UTILITY PROGRAMS

for

Advan Language Designs

BASIC COMPILER

UTILITY PROGRAMS

for

ADVAN LANGUAGE DESIGNS

BASIC COMPILER

WARNING

DISCLAIMER OF WARRANTY

THIS SOFTWARE AND MANUAL ARE SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OR MERCHANTABILITY. THE SELLER'S SALESPERSONS MAY HAVE MADE STATEMENTS ABOUT THIS SOFTWARE. ANY SUCH STATEMENTS DO NOT CONSTITUTE WARRANTIES AND SHALL NOT BE RELIED ON BY THE BUYER IN DECIDING WHETHER TO PURCHASE THIS PROGRAM.

THIS PROGRAM IS SOLD WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES WHATSOEVER. BECAUSE OF THE DIVERSITY OF CONDITIONS AND HARDWARE UNDER WHICH THIS PROGRAM MAY BE USED, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. THE USER IS ADVISED TO TEST THE PROGRAM THOROUGHLY BEFORE RELYING ON IT. THE USER MUST ASSUME THE ENTIRE RISK OF USING THE PROGRAM. ANY LIABILITY OF SELLER OR MANUFACTURER WILL BE LIMITED EXCLUSIVELY TO PRODUCT REPLACEMENT OR REFUND OF THE PURCHASE PRICE.

Introduction

In the center of this manual you will find two disks containing utility programs for Advan BASIC. Make sure both have their write protect tabs in place. One of these is your backup copy, so put it in a safe place. Here is a list of the main utility programs with a brief description of each.

(1) FORMATEX.COD enables people who don't own Advan BASIC to use your compiled programs. It formats a disk and places the Advan BASIC execution module on the disk. If you save the compiled code for one or more programs on this formatted disk, you can execute the programs without loading Advan BASIC.

(2) CROSSREF.COD lists in alphabetical order the variables used in a program and for each variable the line numbers in which the variable was used. Also, it lists in numerical order all line numbers referred to in the program and the lines where the reference occurred. This information is useful in debugging and in modifying programs.

(3) RENUMBER.COD renumbers the lines of a specified program.

(4) MATIO.APP, MATSET.APP, MATCALC.APP, and MAT.APP when appended to a program provide special commands (i.e., named subroutines) to:

- print matrix data
- input or read data to a matrix
- add, subtract, and multiply matrices
- set the elements of one matrix equal to those of another matrix
- calculate the determinant, inverse, and transpose of a matrix
- set matrix elements to a given value

Note: A matrix is a two dimensional array.

(5) DOSADD.APP when appended to a program allows you to use the disk operating commands KILL@, LOCK@, UNLOCK@, DIR@, and RENAME@.

(6) BASECON.APP when appended to a program provides a command which converts a number from one number base to another, for example from hexadecimal to decimal and vice versa.

If you wish, you can use COPYFILE.COD to transfer any of the files on these utility program disks (except FORMATEX.COD) to another disk and use that disk when you want to execute or append them. This will save some wear and tear on your utility program disks.

To execute FORMATEX.COD, CROSSREF.COD, and RENUMBER.COD, turn on your disk drive(s), insert the disk into drive one, and turn on your computer. After about 20 seconds, you will see a menu listing the programs on the disk. Type the letter next to the program you want, and it will be loaded and executed. At the end of the program, you will be asked to reinsert the utility disk into drive one and then press RETURN. The utility menu will be displayed and you can execute another program if you want to.

If you are working in Advan BASIC, you can execute CROSSREF.COD or RENUMBER.COD without going through the above procedure. Just insert the

utility disk into one of your drives and type EXEC followed by the program name. For example:

EXEC CROSSREF.COD or EXEC D2:CROSSREF.COD

To use the commands provided in MATIO.APP, MATSET.APP, MATCALC.APP, MAT.APP, DOSADD.APP, and BASECON.APP, you must append the appropriate file(s) to your program. For example, if you want to use the command KILL@ you must first insert a disk with DOSADD.APP. Then type APPEND DOSADD.APP if the disk is in drive 1 or APPEND D2:DOSADD.APP if it is in drive 2. Because these are all special subroutines like PUSING.APP, they will not be listed when you use the LIST command.

