the digital group
po box 6528 denver, colorado 80206 (303) 777-7133

MAXI-BASIC
Version 2.0

296-021-B-16

the digital group
po box 6528 denver, colorado 80206 (303) 777-7133

Maxi-Basic
Version 2.0

First Edition - April 1978

MAXI-BASIC — Version 2

INTRODUCTION

This manual describes MAXI-BASIC, an extended BASIC with such features as multiple-dimensioned arrays, strings, formatted output, and machine language subroutine capability, with plain english diagnostics.

MAXI-BASIC Version 2 has been created by adding an audio cassette data file capability to MAXI-BASIC Version 1.1. For the PHIMON user, PHIMAX (MAXI-BASIC Version 2 for PHIMON) offers program LOAD and SAVE from Phidecks as well as complete data file capability under PHIMON.

The user of MAXI-BASIC is assumed to be familiar with some version of BASIC. The purpose of this manual is not to teach BASIC but rather to define commands, statements and operating procedures of MAXI-BASIC.

AUDIO VERSION

This version is similar to Maxi 1.1 except for the following changes:

1) Scrolling buffer moved to top 4 pages of memory.
2) Audio data file routines on pages 12-14.

PHIMON VERSION (PHIMAX)

This version is based on Maxi 1.1, but numerous changes have been made to create a custom PHIMON Version:

1) Scrolling buffer is on pages 1-4.
2) Improved scroller.
3) PHIMON data file routines on pages 12-15.
4) PHIMON and audio cassette program LOAD and SAVE.

The starting address of PHIMAX is 5000 (octal) for a TVC-64 and 5004 (octal) for a TVC-32.

PRINTER

The printer handler is on pages 6 (octal) - 10 (octal) with 6000 (octal) being the initialize address and 6030 (octal) being the character print routine. To turn on the printer in BASIC press "CTRL P" on the keyboard. "CTRL O" turns off the printer. To turn on the printer in a BASIC program print a CHR$ (144) and a CHR$ (143) to turn it off.

SYSTEM SIZE

MAXI-BASIC and operating system reside in the first 64 pages of memory (13K). Therefore, the minimum system memory size should be 18K. MAXI-BASIC automatically searches for top of memory and adjusts itself for any size of continuous memory.

INPUTTING A PROGRAM

Every program line begins with a line number. Any line of text typed to MAXI-BASIC in command mode that begins with a digit is processed by the editor. There are four possible actions which may occur:

1. A new line is added to the program. This occurs if the line number is legal (range is 0 thru 65535) and at least one character follows the line number in the line.

2. An existing line is modified. This occurs if the line number matches the line number of an existing line in the program. That line is modified to have the text of the newly typed in line.

3. An existing line is deleted. This occurs if the typed-in line contains only a line number which matches an existing line in the program.

4. An error is generated. If the line number is out of range, or the line is too long, or the memory would become full, then an error message is generated and no other action is taken by MAXI-BASIC.

BLANKS

Blanks preceding a line number are ignored. The first non-digit in a line terminates the line number (even blanks). Multiple blanks are permitted anywhere in a line for indentation purposes, but not within reserved words or constants.

MULTIPLE PROGRAM STATEMENTS

Multiple program statements may appear on a single line, if separated by a (:) colon. A line number must appear only at the beginning of the first statement on the line.
NOTE: The colon (:) must be preceded by a space for correct operation.

TYPING MISTAKES

If a typing mistake occurs during the entering of any line of text to MAXI-BASIC, there are two possible corrective actions available:

When the user types an (@) at-sign character, MAXI-BASIC completely ignores all input on the current line being typed in, and types a carriage return. The correct line may then be typed to MAXI-BASIC.

When the user types a left-arrow (under-line or RUBOUT on some keyboards), MAXI-BASIC will backspace to the previously typed character.

COMPATIBILITY

Certain characters, when they appear in programs, are automatically translated into other characters. This is done to minimize the effort of converting programs written for other BASIC systems. In particular, left bracket ([), and right bracket (]), are converted to left parenthesis, and right parenthesis respectively. This conversion is not done within quoted strings in a program.

COMMANDS

RUN[optional line number]
Begin program execution either at the first line of the program or else at the optionally supplied line number.

LIST[optional line number],[optional second line number]
If no arguments are supplied, then print the entire existing program. If one line number is supplied, then print the specified line number. If two line numbers are supplied, then print the program in the region between the two line numbers. If one line number and a comma are typed with no second line number, then print the program from the specified line number to the end.

SCR
Delete (scratch) the existing program and data, in preparation for entering a new program.

REN[optional beginning value],[optional increment value]
Renumber the entire existing program. If the first argument is not supplied, then 10 is used as the initial statement renumber value. If the second argument is not supplied, then 10 is used as the increment value.

CLEAR
Clear all variables. This command deletes all arrays, strings and functions, and initializes all scalar variables to zero.

CONT
This command causes execution of a running BASIC program to continue after a STOP statement or after a CTRL-C stop.

