" the file. Operation of this program ceases when the length of the string is exceeded by the pointer C1 in line 2090.

The routines developed to perform all of the 4907 graphics data base demonstration will be documented and submitted to the 4051 Applications Library. The master data base of the U.S. Map, the segment files and the program to select and display the segment files will be available for use with the 4907 FILE MANAGER.

Now you have the best of all worlds. With the new applications software program the 4051 will interactively retrieve and display sections of complicated graphics data bases with ease and speed. Interactivity is assured by the 4952 Option 2 Joystick and intelligent flexible disc mass storage is provided by the 4907 FILE MANAGER.

Tektronix

TITLE

Segmented Data Base and
Windowing Routines

WINDOWING ROUTINE

1. DESCRIPTION

The first step in constructing a segmented data base is to define the grid coordinates required to divide up the master data base.

X_1 , X_2 , Y_1 , Y_2 are the coordinates used within the windowing portion of the routine.

The operations are performed in this sequence once the files are selected and opened.

The vector count and arrays are read from the input master data base file. Each vector endpoint is tested for the following four conditions.

a. This point and the last point are within the defined window. The point is stored in the output array and the next point examined.

b. This point and the last point are both outside the defined window. The point is discarded and the next point examined.

c. This point is entering or exiting the boundaries of the defined window. The intercept point of the boundary being crossed must be determined and entered into the output array.

d. Both this point and the last point are outside the window, but the vector drawn between the points passes through the window.

An output array is generated of vector endpoints which lie within the window and those which are calculated to intercept the boundaries. The output array is created when a vector endpoint enters or begins within the area. The array is stored in the segment file upon reading the end of the input array or when a vector exits the window area.

Conditions a and b are fairly simple to handle, the coordinate pair is either stored or ignored.

Condition c requires the entry or exit point be determined and the output array created or stored in the segment file. Determining the intercept point on the boundary is accomplished by using the direction and slope of the vector. One of the two boundary coordinates is determined by the window definition coordinates. The other coordinate is calculated using the slope intercept formula.

See the enclosed hardcopy Figure 1 for a sample situation. The vector exiting originates at P1 and terminates at P2. The point of intersection with the boundary at P3 must be determined and entered into the output array. A set of coordinate axes are placed on the point P1

TITLE

Segmented Data Base and
Windowing Routine s

and the quadrant of operation determined by the direction of P1-P2. In this example, quadrant 1 is defined. Quadrant 1 indicates an interception of the maximum X(X4) or maximum Y (Y4) boundary. The slope of P1-P2 is compared to the slope of P1-P4 to determine which boundary value is used for the slope intercept calculation. For this example Y4 is selected as the boundary value and X3 is determined by the slope intercept calculation. If the P1-P2 slope were less than the P1-P4 slope, then X4 would be used for the known boundary value and the associated Y value would be calculated. The value for the boundary intercept value X3 is calculated from the slope of P1-P2 and the Y maximum boundary value Y4.

$$X3 = SLOPE * DISTANCE + ORIGIN$$
$$= ((Y2-Y1)/(X2-X1) * (Y3-Y1)) + X1$$

The coordinate pair P3 (X3,Y4) is added to the output array and the number of array elements and the array are added to the segment file. A vector exiting the boundary causes the output array to be transferred to the segment file. Any subsequent vectors from the master data file will be ignored until the window boundary is entered by another vector. A new output array will be started and filled until another exit occurs or the end of the master data file is detected.

The example just described applies to the case where a vector is exiting the window. A vector entering the window has its intercept point calculated by the same technique. The end points of the vector are exchanged and the vector treated as exiting the window. The reverse of the array storage process is also used. An output array and counter are initialized and the two points entered.

Condition d, the traversing vector, is the most difficult condition to detect, the most uncommon to occur and the most difficult case for calculating the window intercept points. When a vector traverses the window, both intercept points, entering and exiting, must be determined. A traverse condition is established by the next point and last point both outside the window area. The quadrant position in respect to the window is determined by the origin of the vector. See Figure 2 for an illustration. Point P1 to point P2 is the vector traversing the window. Quadrant A is selected because P1 is to the left of the defined window. If P2 were the origin, quadrant B would be selected. Note also a further breakdown of quadrants into subquadrants AC, AD, BC, and BD. A vector originating in AC might intercept boundary A or C, and requires an extra slope comparison to determine the intercepts.

