A S T R A L 2 0 0 0

DUAL FLOPPY DISK SYSTEM

Reference Manual

1710 Oldridge Ave. North
Stillwater, Minnesota 55082

NOTICE

The information contained in this document is subject to change without notice.

The ASTRAL COMPUTER COMPANY MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ASTRAL shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

TABLE OF CONTENTS

INTRODUCTION

AND

LAB EXERCISES

FOR THE

ASTRAL DISK OPERATING SYSTEM

1710 Oldridge Avenue North
Stillwater, Minnesota 55082

ASTRAL Dual Floppy Disk

The ASTRAL Dual Floppy Disk enables the user to store collections of information called FILES. Files may be BASIC programs or data and ASCII files, such as assembly language programs, object programs, textual information, or other user generated data.

An ASTRAL DOS System Diskette has been supplied with your floppy disk drive. This System Diskette should always be placed label side up in the drive on the left. This leftmost drive is called unit zero (Ø). To insert a diskette; turn the drive on, open the door, hold the diskette by the labelled edge and insert it gently in the slot. You will feel a little resistance as you push the diskette in the last inch or two. When it is inserted all the way, notice the small catch on the right end that holds the installed diskette in place. (To take the diskette out, push it in slightly, disengage from this catch and gently pull it out.) After inserting the diskette close the door.

The first thing you should do is make another copy of the ASTRAL DOS System Diskette. After inserting this diskette in unit Ø (left), insert a blank diskette in unit 1. While under the control of the Monitor (the prompt showing is ">"), type "GFØØØ" followed by a carriage return. This loads the DOS executive into memory. After it is loaded the "!" prompt appears meaning the system is now under the control of DOS (Disk Operating System). Now type "COPY". This will copy everything from the diskette in unit Ø onto the diskette in unit 1. Now take out the diskette in unit 1, label it "ASTRAL DISK BACKUP" and the date. Keep the System Diskette supplied by ASTRAL in a secure place, in its envelope, preferably in a vertical position in a closed container. Use your new copy of the System Diskette. Do not allow dust, dirt, fingers, etc. to contaminate the surface of the disks.

BASIC

Install the System Diskette in unit Ø and another diskette in unit 1. While under control of DOS (!) type "RUN, BASIC". The BASIC interpreter will be brought into memory. After a few seconds the word "READY" followed by the BASIC prompt (:) will appear. Begin working with BASIC. (See the BASIC section of this manual.)

The Disk and DOS

DOS Diskette Layout

Disk storage space is divided into three distinct regions on a System Diskette and two distinct regions on a User Diskette. The System diskette contains the file directory on track Ø, tracks 1-3 are reserved for the DOS System Executive, and the balance of the disk storage area is available as user file area. On a User Diskette, track Ø is reserved for the file directory, and the rest is available to the user. This means that a user Diskette could contain more than 25ØK bytes or characters.

To learn about the File Directory type "DIR" while in DOS. You will notice the headings for the file directory listing are: Name, Attribute, Track, Sector, and Size.

File Names. The file name may be from one to five ASCII characters long. If you want to indicate that the file is on disk unit one, suffix ":1" to the file name. "PETE:1" could be a name for a file located on the diskette placed in unit 1.

Attributes. The ASCII file attributes are Ø and 1. When a file is built or created it has an attribute of zero (Ø). It can be deleted with the DELET command. To protect this file, the attribute can be changed to a one (1) with the ATTR command. The DELET command cannot delete a file with an attribute of one. To delete it, the ATTR command must first be used to change the attribute back to zero.

BASIC programs and files have attributes of ten (1Ø) and twenty (2Ø), respectively. With these attributes they can be purged from within BASIC or deleted from within DOS. The protect command can be used within BASIC to prevent the accidental deletion of these BASIC files and programs. Protected programs and files will have attributes of eleven (11) and twenty-one (21).

The ASTRAL Disk BASIC has the unique capability of allowing you to save programs in a format which prevents others from being able to list them. The CSAVE command accomplishes this. CSAVEd programs will have attributes of ninety (9Ø). If you also protect this program the attribute becomes ninety-one (91).

When a program or file is purged from within BASIC, its name will still appear in an DOS directory listing. Its attribute will appear as an eighty (8Ø). This means that the

program or file has been deleted but the diskette has not been packed. Periodically the diskettes should be packed by using the DOS DELET command. If you just want to pack the diskette and do not want to delete any files with attributes of Ø, 1Ø, 2Ø; just type "DELET" or "DELET:1" depending upon whether the diskette is placed in unit zero or unit one.

Track and Sector indicate where the file is located and Size gives its sector size in hexadecimal.