FORMATEX.COD

To execute a compiled program you must have the Advan execute module in the computer. When you load Advan BASIC this module is automatically loaded. Suppose, however, that you want to sell or give one or more of your programs to someone who doesn't own Advan BASIC. In this case you can use FORMATEX.COD, which will format a disk and also place a special Advan execute module on the disk. Then you can use COPYFILE.COD (see Advan BASIC manual, Ch. 17) to transfer the compiled code for your programs to this special execute disk. To run one of the programs on this disk, the user turns on his disk drive(s), inserts the disk into drive 1, and then turns on his computer. The Advan execute module is automatically loaded and a list of the programs is displayed, with a letter next to each one. The user types the letter of the program he selects. If and when the program ends, the system will direct him to reinsert the execute disk into drive 1 and press RETURN. After a few seconds the menu will reappear and another program can be selected.

If there are programs that you execute fairly often, you might want to use FORMATEX.COD to format a disk and place your programs on it. Then you can execute them without loading Advan BASIC. This will save time since the execute disk loads faster than the full BASIC, and it will also save wear and tear on your Advan BASIC Master disk.

FORMATEX.COD has three special features:
(1) The menu displays and allows you to execute only those files ending in .COD. Only the first part of the name is displayed, without the .COD ending. This is a helpful feature if you need to put data files on a disk. Having the file name end in something other than .COD prevents both the file name from being displayed and the user from trying to execute it. However, a program on the disk can open the file and read or write data to it.

(2) One or more programs can be automatically executed before the menu appears. In fact, there are many situations where you don't want a menu to appear at all. For example, suppose you are selling a disk with only one program on it. There is no point in listing one program and then having the user select it. To automatically execute a program, use COPYFILE.COD to transfer the program to the specially formatted disk and name it AUTORUN. This must be the first file transferred to the disk. If you want a second program to be automatically executed right after the first one, again use COPYFILE.COD to transfer this program to the disk and name it AUTORUN.002. This must be the second program placed on the disk. You can put up to 16 of these AUTORUN programs on a disk. Note that since they do not end in .COD,

they will not be shown on the menu.

(3) You can customize the menu. There are always three lines of text before the list of programs and four lines after it. You can have the system use its standard text for these seven lines or you can specify the text.

To execute FORMATEX.COD, type the letter next to FORMATEX. First, the program will load all of the data it needs. Next, it tells you to insert the disk to be formatted into drive 1 and press RETURN. Then it tells you to type Y to format the disk. This gives you a chance to change your mind before the disk is formatted and all the data on it is lost. Typing N will stop the program from formatting the disk. If you type Y, the program asks whether you want the standard text to appear in the menu. If not, it tells you to enter the text for the menu's top three and bottom four lines.

Note that the screen is blanked during a portion of the format. When the disk has been formatted, the program asks if you want to format another disk. If you do, type Y. When you are finished, the program will come to an end and you will be asked to insert the execute disk (the disk with FORMATEX.COD) into drive 1 and press RETURN; then the menu reappears. You can now use COPYFILE.COD (which is also on this utility disk) to transfer any programs and data files to your newly formatted disk. One of these specially formatted disks will appear to Advan BASIC to be a normally formatted disk, except that it will have fewer free sectors. If you are in Advan BASIC, you can insert one of these disks into a drive and work with it just like a regular disk. You can even use COPYDISK.COD to duplicate the disk.

CROSSREF.COD

CROSSREF.COD first asks for the filename of the program to be analyzed; the program must have been saved with the SAVE command (not SAVES). Use the same filename format as the SAVE command. CROSSREF.COD lists in alphabetical order the variables used in the specified program and also the line numbers where each variable was used. The list can be output to the screen or to a printer. If the variable was set equal to something, the line number will have a negative sign before it. Consider the following program:

```
10 INPUT A
20 PRINT A
```

If you use CROSSREF.COD for this program, it will give:

```
A
  20,-10
```

At line 10 the variable is given a value and thus -10 is listed. At line 20, A is used and 20 appears in the list.