LINE[number of characters]
This command defines the line length of the user terminal. No input line will be accepted longer than the specified value, and no output line will be printed longer than the specified value. The maximum value is 132. The initial value is 72.

SAVE
This command is used to save a program onto a cassette. See saving and loading programs.

LOAD
This command is used to load a program from cassette to memory. See saving and loading programs.

CONSTANTS

Magnitude range: .1E-63 thru .99999999+63

Constants appearring in programs are rounded to 8 digits if necessary. Internal representation of numbers is binary-coded-decimal.

NAMES

All user defined names are one or two characters long: a letter of the alphabet optionally followed by any digit. For example: A, Z0, and Q9 are legal names. The same name may be used to identify different values, as long as the values they identify are of different types. For example, it is possible to have a scalar variable named A1, an array named A1, a string named A1$ and functions named FNA1 and FNA1$. There is no relationship between these entities.

OPERATORS

Numeric: +, -, /, *, ↑ (or ^ on some keyboards)

Relational: =, <, >, <>, >=, =>, <=, =<
A relational operation gives a 1 (true) or 0 (false) result.

Boolean: AND, OR, NOT
A Boolean operand is true if non-zero, and false if zero. The result of a boolean operation is 1 or 0.

STATEMENTS

Only some **statements** listed below are accompanied by discussion. Consult the example programs in Appendix 1 for questions about the use of a particular type of statement.

LET
The LET is optional in assignment statements. Multiple assignments are not allowed. The statement A=B=0 assigns true or false to A depending on whether or not B equals 0.

```
100 LET A=A+1 : B(J)=B(J-1)
```

IF, THEN, ELSE
An IF statement may optionally have an ELSE clause. A THEN or ELSE clause may be a LET statement, a RETURN statement, another IF statement or a GOTO, for example. If either the THEN clause or the ELSE clause is a simple GOTO, then the GOTO reserved word may be optionally omitted.

```
100 IF A=B THEN 150 ELSE A=A-1
```

FOR, NEXT
FOR loops may be multiply nested. The optional STEP value may be positive or negative. It is possible to specify values such that the FOR loop will execute zero times. For example:

```
100 FOR J=5 to 4 : PRINT J : NEXT
```

A NEXT statement may optionally specify the control variable for the matching FOR statement, as a check for proper nesting.

GOTO
The GOTO statement is a direct branch to the designated line number.

```
100 GOTO 710
```

ON
The ON statement provides a multi-branched GOTO capability. For example:

```
100 ON J GOTO 500, 600, 700
```

will branch to 500, 600 or 700 depending on the value of J being 1, 2, or 3 respectively.

EXIT
The EXIT statement is identical to a GOTO except that it has the effect of terminating any active FOR loops and reclaiming the associated internal stack memory. It should be used for branching out of a FOR loop.

```
100 IF A (J)=100 THEN EXIT 320
```

STOP
The STOP statement halts execution of the program and displays the message "STOP IN LINE XXX". After a STOP has been encountered, the program can be continued starting at the next line by typing CONT.

```
100 STOP
```

END

The END statement also halts the execution of the program. However, unlike STOP, there is no way to continue from an END statement. If the END statement is the last line number of the program, it may be optionally omitted.

```
100 END
```

REM

The REM statement is used to annotate the program. Any REM statement is ignored by the MAXI-BASIC interpreter.

```
100 REM THIS PROGRAM CALCULATES PI
```

READ, DATA

The READ and DATA statements allow the user to input pre-determined data into a program. The READ statement transfers data named in the DATA statement into the variables or arrays which have been named by the READ statement.

```
100 DATA 12.17, "VOLTS", 2.4E09, "OHMS"
110 READ V, V$, O, O$
```

RESTORE

The RESTORE statement may optionally include a line number, specifying where the READ pointer is to be restored to. In the absence of the optional line number, the READ pointer is set to the first line of the program.

```
100 RESTORE 75
```

INPUT
INPUT1

The INPUT or INPUT1 statement may optionally specify a literal string which is typed on the terminal as a prompt for the input instead of a question mark. To inhibit the echoing of the carriage return at the end of user input, use the INPUT1 statement.

```
100 INPUT "TYPE VALUE: ",V
```

GOSUB, RETURN

The GOSUB statement branches the program to a subroutine with the starting line number specified in the GOSUB statement. The RETURN statement is the last line of the subroutine, and branches the program to the line following the GOSUB statement

```
100 GOSUB 1300
1350 RETURN
```

PRINT

The PRINT statement may include a list of expressions, variables, or constants separated by commas (,), or semicolons (;). Note that if the list of variables is terminated by a comma, or semicolons then a carriage return is not typed. A comma separator will output five spaces between variables. A semicolon separator will output no spaces between variables. The PRINT "" statement will cause a carriage return to be printed. All values are printed in free format, unless formatting is specified. If a value will not fit on the current output line, then it is printed on the next output line. Advancement of the printer to a specified output position may be accomplished with the TAB function. Formatting may be accomplished by including a "format string" in a print statement (see below). A # sign is interpreted as the word PRINT.

```
100 PRINT "PT="; P: PRINT"": PRINT D, 17.5, E
```