TITLE

Segmented Data Base and
Windowing Routine s

Once the quadrant of origin is determined, the slope of the vector P1-P2 is calculated, and shown as A3. Angle A1 is constructed between P1 and the window corner AD. If slope A3 is greater than the slope A1, then the traverse vector does not pass through the window. Angle A2 is defined next and the slopes A3 and A2 compared to determine which boundary, D or B, is intercepted. The first intercept point in boundary A is calculated just as performed for condition 3. The second boundary intercept is calculated the same way. In both cases one boundary value is known, and the other may be determined by the slope intercept technique. The difficulty in finding the coordinates is simply the number of comparisons required to select the proper boundaries for the calculation.

As the traversing result is a single vector, the two coordinate pairs are written into the segment file directly.

2. DATA FILE STRUCTURE

Both the input master file and the output segment file contain the same format, data coordinate pairs. The data is stored in the format of:

$$N, X_1, X_2, X_3, \ldots X_n, Y_1, Y_2, Y_3, \ldots Y_n$$

Where N is the number of coordinate pairs. These arrays of data are accessed by this method.

```
LIS 600,650
600 DIM X(591),Y(591),O(592),P(591)
610 ON EOF (1) THEN 890
620 READ #1:N
630 DELETE X,Y,O,P
640 DIM X(N),Y(N),O(N),P(N)
650 READ #1:X,Y
```

The input arrays are read by this technique while output arrays are built up in memory and stored in the segment file upon completion of an array.

TITLE

Segmented Data Base and
Windowing Routines

This section of the program is used to output the arrays to the segment file logical unit #2.

```
2040 WRITE #2:J
2050 DIM O(J),P(J)
2060 WRITE #2:O,P
2070 DELETE O,P
2080 DIM O(591),P(591)
2090 J=0
```

3. VARIABLE LIST AND PROGRAM SEGMENTATION

A	slope of vector being tested	
A1	slope to near window corner	
A2	slope to far window corner	
B,C	last X and Y coordinate	
D,E	next or current point	
B1,C1 D1,E1	intercept coordinates	
F1	direction flag (exit or enter)	1 indicates enter
F	window flag	1 indicates outside
J	output array counter	
J1	Y index for segment generations	
I1	X index for segment generation	
Q2	last quadrant file	
Q1	next quadrant file selected	
S(1,n)	X segments	n increments 1-9
S(2,n)	Y segments	n increments 1-13
S1	segment number in use	
X1 X2	X segment range	
Y1 Y2	Y segment range	
X3 Y3	dummy for swap of points for entrance calculation	
X & Y	input arrays	
O & P	output arrays	
N	array size	
Z	quadrant selection for intercept	

TITLE

Segmented Data Base and
Windowing Routine s

4. SEGWINDOW

Subroutine Locations

160- 410	Segment increments defined
420- 950	Driver section - defines segments, outlines, and inputs arrays until end of file
960-1210	Selects quadrant for input
1220-1370	First point in array and set flag F if out
Condition 1	
1380-1460	Point in window
Condition 2	
1470-1940	Exit intercept
1950-2100	Store output array and get next point
2110-2160	Vertical exit
2170-2220	Horizontal exit
2230-2310	Last point outside window
Condition 3	
2320-2400	Vector entering window - treated same as exit but points swapped
2410-2540	Begin array and store intercept coordinates
Condition 4	
2550-2650	Quadrant selection
2660-2920	Quadrant A
2930-3080	Quadrant B
3090-3230	Quadrant C
3240-3380	Quadrant D
3390-3490	Calculate intercepts and save
3500-3560	Both end points in Y range of window
3570-3600	Y values do not change
3610-3670	Both end points in X range of window
3680-3710	X values do not change
3720-3910	SUBquadrants for second intercept steering to correct section ofr calculation

5. OPERATING INSTRUCTIONS

Five files accompany this documentation. The first routine (File 24 on PROGRAMMING AIDS T1 tape) is the most extensively documented and is the routine used to generate a segmented data base.