ASTRAL DOS - Lab Exercise 1

(Use the DOS Command section of the manual)

1. If using a new diskette, place it in drive unit 1 and initialize it using the INIT command within DOS.

2. Create a file called LAB1 on the diskette placed in unit one. Make the size be ten sectors.

3. List the file directory for unit one.

4. Create a file called LAB2 on unit one. Change the attribute of this file to a one.

5. List the file directory again. Notice the attributes of the two files you have created.

6. With one "DELET" command try to delete both of these files.

7. Explain how you could delete the file called "LAB2"

8. Rename "LAB2" to "PAT".

ASTRAL DOS - Lab Exercise 1 - Answers

1. !INIT:1

2. !CREAT,LAB1:1,A (Remember "A" in hexadecimal represents ten)

3. !DIR:1

4. !CREAT,LAB2:1,5 (Your size may differ)
   !ATTR,LAB2:1,Ø1

5. !DIR:1

6. !DELET:1,LAB1,LAB2

7. You could first change the attribute back to zero, then delete it.

8. !RENAM,LAB2:1,PAT:1

Other suggestions:

To open the door on each disk drive simply push up on the door latch which is at the center of each door about one and one-half inches above the metallic horizontal stripe. The door is designed to open rapidly. Just push up on the latch and let it fly open.

When inserting the diskette, you will notice it slides in more easily if you push it in by holding it to the right of center instead of on the left end.

ASTRAL DOS - Lab Exercise 2

Now let's take a look at one way of entering information into a file. The commands used will be BUILD, LIST, and VIEW from DOS and I(INSERT) and E(END) from the EDITOR. (We believe that a person learns best by "doing". We hope you think so too.)

1. First build a file called "TEXT" on unit 1. After executing BUILD,TEXT:1 the prompt character "@" will appear. This means that the system is now under the control of the EDITOR.

2. The double ESC or ALT MODE characters are always used in the EDITOR to end a command since a carriage return is a legal EDITOR character. The "@" prompt now appears again. Type "E" to end your use of the EDITOR. This will bring you back to DOS.

   Enter the following paragraph by typing "I" followed by the paragraph text. Enter carriage returns when you reach the end of each line as you would with a typewriter. If you make mistakes don't correct them, just keep typing. Remember, when you get done press the ESC or ALT MODE key twice and the "E" to get back to DOS.

The ASTRAL Text Editor allows the user to create and modify alphanumeric text with capabilities including inserting, deleting, and changing characters or lines. The resulting edited text is then output to the user designated floppy disk file.

3. While in DOS , list the contents of file "TEXT".

4. If you had a very long file and were using a CRT

with the LIST command, the first part of the listing of the file would roll off the top of the screen. The VIEW command solves this problem. VIEW,TEXT:1,24,2 for example, would display the TEXT file 24 lines to a frame starting with line 2.

View your file TEXT, with 2 lines to a frame starting with line 1. Is your first line listed? If not, list it.

N causes the next frame to be displayed.

P causes the previous frame to be displayed.

CR (carriage return) returns you to the DOS prompt.

(See the description of VIEW for other options.)

5. Type "EXIT" to get back to the monitor.

Before continuing read the DOS and EDITOR sections of this manual.

We, at ASTRAL, hope this first experience with your system has been a good one. If you have any questions or comments concerning this approach to learning about ASTRAL, The Supported Microcomputer System, please write to the ASTRAL Marketing Manager, 1710 Oldridge Avenue North , Stillwater, Minnesota 55082.

ASTRAL DOS - Lab Exercise 2 - Answers

1. BUILD,TEXT:1

2.
@ITHE ASTRAL TEXT EDITOR ALLOWS THE USER TO CREATE AND MODIFY ALPHANUMERIC TEXT WITH CAPABILITIES INCLUDING INSERTING, DELETING, AND CHANGING CHARACTERS OR LINES. THE RESULTING EDITED TEXT IS THEN OUTPUT TO THE USER DESIGNATED FLOPPY DISK FILE.

(You may use upper and lower case letters if your terminal allows it.)

3. !LIST,TEXT:1

4. !VIEW,TEXT:1,2,Ø

5. !EXIT

ASTRAL Text Editor Lab Exercise

The ASTRAL Text Editor can be a very useful tool in preparing a file containing ASCII information. Your file may be a letter you want to send to several people, using each individual's name in his copy. It may be an assembler program or ASCII data of some type.

The edit commands enable you to insert and delete characters and lines and change one character string to another. These actions take place from the buffer pointer in either a forward or backward direction through the file. Learning to control the position of the pointer is paramount.

In order to give practice with the Text Editor, a poem with a few mistakes scattered throughout is included. The poem is called "Take Time" and is on the System Diskette as file "TKTIM". The following lab exercises will use this file. You may need to use the DOS and Editor reference manuals.

Remember - Editor commands are terminated by two ESC or ALT MODE characters. The ESC prints a $ on your terminal.

1. Use the DOS EDIT command to edit TKTIM to TKTM2 on unit one. Append (A) the file immediately after getting into the Editor. (The Editor prompt is the "@".)