FILL

This statement permits filling a specified byte in the computer memory with a given expression value. For example, FILL 100, J+3 will fill memory byte 100 with J+3.

```
100 FILL 100, J+3
```

OUT

This instruction permits doing an 8080 or Z-80 OUT instruction. For example, OUT 5, 3 will perform an OUT 5 instruction with 3 in the 8080 or Z-80 accumulator.

```
100 OUT 5, 3
```

DATA FILE ROUTINES

The data file routines in the two versions of Maxi-2 have virtually the same calling sequence, allowing most data file programs to run under either version. Any differences between the two will be mentioned in the description below.

Buffers

File commands are passed to the file routines through a Maxi-BASIC string variable of 256 characters. This string is initialized by filling it with 256 "P"s and executing a call to the data file initialize routine (decimal address = 2560).

Data buffers for the file routines are also Maxi-BASIC string variables. The user may specify up to 4 data buffers, corresponding to each of the Phideck drives. Each data buffer may be any size as long as it is a multiple of 256 characters (256, 512, 768, 1024, etc.). To initialize these strings, dimension them to the desired length, fill them with 256 of the number corresponding to their drive number (256 "0"s, for example) and execute a call to the data file initialize routine (decimal address = 2560). This call should be the same as the call for the command buffer described above.

See the sample programs for examples of setting up the various buffers.

File Commands

The command buffer is divided into four sections of 64 characters each. These four sections correspond to the four cassette drives 0-3. Characters 1-64 are for unit 0, characters 65-128 are for unit 1 and so on. A recommended method of putting the command in the buffer is shown here:

```
P$(N* 64+1,N* 64+64)="(DESIRED COMMAND STRING)"
```

where N=unit number

Once the command is in the correct portion of the command buffer, it is executed by doing a call to the main data file routine (decimal address = 2675):

```
Q7=CALL (2675, N)
```

where N=unit number and Q7=value returned by the file function.

OPEN Command

The first command normally used when using the file routines is OPEN. OPEN has two forms; opening a file for

OUTPUT (creating a new data file) and opening a file for INPUT (preparing an existing data file for reading). The format of the OPEN command is:

"FILE #N OPEN, (OUTPUT or INPUT), NAME.EX"

where N=unit number and NAME.EX is the name of the file to be created (OPEN, OUTPUT) or the name of the file to be read (OPEN, INPUT). The name is ignored in the audio version.

The value returned by the OPEN (OUTPUT) command is the number of blocks available for use on the device. The value returned by the OPEN (INPUT) command is the number of blocks in the file opened for reading. If the file is not found a Ø is returned. In the audio version 65535 is always returned.

If a file is opened with a .WK extension it becomes a work file. Work files ignore directories and start at block Ø of the tape. These may be used on blank tapes as scratch files. Trying to create a work file on a tape with a directory will destroy that tape's directory.

WRITE Command

The WRITE command is used to write data from a data buffer into a data file that has been opened for OUTPUT. The data is written from the data buffer corresponding to the unit number. Writing continues until either the dimensioned length of the data buffer string, the space on the tape, or the optional write block count is exhausted. The format of the WRITE command is:

"FILE#N WRITE (OPTIONAL WRITE BLOCK COUNT)"

where N=unit number. If specified, the write block count must be in parentheses. If a variable is desired as the write block count, the STR$ function must be used to convert the number into a string:

"FILE#N WRITE ("+STR$(W)+")"

The value returned by the WRITE command is the number of blocks actually written into the file. If a Ø is returned, the file has been exhausted.

The audio version will clear the screen during a WRITE and display the message:

START RECORDING #N (SPACE)

Start the tape and press a space to begin writing.

READ Command

The READ command is used to read data from a file that has been opened for INPUT. The data is read into the data buffer corresponding to the unit number. Reading continues until either the dimensioned size of the string, or the data file is exhausted. The format of the READ command is:

"FILE#N READ (OPTIONAL OFFSET)"

where N=unit number. If a number enclosed in parentheses appears in the command string, that number becomes the offset block pointer to the file for random access files. A Ø will reset a file to the beginning. If a variable is desired as the offset, the STR$ function must be used to convert the number into a string as in the WRITE command. The offset option is not available in the audio version.

The value returned by the READ command is the number of blocks actually read. If a Ø is returned, the file has been exhausted and no reading was done. The length of the data buffer string is set to the length of the data that was actually read in.

The audio version will clear the screen during a read and display the message:

START READING #N (SPACE)

Start the tape and press a space to begin reading. Reading will continue until trailer is reached or the string length is exhausted.