The third routine is a small program which generates the data files found in Files 27 and 28. These files consist of 71 radial vectors which form the display shown in Figure 3.

TITLE
Segmented Data Base and
Windowing Routines

File 27 contains vectors which begin at the center of the circle and end on the radius.

File 28 contains vectors which begin at the radius and end at the center of the circle. These files are used with File 25 which is a version of the window routine used for evaluation.

Hard copies of the displays generated and parameters entered are shown in Figures 4-7. This program may be used to familiarize and demonstrate operation of the windowing routine.

The first and second routines (files 24 and 25) are similar in function beyond program line number 1220.

Output from the second routine is directed to the display rather than a segment file and input files are read from the tape rather than the disk.

To use these routines in a user application will require some revision of the program I/O and segment definition.

NOTE: If these five files are transferred to different file locations on a tape, change the following statements:

Second Routine (file 25)	Statement 470
Third Routine (file 26)	Statement 130
	Statement 230

TITLE

Segmented Data Base and
Windowing Routines

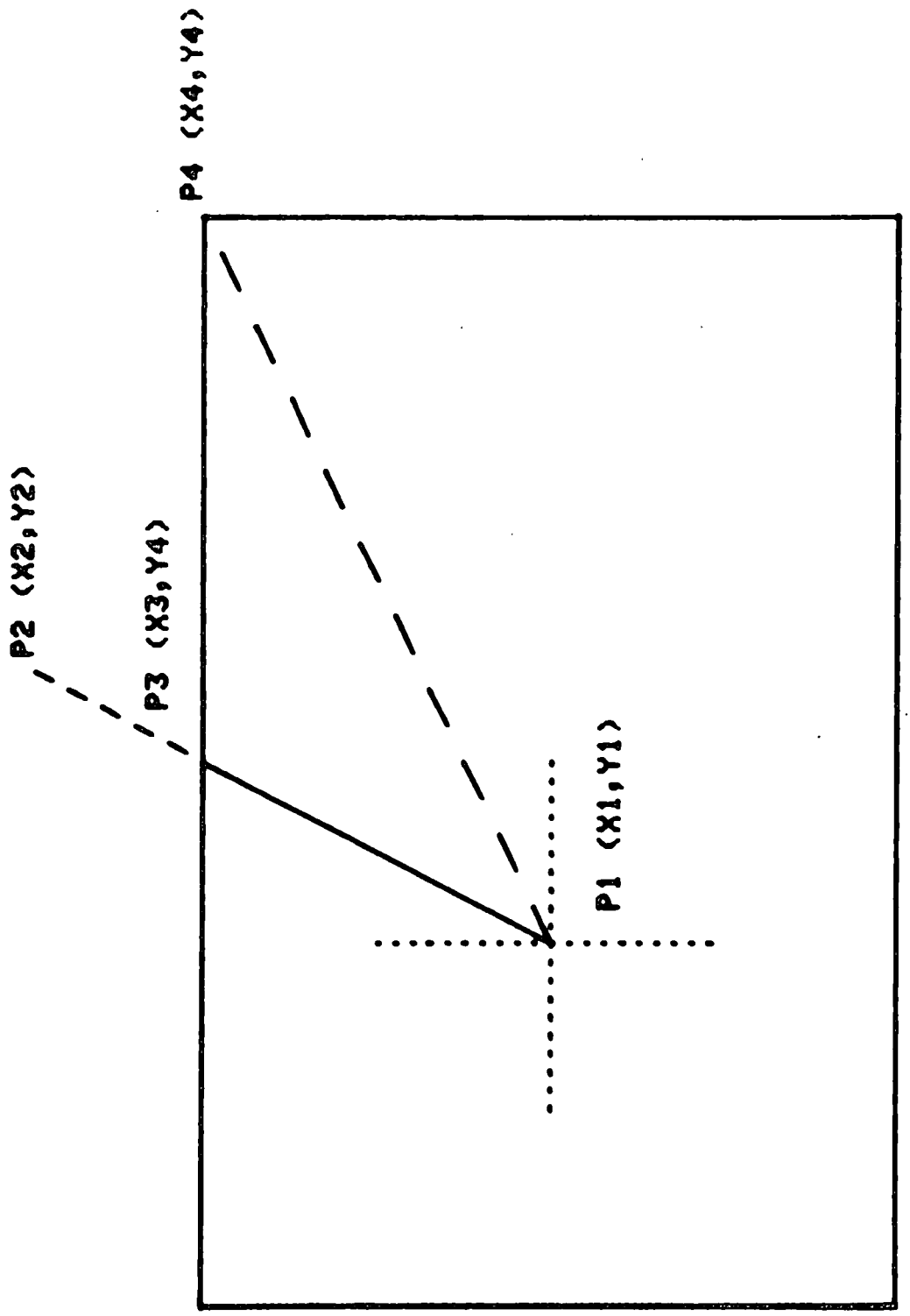


Figure 1. Exiting vector example condition 3

TITLE

Segmented Data Base and
Windowing Routine s

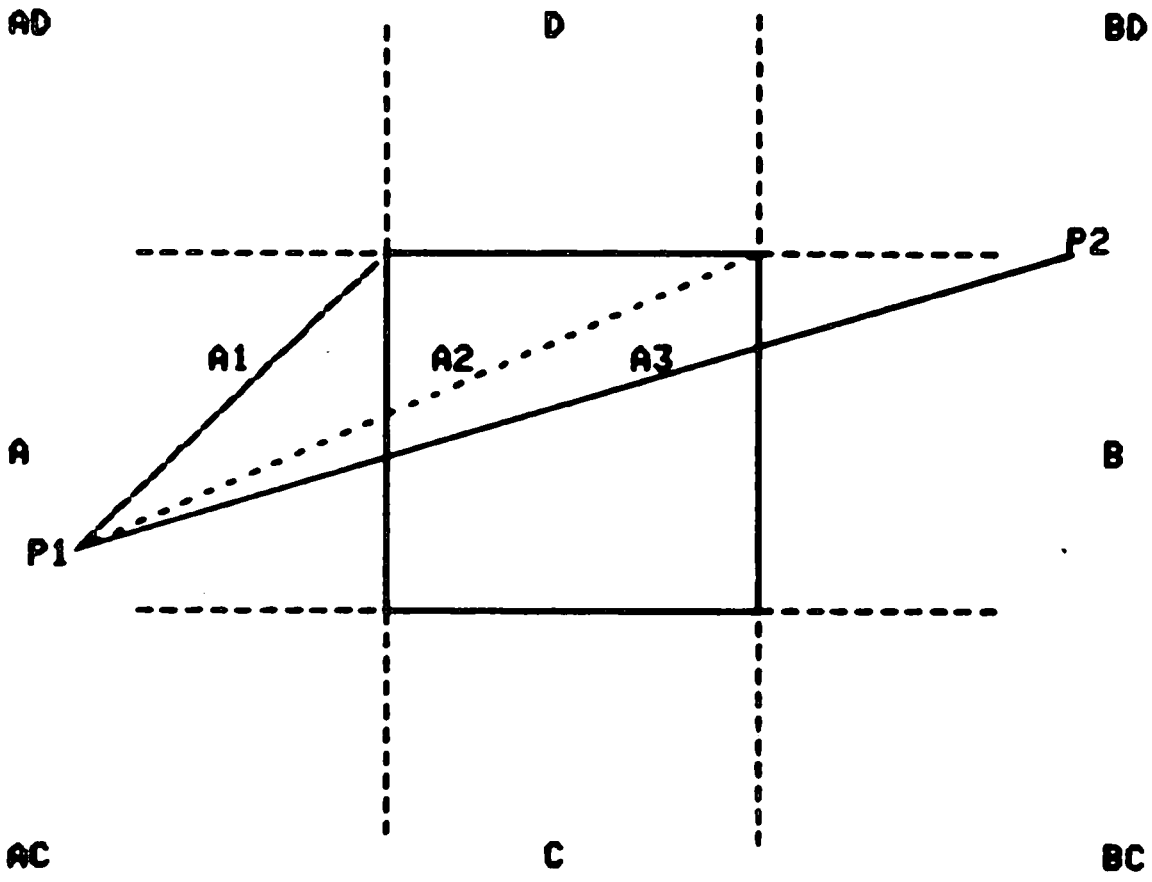


Figure 2. Traversing vector example Condition 4

TITLE

Segmented Data Base and
Windowing Routines

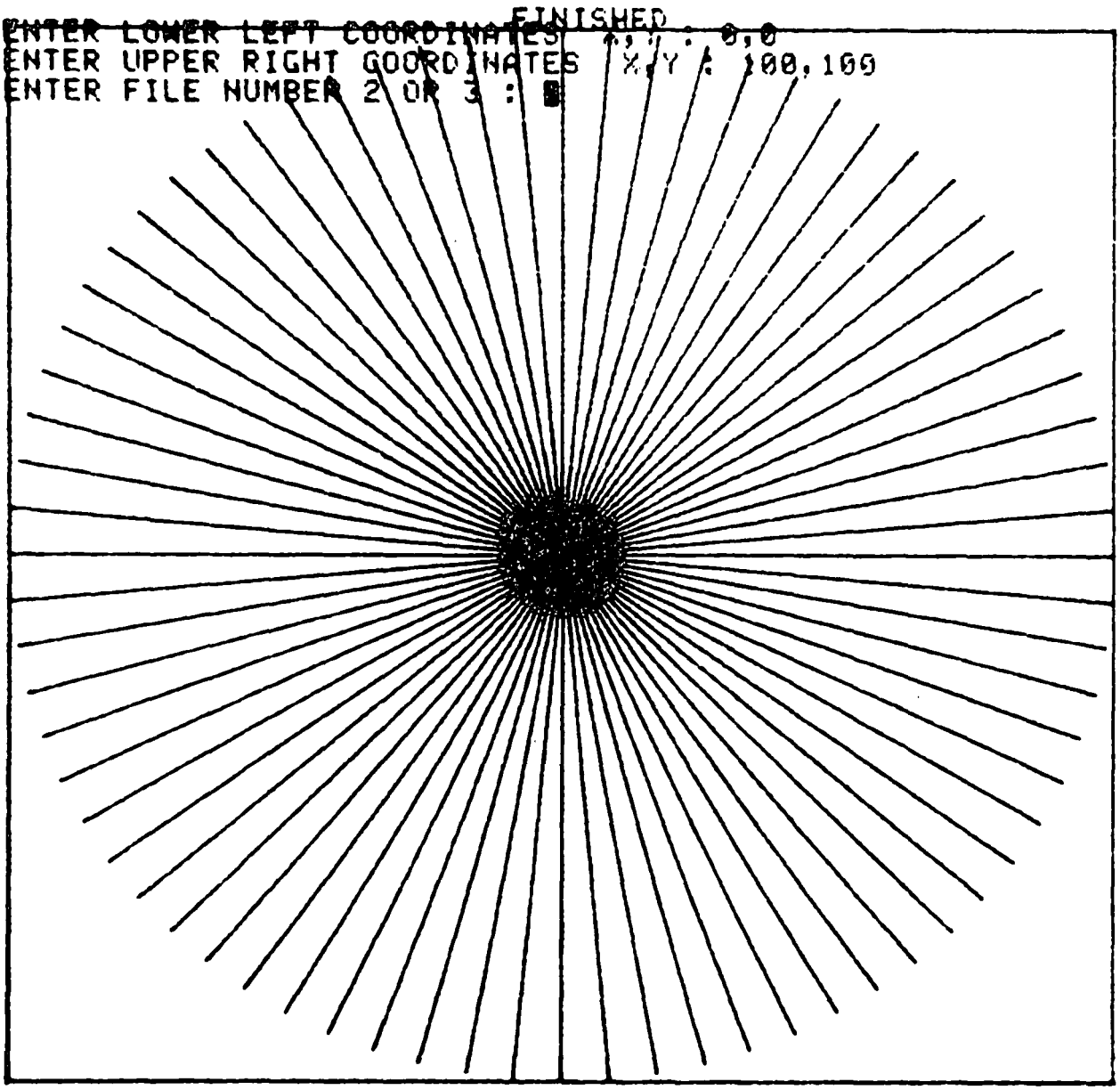


Figure 3.