CROSSREF.COD also provides a list of the line numbers referred to in the program and the lines at which they were used. Consider the following program:

```
10 S=0
20 INPUT A
30 IF A=0 THEN 50
40 S=S+A: GOTO 20
50 PRINT S: END
```

For this program, CROSSREF.COD yields:

```
20    40
50    30

A
   30,40,-20
S
   40,50,-40,-10
```

Line number 20 is used in line 40 and line number 50 is used in line 30.

If a variable is set equal to something and never used, or used but never set equal to something, there is probably a mistake in the program, and the following message will appear right before the variable is listed:

**PROBABLE ERROR IN FOLLOWING VARIABLE

The special Advan BASIC subroutines, such as DLISTINT.APP, have line numbers greater than 32767. Because CROSSREF.COD ignores line numbers in this range, the listing generated will be only for the program you have written. This keeps the listing as simple as possible; however, any references to the special appended program(s) will generate the error message described above. For example, SETINT@ will be listed as a subroutine name used by your program, and the line numbers at which it is used will be listed. Since this subroutine has line numbers above 32767, it will not be found by CROSSREF.COD. In this case, the error message will be given even though there is no error.

RENUMBER.COD

This program is used to renumber the lines of an existing program, which must have been saved with the SAVE (not SAVES) command. RENUMBER.COD first asks for the filename of the program to be renumbered and then for the filename of the renumbered program (i.e., the program created by RENUMBER.COD). Use the same filename format as in the SAVE command. Next RENUMBER.COD asks for the line number to be used for the first line and finally for the interval between lines. The renumbered program will now be created. Note that the display will be blanked to reduce execution time.

For example, suppose you have a program with five lines. If you specify 100 for the first line and an interval of 10, RENUMBER.COD will create a program identical to the original, except the lines will be numbered 100, 110, 120, 130, and 140. If any line numbers were referred to in the program, these will also be changed (i.e., in a GOTO, GOSUB, etc.).

Note that RENUMBER.COD will not change the line numbers of any of the special Advan BASIC subroutines, such as PUSING.APP, which you may have appended to your program. Thus, RENUMBER.COD can be used even if you have appended one or more of these special subroutines.

MATIO.APP

Appending MATIO.APP to a program makes available the command MATIN@, which provides an easy way to input or read data to a matrix, and the command MATPRINT@, which simplifies printing matrix data. The format for MATIN@ is

MATIN@ stringexpression;matrixname

Note that a semicolon rather than a comma precedes matrixname, and that the matrix must be real (not integer). With this command you can either read matrix data from data statements or you can input data to the matrix from the keyboard. If stringexpression equals "R", data will be read; if it equals "I", data will be input from the keyboard. Matrixname specifies a real matrix to which the data is sent. For example, the following program lines will read the numbers 1, 2, 3, 4, 5, 6 into the real matrix T and then print the six numbers:

```
100 DIM T(2,1)
110 MATIN@ "R";T
120 DATA 1,2,3,4,5,6
130 FOR A%=0% TO 2%
140   FOR B%=0% TO 1%
150     PRINT T(A%,B%)
160   NEXT B%
170 NEXT A%
```

If you want to input data from the keyboard to the matrix, you would delete line 120 and change line 110 to MATIN@ "I";T. When line 110 is executed the row and column numbers for each element will be displayed and then the number to be put into that location will be asked for. Note that the first row and the first column are both numbered zero.

The format for MATPRINT@ is

MATPRINT@ stringexpr,integerexpr,integerexpr;matrixname

Again note that a semicolon rather than a comma precedes matrixname. Stringexpression must equal "P" if you want the data output to a printer and "S" if you want it sent to the screen. Matrixname specifies the matrix whose data is to be printed. Each row of the matrix will be printed on one line. The first integerexpression specifies how much space is available for each number. For example, if this number is nine, each element of the array should have less than nine digits; otherwise, the numbers will be printed without spaces between them. Before printing, all of the elements of the matrix will be rounded to the number of decimal places specified by the second integerexpression.

MATSET.APP

Appending MATSET.APP to a program makes available the command MATSET@, whose format is

MATSET@ stringexpression,realexpression;matrixname

Note that a semicolon rather than a comma precedes matrixname and that the matrix must be real. If stringexpression equals "A", all elements of the array will be set to the value of realexpression. If stringexpression

equals "D", the diagonal elements (elements whose row equals column) will be set to the value of realexpression, and all of the other elements will be set to zero.