CLOSE Command

The CLOSE command is used to close any open file. It must be used before reopening another file on that device because only one file is allowed open on a device at a time. Also, when creating a new file, the directory of that device is updated only during a CLOSE. The format of the CLOSE command is:

"CLOSE#N"

where N=unit number.

USE Command

The USE command allows one data buffer to be used by more than one device. The format of the USE command is:

"FILE#N USE#M"

where N is the unit that is to use unit M's data buffer. The value returned by the USE command is the memory address of the data buffer specified by M in the format above. This is useful in determining the address of a data buffer.

ARRAYS

Arrays may be dimensioned with any number of dimensions, limited only by available memory, e.g.,

100 DIM A(1), B7 (5,2,3,4,5,6)

Array indexing starts at element Ø. Array A in the above example actually has two elements, A(Ø) and A (1). Use of an undimensioned array causes automatic dimensioning to a one dimension, 10 element array. Arrays may not be re-dimensioned within a program.

STRINGS (see Appendix 1)

Strings of **8-bit characters** may be dimensioned to **any size**, limited only by available memory, e.g.,

100 DIM A$ (1),A1$ (10000)

Note that a string name is a variable name followed by a ($) dollar sign. Substrings may be accessed as A$ (N,M) which is the substring of characters N thru M. For example, if A$ is "ABCDEF" then A$ (3,5) is "CDE". Alternatively, A$ (N) identifies the substring including characters N thru the last character in the string. The concatenation operator is a plus sign.

If an assigned value is larger than the destination string or substring, then it is truncated to fit. If the value assigned to a substring is shorter than the substring, then the extra characters of the substring are left unmodified. A string variable used before being DIMensioned is given the default dimension of 10. Strings may not be redimensioned within a

program. Strings may not be modified until they have been defined by a LET A$= or INPUT A$ statement.

Strings, substrings and string expressions may be used in conjunction with: LET, READ, DATA, PRINT, IF, and INPUT statements. The string IF statement does alphabetic comparisons when the relational operators are used, e.g.

```
100 IF A$+B$<"SMITH" THEN 50
```

When string variables are INPUT, they must not be quoted. When strings appear in data statements, they must be quoted.

NOTE: A string array is initialized as follows:
(Where N = Length of string).

```
For X=1 to N : A$=A$+" " : NEXT X
```

USER DEFINED FUNCTIONS

User-defined functions (either of type string or numeric) may be 1-line or multiple line functions. There may be any number of numeric arguments. Parameters are "local" to a particular call of a function. That is, the value of the variable is not affected outside of the execution of the function.

Functions are defined before execution begins (at RUN time), so definitions need not be executed, and functions may be defined only once.

Multiple line functions must end with a FNEND statement. A multiple-line function returns a value by executing a RETURN statement with the value to be returned, for example:

```
100 DEF FNA (X, Y, Z)
200 IF Z=1 THEN RETURN X
300 X=Y* Z+X* 3
400 RETURN X
500 FNEND
600 PRINT FNA (1,2,X+Y)
```

BUILT IN FUNCTIONS

FREE (0)	returns number of bytes remaining in free storage
ABS (expr)	returns the absolute value of the expression
SGN(expr)	returns 1,0, or -1 if the value is +, 0, or -
INT (expr)	returns the integer portion of the expression value
LEN (string name)	returns the length of the specified string
CHR$ (expr)	returns a string with the specified character
VAL (string expr)	returns the numeric value of the string
STR$ (expr)	returns a string with the specified numeric value
ASC (string name)	returns ASCII code of the first character in the string
SIN (expr)	returns the SINE of the expression
COS (expr)	returns the COSINE of the expression
RND (expr)	returns a random number between 0 and 1
LOG (expr)	returns the natural log of the expression
EXP (expr)	returns the value of e raised to the specified power
SQRT (expr)	returns the positive square root of the expression
CALL (expr, optional expr)	see below
EXAM (expr)	return contents of addressed memory byte
INP (expr)	return result of 8080 or Z-80 IN to specific port

MACHINE LANGUAGE SUBROUTINE INTERFACING

The built-in function CALL takes a first argument which is the decimal address of a machine language subroutine to call. The optional second argument is a value which is converted to an integer and passed to the machine language subroutine in DE. The CALL function returns as a value the integer which is in HL when the machine language subroutine returns.

NOTE: CALL is a function and not a verb. Therefore:

```
10 LET X=CALL(1234) and not
10 CALL (1255)
```

FORMATTED OUTPUT

If no format string is present in a PRINT statement, then all numeric values will be printed in the "default format". (The default format is initially set to be free format). A format string appears anywhere in the print list and must begin with a per cent (%) character, e.g.

```
PRINT %$10F3,J
```

A format string consists of optional format characters followed optionally by a format specification. The format characters are:

- C place commas to the left of decimal point as needed
- $ put a dollar sign to the left of value
- Z suppress trailing zeroes
- ? make this format string the default specification

Format specifications (similar to FORTRAN) are:

nFm* F-format. The value will be printed in an n-character field, right justified, with m digits to the right of the decimal point.

nI* I-format. The value will be printed in an n-character field, right justified, if it is an integer. (Otherwise an error message will occur.)

nEm* E-format. The value will be printed in scientific notation in an n-character field, right justified, with m digits to the right of the decimal point.