TITLE

Segmented Data Base and
Windowing Routines

```
ENTER LOWER LEFT COORDINATES  X,Y : 20,20
ENTER UPPER RIGHT COORDINATES  X,Y : 80,80
ENTER FILE NUMBER 2 OR 3 : 2
```

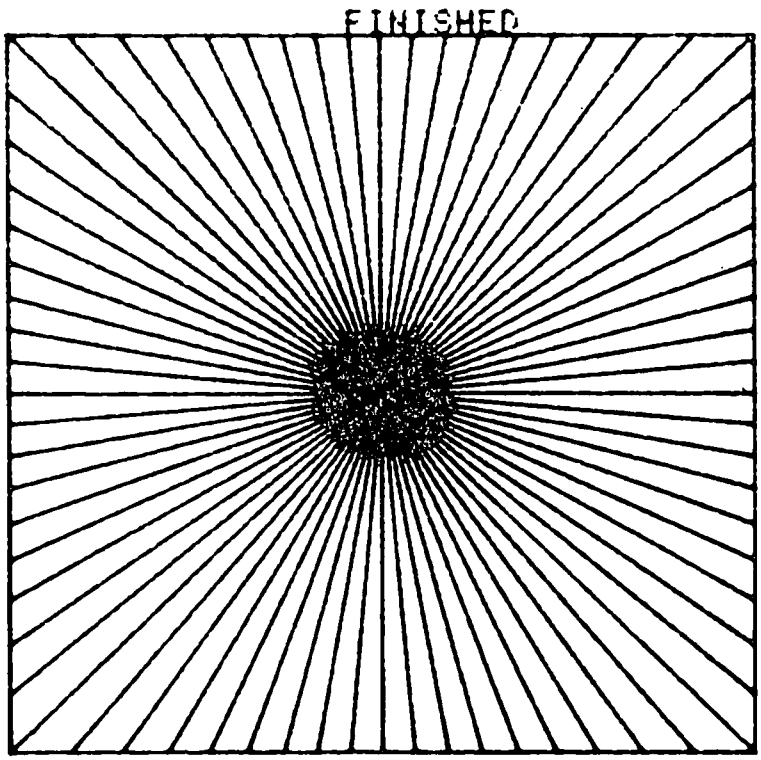


Figure 4.

TITLE

Segmented Data Base and
Windowing Routines

```
ENTER LOWER LEFT COORDINATES  X,Y : 40,40
ENTER UPPER RIGHT COORDINATES  X,Y : 60,60
ENTER FILE NUMBER 2 OR 3 : 2
```

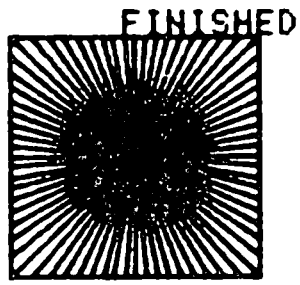


Figure 5.

TITLE

Segmented Data Base and
Windowing Routine s

```
ENTER LOWER LEFT COORDINATES  X,Y : 60,30
ENTER UPPER RIGHT COORDINATES  X,Y : 80,70
ENTER FILE NUMBER 2 OR 3 : 2
```

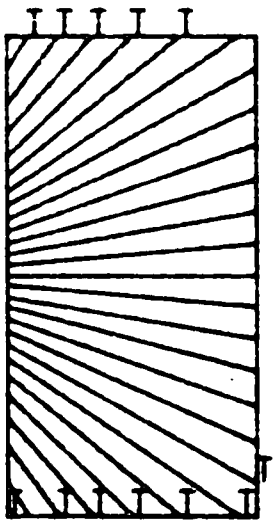


Figure 6.