MATCALC.APP

Appending MATCALC.APP to a program makes available the command MATCALC@, whose format is

MATCALC@ stringexpression;matrixname;matrixname;matrixname

Note that a semicolon rather than a comma precedes matrixname and that the three matrices must be real. If stringexpression equals "+", the third matrix is set equal to the sum of the first two matrices. If stringexpression equals "-", the third matrix is set equal to the first matrix minus the second matrix. For both options all three matrices must have the same number of rows and the same number of columns, or you will get the 'ARGUMENT ERR' message. If stringexpression equals "*", the third matrix is set equal to the product of the first and second matrix. Note that for the last option the number of rows of the first and third matrices must be equal, the number of columns of the second and third must be equal, and the number of columns of the first equal to the number of rows of the second; otherwise the message 'ARGUMENT ERR' will be given.

MAT.APP

Appending MAT.APP to a program makes available the commands MAT@ and DET@. The format for MAT@ is

MAT@ stringexpression;matrixname;matrixname

Note that a semicolon rather than a comma precedes each matrixname and that the matrices must be real. If stringexpression equals "=", the elements of the second matrix are set equal to the elements of the first matrix. The number of rows of the two matrices must be equal as well as the number of columns; otherwise the message 'ARGUMENT ERR' will be given.

If stringexpression equals "T", the second matrix is set equal to the transpose of the first matrix. The number of rows of the first matrix must equal the number of columns of the second and the number of columns of the first must equal the number of rows of the second; otherwise the message 'ARGUMENT ERR' will be given.

If stringexpression equals "INV", the second matrix is set equal to the inverse of the first matrix. The number of rows and columns of the first matrix and the number of rows and columns of the second must all be equal or you will get the 'ARGUMENT ERR' message. Note that if you try to take the inverse of matrix and it has no inverse you will get the 'ARITH.ERR' message.

The format for DET@ is

DET@ stringname;matrixname

Note that a semicolon rather than a comma precedes matrixname and that the matrix must be real. The number of rows must equal the number of columns. The determinant of the matrix is calculated, and the string specified by

stringname is set equal to the value of the determinant. If you want to do calculations using the value of the determinant you must use VAL or VAL% to convert from the string form to the real or integer form.

Special note:
(1) Since the named subroutines in MATIO.APP, MATSET.APP, MATCALC.APP, and MAT.APP use the names MAT, MAT1@, MAT%, MAT1%, MAT2%, MAT3%, MAT4%, MAT5%, and MAT6%, you should not use these names in your programs.

(2) For some errors a line of the named subroutine will be listed rather than the line of your program on which the subroutine was called.

DOSADD.APP

Appending DOSADD.APP to your program enables you to use the commands KILL@, LOCK@, UNLOCK@, DIR@, and RENAME@. The format for KILL@ is

KILL@ filename

The file specified by filename will be deleted. In the example below, the file named ALPHA.DAT will be deleted from the disk in drive 1, and the file named BETA will be deleted from the disk in drive 2.

```
100 KILL@ "ALPHA.DAT"
110 KILL@ "D2:BETA"
```

The formats for LOCK@ and UNLOCK@ are

LOCK@ filename UNLOCK@ filename

The file specified by filename will be either locked or unlocked. If a file is locked, you cannot delete or write to it.

The format for RENAME@ is

RENAME@ filename,filename

This command is used to change the name of a file. The first filename gives the current name and the second filename gives the new name. In the example below, the file named ALPHA.DAT on the disk in drive 1 is renamed ALPHA1.DAT, and the file named BETA.A on the disk in drive 2 is renamed BETA.C:

```
100 RENAME@ "ALPHA.DAT","ALPHA1.DAT"
110 RENAME@ "D2:BETA.A","D2:BETA.C"
```

The format for DIR@ is

DIR@ stringexpression

Stringexpression must be "D1:", "D2:", "D3:", "D4:". This command (e.g., DIR@ "D1:") prints the directory for the specified disk on the video display.