All printed values are rounded if necessary. A null format string will print values in free format.

*n includes preceeding + or -, and all commas and dollar signs

The general form is PRINT % XY ; I

Where

X=any combination (or none) of C, $, and Z
Y=any format specification
I=variable or constant
and where the separating comma or semicolon is as in any non- formatted PRINT statement
i.e. PRINT %C$Z12F3; 1234.5609
$1,234.561
PRINT %C$Z12F2; 1234.5600
$1,234.56

CONTROL-C

Typing the CTRL-C character (ETX on some keyboards) has the effect of prematurely interrupting MAXI-BASIC from whatever it is doing. If a LIST is in progress, the listing will be terminated at the completion of the output of the current line. If a RUN or CONT is in progress, then execution will stop after the completion of the currently executing statement, and a CONT will continue executing the program.

DIRECT STATEMENTS

When MAXI-BASIC is in command mode, certain statements may be typed for immediate execution. This is typically used for examining the values of certain variables to diagnose a programming error. Note that a pound sign (#) may be used as a shorthand way of typing the PRINT reserved word. No direct statement is permitted which transfers control to the BASIC program. Also, DATA, DEF, FOR, NEXT, INPUT, and REM are forbidden.

SAVING AND LOADING PROGRAMS

To save a current program onto cassette, the user should turn on his recorder (on record) and type SAVE(cr). The CRT screen will indicate that the tape is being written. When finished, the screen will return with the ready message.

To load a program from the cassette, the user should start playing the cassette. When the leader tone is heard, type LOAD(cr). The CRT screen will indicate that the tape is being read. When finished the screen will return with the READY message.

To LOAD or SAVE programs from Phidecks use this format:

LOAD#N NAME or SAVE#N NAME

where N=unit number and NAME is the name of the BASIC program file. A .BA extension is automatically added.

All programs written on the Z-80 are usable on the 8080 version of MAXI-BASIC and vice versa. Also, cassettes written with level 1 MAXI-BASIC will be upward compatible on all later levels of MAXI-BASIC.

SUMMARY

Maxi-BASIC, with these new data file routines, is a powerful tool for the BASIC programmer. Unlike many standard forms of BASIC data file routines, these allow the user to create and read virtually any type of data file. Using the CHR$ and ASC functions of BASIC, work files, and the random access capability, the user has total control of data files.

APPENDIX

SAMPLE PROGRAMS

Program 1 is a demo program that copies a file from unit Ø to unit 1 using the file routines in their standard form.

Program 2 uses a Maxi-BASIC user defined function to simplify the use of the file routines and reduce the number of lines in Program 1. Functionally, Program 2 is identical to Program 1. The user may use this function to simplify the file routine if he wishes.

Program 3 demonstrates one method of storing numbers in data files. This program allows the user to input a series of numbers (terminated by a -1) to be saved in a file named "NUMBER.DA". Then the program reopens the file for input and prints out the numbers stored there.

LIST

Program 1

```
10  REM ** PROGRAM TO COPY A FILE FROM UNIT Ø TO UNIT 1
20  REM **
30  REM ** DIMENSION AND SET UP BUFFERS
40  DIM P$(256),B$(1024)
50  P$="P" : B$="Ø"
60  FOR X=1 TO 8
70    P$=P$+P$
80    B$=B$+B$
90    NEXT X
100 Q7=CALL(2560)
110 REM ** MAKE UNIT 1 USE BUFFER FOR UNIT Ø
120 P$(65,128)="FILE#1 USE#Ø"
130 Q7=CALL(2675,1)
140 PRINT
150 INPUT "NAME OF FILE ? ",N$
160 REM ** OPEN FILE ON UNIT Ø FOR READING
170 REM ** L WILL EQUAL NUMBER OF BLOCKS IN FILE
180 P$(1,64)="FILE#Ø OPEN,INPUT,"+N$
190 L=CALL(2675,Ø)
200 IF L=Ø THEN 440
210 PRINT
220 INPUT "NEW NAME IF ANY ? ",N1$
230 IF N1$<>"" THEN N$=N1$
240 REM ** OPEN FILE ON UNIT 1 FOR WRITING
250 REM ** L1 WILL EQUAL NUMBER OF AVAILABLE BLOCKS
260 P$(65,128)="FILE#1 OPEN,OUTPUT,"+N$
270 L1=CALL(2675,1)
280 IF L>L1 THEN 460
290 REM ** READ NEXT BUFFER-FULL
300 REM ** B WILL EQUAL BLOCKS READ
310 P$(1,64)="FILE#Ø READ"
320 B=CALL(2675,Ø)
330 IF B=Ø THEN 390
340 REM ** WRITE B BLOCKS ONTO UNIT 1
350 P$(65,128)="FILE#1 WRITE ("+STR$(B)+")"
360 Q7=CALL(2675,1)
370 GOTO 320 : REM ** DO NEXT BUFFER LOAD
380 REM ** CLOSE FILES
390 P$(1,64)="FILE#Ø CLOSE"
400 P$(65,128)="FILE#1 CLOSE"
410 Q7=CALL(2675,Ø)
420 Q7=CALL(2675,1)
430 GOTO 140
440 PRINT "FILE NOT FOUND"
450 GOTO 140
460 PRINT "NO ROOM"
470 GOTO 140
```