TITLE

Segmented Data Base and
Windowing Routines

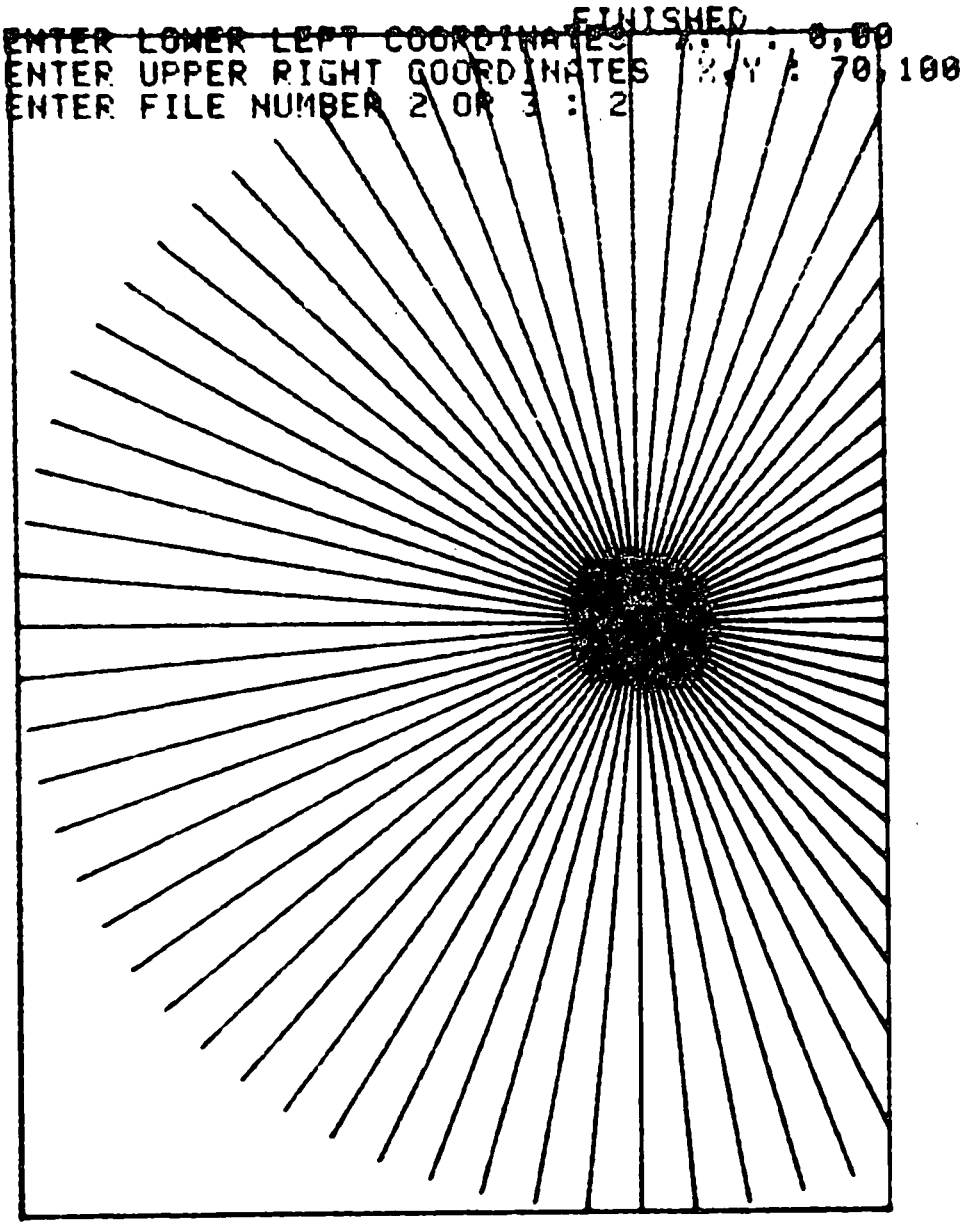


Figure 7.