BASECON.APP

Appending BASECON.APP to the program makes available the command BASECON@,

whose format is

BASECON@ integerexpression,integerexpression,stringname

The command is used to convert a number from one number base to another, for example from hexadecimal to decimal. The first integerexpression equals the current base of the number, and the second integerexpression equals the new base. The string, whose name is given by stringname, must equal the number to be converted. After this command is executed, the string will be changed so that it represents the number in the new number base. The following program will input a hexadecimal number, convert it to decimal, and print the result:

```
100 INPUT "ENTER HEX NUMBER"A$
110 BASECON@ 16%,10%,A$
120 PRINT A$
```

If you want to use the number in your program, you need to use the VAL or VAL% command to convert the number from string format to the real or integer form.

CHAIN.APP

Appending CHAIN.APP to a program makes available the command CHAIN@ whose format is

CHAIN@ stringexpression

This command is used to chain from one program to another (i.e., to have one program automatically execute another). Stringexpression gives the name of the program to which control is being transferred. Suppose you want to go from a program named ALPHA to a program named BETA.COD. If BETA.COD is on a disk in drive 1 the following line in ALPHA will achieve this:

1000 CHAIN@ "BETA.COD"

If BETA.COD had been on a disk in drive 2 you would use:

1000 CHAIN@ "D2:BETA.COD"

BETA.COD could also have a CHAIN@ command and transfer control to yet another program. Note that BETA.COD must be a compiled program. In addition, if the program being chained to has a GRAPHICS command, the first program must also have a GRAPHICS command. If the first program does not have such a command, just put GRAPHICS 0% somewhere in the program. It is not necessary to execute this command; for example, you could put it immediately after the CHAIN@ command.

POP.APP

Appending POP.APP to a program enables you to use the command POP@ whose format is

POP@

Suppose you execute a GOSUB 1000 and the subroutine which starts at line 1000 executes a GOSUB 2000. Normally the RETURN at the end of the second subroutine would send you back to the first subroutine, and then a RETURN in this subroutine would transfer you back to the statement immediately after the GOSUB 1000. In some cases you might want to return directly to this statement without going back to the first subroutine. The POP@ command allows you to do this. If the second subroutine executes a POP@ command, the RETURN command will send you back to the statement following GOSUB 1000 instead of returning you to the first subroutine. In effect, POP@ removes the information about the last GOSUB executed.

GETATR.APP

ATARI strings and Advan strings are stored differently on a disk. Appending GETATR.APP to a program makes available the command GETATR@ which allows you to get strings stored in ATARI format from a disk. Its format is

GETATR@ integer1,stringname,integer2

Stringname is the name you are assigning to the string read from the disk. Integer1 is the file number. Integer2 is zero if you are reading from a single density disk and nonzero if you are reading to a double density disk.

PUTATR.APP

Appending PUTATR.APP to a program makes available the command PUTATR@ which allows you to store strings on a disk in ATARI format.

PUTATR@ integer1,stringname

Stringname tells the system which string to save, and integer1 is the file number.

OFFVER.COD

As in ATARI DOS, the standard Advan write to a disk will verify after each write. This increases the reliability of the write; however, it also increases the time required for SAVEs and PUTs. If you execute the program OFFVER.COD it will switch off the verify and speed up your disk write operations. If you are using a RAMDISK, be sure to set it up (i.e., EXEC RAMDISK.COD) before executing OFFVER.COD.

DIRI@

Appending DIR1.APP to a program will allow you to get the directory of a disk. Its format is

DIRI@ disknumber%

Disknumber% is the number of the disk drive (1-4). The array DIR$ has the names and lengths of the files on the disk. DIR$(0%) has the data on the first file, DIR$(1%) the data on the second, etc. A zero length DIR$ indicates that there are no more files on the disk. Bytes 3 through 11 of DIR&( ) have the file name, and bytes 1 and 2 contain an integer equal to the file length. The following program lists on a printer the directory of disk drive 1, and illustrates the use of DIR1@.

```
100 DIR1@ 1%:D%=0%
110 WHILE LEN(DIR$(D%))<>0% DO
120    LPRINT ASCW(DIR$(D%),1%),RIGHT(DIR$(D%),3%)
130   D%+=1%:IF D%=64% THEN 150
140 WEND
150 END
```