Program 2

```
LIST

10  REM ** PROGRAM TO COPY A FILE FROM UNIT Ø TO UNIT 1
20  REM **
30  REM ** DIMENSION AND SET UP BUFFERS
40  DIM P$(256),B$(1024)
50  DIM C$(64)
60  P$="P" : B$="Ø"
70  FOR X=1 TO 8
80    P$=P$+P$
90    B$=B$+B$
100    NEXT X
110  Q7=CALL(2560)
120  REM ** USER DEFINED FUNCTION TO DO FILE CALLS
130  DEF FNF(C$)
140  U=ASC(C$(6,6))-48
150  P$(U*64+1,U*64+64)=C$
160  RETURNCALL(2675,U)
170  FNEND
180  REM ** MAKE UNIT 1 USE THE BUFFER FOR UNIT Ø
190  Q7=FNF("FILE#1 USE#Ø")
200  PRINT
210  INPUT "NAME OF FILE ? ",N$
220  REM ** OPEN FILE FOR READING, L=BLOCKS IN FILE
230  L=FNF("FILE#Ø OPEN,INPUT,"+N$)
240  IF L=Ø THEN 410
250  PRINT
260  INPUT "NEW NAME IF ANY ? ",N1$
270  IF N1$<>"" THEN N$=N1$
280  REM ** OPEN FILE FOR WRITING, L1=AVAILABLE BLOCKS
290  L1=FNF("FILE#1 OPEN,OUTPUT,"+N$)
300  IF L>L1 THEN 430
310  REM ** READ A BUFFER-FULL, B=BLOCKS READ
320  B=FNF("FILE#Ø READ")
330  IF B=Ø THEN 380
340  REM ** WRITE B BLOCKS ONTO OUTPUT DEVICE
350  Q7=FNF("FILE#1 WRITE("+STR$(B)+")")
360  GOTO 300
370  REM ** CLOSE FILES
380  Q7=FNF("FILE#Ø CLOSE")
390  Q7=FNF("FILE#1 CLOSE")
400  GOTO 200
410  PRINT "FILE NOT FOUND"
420  GOTO 200
430  PRINT "NO ROOM "
440  GOTO 200
```

LIST

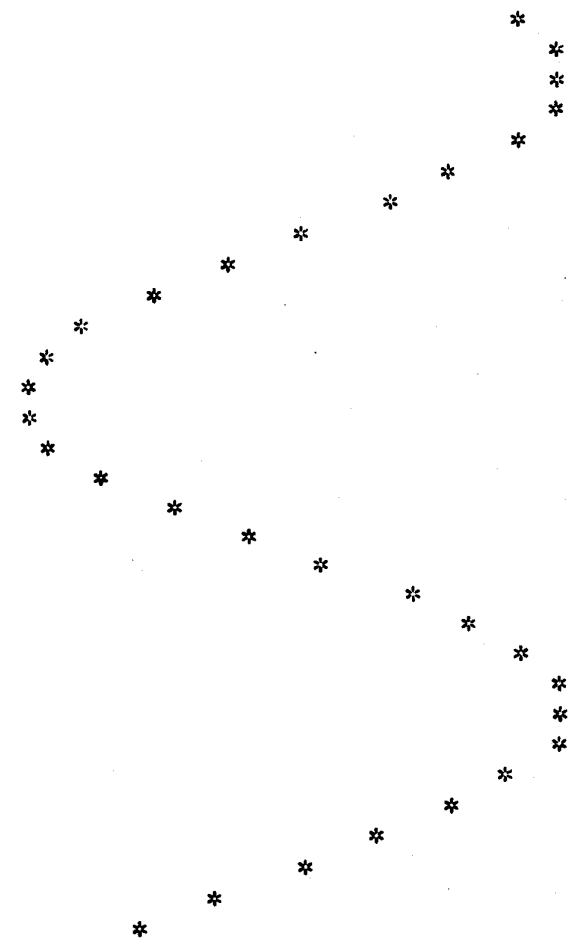
Program 3

```
10  REM ** SAMPLE PROGRAM TO CREATE DATA FILE OF NUMBERS
20  REM ** AND READ THEM BACK
30  REM **
40  REM ** DIMENSION AND SET UP STRINGS
50  DIM P$(256),B$(256)
60  DIM C$(64),N$(20)
70  P$="P" : B$="1"
80  FOR X=1 TO 8
90    P$=P$+P$
100    B$=B$+B$
110    NEXT X
120  Q7=CALL(2560)
130  REM ** USER FUNCTION TO DO FILE CALLS
140  DEF FNF(C$)
150  U=ASC(C$(6,6))-48
160  P$(U*64+1,U*64+64)=C$
170  RETURNCALL(2675,U)
180  FNEND
190  REM ** OPEN FILE FOR WRITING - Q7=AVAILABLE BLOCKS
200  Q7=FNF("FILE#1 OPEN,OUTPUT,NUMBER.DA")
210  PRINT "INPUT NUMBERS TO BE SAVED"
220  PRINT "-1 TERMINATES LIST"
230  PRINT
240  B$=""
250  INPUT N
260  IF N=-1 THEN 320
270  REM ** ADD NEW STRING TO BUFFER, WRITE IF OVERFLOW
280  N$=STR$(N)
290  IF LEN(B$)+LEN(N$)>255 THEN 320
300  B$=B$+N$
310  GOTO 250
320  P$=B$+CHR$(1) : REM ** MARK END OF BUFFER WITH A 1
330  REM ** WRITE BUFFER, Q7=ACTUAL BLOCKS WRITTEN
340  Q7=FNF("FILE#1 WRITE")
350  IF Q7=0 THEN 530
360  B$=""
370  IF N<>-1 THEN 300
380  REM ** CLOSE FILE (PUTS FILE INTO DIRECTORY)
390  Q7=FNF("FILE#1 CLOSE")
400  REM ** RE-OPEN FILE FOR READING - Q7=BLOCKS IN FILE
410  Q7=FNF("FILE#1 OPEN,INPUT,NUMBER.DA")
420  REM ** READ NEXT BUFFER-FULL - Q7=ACTUAL BLOCKS READ
430  Q7=FNF("FILE#1 READ")
440  IF Q7=0 THEN 540
450  REM ** GET NUMBERS OUT OF BUFFER
460  I=1
470  N=VAL(B$(I+1,256))
480  PRINT N;
490  I=I+1
500  IF B$(I,I)=" " THEN 470
510  IF ASC(B$(I,I))<>1 THEN 490
520  GOTO 430
530  PRINT "NO MORE ROOM"
540  END
```

```
LIST

100  REM    PRINT A VERTICAL SINE WAVE
110  REM
120  FOR J=1 TO 10 STEP .3
130    S=INT(15*(SIN(J)))
140    PRINT TAB(15+S);"*"
150    NEXT J
160  END
REN 10,2
LIST

10   REM    PRINT A VERTICAL SINE WAVE
12   REM
14   FOR J=1 TO 10 STEP .3
16     S=INT(15*(SIN(J)))
18     PRINT TAB(15+S);"*"
20     NEXT J
22   END
RUN
```



```
LIST

100  REM  A NUMERIC SORT PROGRAM
110  REM
120  DIM A(15)
130  PRINT "INPUT FIFTEEN VALUES, ONE VALUE PER LINE"
140  FOR J=1 TO 15
150    INPUT A(J)
160    NEXT J
170  REM  DO EXCHANGE SORT UNTIL ALL IN ORDER
175  F=0 : REM  THIS FLAG USED TO SIGNAL WHETHER DATA IN ORDER YET
180  FOR J=2 TO 15
190    IF A(J-1)<=A(J) THEN 220
200    T=A(J) : A(J)=A(J-1) : A(J-1)=T : REM  EXCHANGE A(J) AND A(J-1)
210    F=1 : REM  SET FLAG
220    NEXT
230  IF F=1 THEN 175 : REM  LOOP IF EXCHANGES HAPPENED
240  PRINT "SORTED ARRAY: ";
250  FOR J=1 TO 15 : PRINT A(J);" "; : NEXT
RUN

INPUT FIFTEEN VALUES, ONE VALUE PER LINE
?123
?22
?-37
?0
?2
?-54
?31
?8
?-9.4
?1.54
?-3.8
?36
?21
?-43
?213
SORTED ARRAY:  -54  -43  -37  -9.4  -3.8  0  1.54  2  8  21  22
 31  36  123  213
```

```
LIST

10  REM  TO INITIALIZE A STRING VARIABLE
20  REM  USE THE FOLLOWING ROUTINE
30  REM BEFORE ATTEMPTING TO ALTER THE STRING
40  DIM A$(N) : REM  WHERE N=STRING LENGTH
50  FOR X=1 TO N
60    LET A$=A$+" "
70    NEXT X


LIST

10  REM  CHARACTER SORT
20  REM  EXAMPLE USING STRINGS AND FUNCTION
30  DIM A$(72)
40  INPUT "TYPE A STRING OF CHARACTERS: ",A$
50  IF LEN(A$)=0 THEN 40 : REM MAKE SURE SOMETHING WAS ENTERED
60  IF FNA(LEN(A$))=1 THEN 60 : REM CALL FNA UNTIL IT RETURNS A ZERO VALUE
70  PRINT "SORTED ARRAY: ";A$
80  END
90  DEF FNA(N) : REM  CHARACTER SORT
100  REM  RETURN 0 IF A$ SORTED, ELSE RETURN 1
110  LET F=0
120  FOR J=2 TO N
130    IF A$(J-1,J-1)<=A$(J,J) THEN 160
140    T$=A$(J,J) : A$(J,J)=A$(J-1,J-1) : A$(J-1,J-1)=T$
150    F=1
160    NEXT J
170  RETURNF
180  FNEND
RUN

TYPE A STRING OF CHARACTERS: DIGITAL GROUP
SORTED ARRAY:  ADGGIILOPRTU
```

```
LIST

10  REM  TV DESIGNER
20  REM
30  FOR A=1 TO 5 : PRINT "" : NEXT A
40  PRINT TAB(8);"TV DESIGNER"
50  PRINT
60  INPUT "ENTER LINE FREQUENCY ",L
70  INPUT "ENTER HORZ HOLDOFF RATIO ",H
80  INPUT "ENTER VERT HOLDOFF RATIO ",V
90  INPUT "ENTER CHARACTERS/LINE ",C
100  INPUT "ENTER ROWS OF CHARACTERS ",R
110  INPUT "ENTER HORZ PEL/CHARACTER ",P
120  INPUT "ENTER H PEL SPACES ",S
130  INPUT "ENTER LINES/CHARACTER ",X
140  INPUT "ENTER LINES DURING JUMP ",J
150  REM
160  REM  CALCULATIONS
170  REM
180  REM  CORRECTED TOTAL LINES
190  LET A=INT(((X+J)*R*V)+.5)
200  REM  CORRECTED V HOLDOFF RATIO
210  LET V=A/((X+J)*R)
220  REM  HORZ FREQUENCY
230  LET B=A*L
240  REM  VISIBLE H PEL
250  LET D=C*(P+S)
260  REM  CORRECTED EFFECTIVE H PEL
270  LET E=INT((D*H)+.5)
280  REM  CORRECTED HORZ HOLDOFF RATIO
290  LET T=E/D
300  REM  PEL RATE IN MHZ
310  LET F=E*B/1000
320  REM  TIME/PEL IN NS
330  LET G=(1/F)*1000000
340  REM  TIME/CHARACTER IN NS
350  LET I=G*(P+S)
360  REM  HORZ LINE TIME IN US
370  LET K=(1/B)*1000000
380  REM  HORZ BLANKING TIME IN US
390  LET M=K*((E-D)/E)
400  REM  FRAME TIME IN MS
410  LET N=(1/L)*1000
420  REM  VERT BLANKING IN MS
430  LET O=(N*((A-((X+J)*R))/A))
440  REM  TOTAL VISIBLE PEL
450  LET S=D*(P+S)
460  REM  TOTAL VISIBLE LINES
470  LET U=(X+J)*R
480  PRINT "H FREQ ";INT(B);"    XTAL ";INT(F)
490  PRINT "VIS H PEL ";D;"   TOTAL H PEL ";E
500  PRINT "VIS LINES ";U;"   TOTAL LINES ";A
510  PRINT "H RATIO ";%4F2;T;"    V RATIO ";%4F2;V
520  PRINT "NS/PEL ";%4F1;G;"    NS/CHAR ";%5F1;I
530  PRINT "H LINE ";%6F2;K;"    H BLANK ";%6F2;M
540  PRINT "FRAME ";%5F2;N;"    V BLANK ";%6F2;O
550  PRINT TAB(10);"*******"
```

```
RUN

        TV DESIGNER

ENTER LINE FREGUENCY 60
ENTER HORZ HOLDOFF RATIO 1.5
ENTER VERT HOLDOFF RATIO 1.25
ENTER CHARACTERS/LINE 64
ENTER ROWS OF CHARACTERS 16
ENTER HORZ PEL/CHARACTER 7
ENTER H PEL SPACES 1
ENTER LINES/CHARACTER 12
ENTER LINES DURING JUMP 1
H FREQ  15600    XTAL  11980
VIS H PEL  512    TOTAL H PEL  768
VIS LINES  200    TOTAL LINES  260
H RATIO 1.50    V RATIO 1.25
NS/PEL 83.5    NS/CHAR 667.7
H LINE  64.10    H BLANK  21.37
FRAME 16.67    V BLANK   3.33
          ********
```

READER'S COMMENTS

The Digital Group would like to improve the quality and usefulness of this publication. To do this effectively, we need user feedback — your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments.

NAME: _____ DATE: _____

STREET: _____

CITY: _____ STATE: _____ ZIP: _____

TELEPHONE NUMBER: _____

Please send this form to:

SOFTWARE DEVELOPMENT/MAXI-BASIC
DIGITAL GROUP INC.
P. O. BOX 6528
DENVER, COLORADO 80206