2. Have the Editor type out the poem. It is 30 lines long. If you are using a hard copy terminal you may want to circle the errors you find. If using a CRT, just note some of the errors as you read it. A copy of TKTIM is on the next page.

3. To find out where the pointer is located at any time enter T. Move the pointer to the beginning of the line which says "TAKE TIME TO READ".

4. Move the pointer over in front of the first "E" in "REEADING". Then delete the "E". Check to see if reading is spelled correctly now.

One way of checking to see if it was changed properly is to enter a ØL and a T. The ØL moves the pointer to the beginning of the current line.

(TKTIM) TAKE TIME

TAKE TIME TO THINK--THOUGHTS ARE THE SOURCE OF POWER.

TAKE TIME TO PLAY--PLAY IS THE SECRET OF PERPETUAL YOUTH.

TAKE TIME TO READ--REEADING IS THE FOUNTAIN OF WISDOM.

TAKE TIME TO PRAY--PRAYER CAN BE A ROCK OF STRENGTH IN TIME OF TROUBLE.

TAKE TIME TO LOVE-- IS WHAT MAKES LIVING WORTHWHILE.

TAKE TIME TO BE FRIENDLY--FRIENDSHIPS GIVE LIFE A DELICIOUS

TAKE TIME TO LAUGH--LAUTER IS THE MUSIC OF THE SOUL. SOUL.

TAKE TIME TO GIVE--ANY DAY OF THE YEAR IS TOO SHORT FOR SELFISHNESS.

TAKE TIME TO DO YOUR WORK WELL--PRIDE IN YOUR WORK, NO MATTER WHAT IT IS, NOURISHES THE EGO AND THE SPIRIT.

TAKE TIME TO SHOW APPRECIATION--THANKS IS THE FROSTING ON THE CAKE OF LIFE.

THE END

(CORRECTED) TAKE TIME

TAKE TIME TO THINK--THOUGHTS ARE THE SOURCE OF POWER.

TAKE TIME TO PLAY--PLAY IS THE SECRET OF PERPETUAL YOUTH.

TAKE TIME TO READ--READING IS THE FOUNTAIN OF WISDOM.

TAKE TIME TO PRAY--PRAYER CAN BE A ROCK OF STRENGTH IN TIME OF TROUBLE.

TAKE TIME TO LOVE--LOVING IS WHAT MAKES LIVING WORTHWHILE.

TAKE TIME TO BE FRIENDLY--FRIENDSHIPS GIVE LIFE A DELICIOUS FLAVOR.

TAKE TIME TO LAUGH--LAUGHTER IS THE MUSIC OF THE SOUL.

TAKE TIME TO GIVE--ANY DAY OF THE YEAR IS TOO SHORT FOR SELFISHNESS.

TAKE TIME TO DO YOUR WORK WELL--PRIDE IN YOUR WORK, NO MATTER WHAT IT IS, NOURISHES THE EGO AND THE SPIRIT.

TAKE TIME TO SHOW APPRECIATION--THANKS IS THE FROSTING ON THE CAKE OF LIFE.

THE END

ASTRAL Text Editor Lab Exercises - Answers

1. !EDIT,TKTIM:Ø,TKTM2:1
   @A$$

2. @3ØT$$

3. @GL$$
   @T$$ (not necessary)

4. @2ØM$$
   @T$$ (not necessary)
   @D$$

   The above could be written as the command string 2ØMTD$$ or 20MD$$.

   To print the entire current line:
   ØL$$
    T$$

5. @SLOVE--$$
   @ILOVING$$

   or as the string SLOVE--$ILOVING$$

   Searching for "--" would not work because those characters would be found in the previous sentence.

   Three other combinations that would work are:

   @5L19MILOVING$$

   @5LSIS$-3MILOVING$$

   @152MILOVING$$

   Some are more error prone than the first illustrated method.

6. @3LIFLAVOR.$$

   @SIOUS CR $IFLAVOR.$$ (another possibility)

7. @CLAUTER$LAUGHTER$$

8. @BT$$

9. @SSOUL.$SSOUL.$-5MK$$ or

   @SSOUL. CR $K$$

   There are, of course, many ways of doing this. Using the S command seems simpler than counting the number of lines and using L. Before using the K command you may want to use T to verify that the pointer is where you think it is.

10. @BSPOWER$3T$$

    If you search for "POWER" from the position of the pointer after a problem 9, it will not be found.

11. @Z-1LT$$

12. @E$$

    !RENAM,TKTM2:1,TAKTM:1

ASTRAL DISK BASIC

Samples, Suggestions, etc.

Incomplete

This part of the ASTRAL User's Manual is not completed. In fact, it may never be finished because it contains miscellaneous samples, suggestions, hints, and creative ways of getting things accomplished in BASIC. As you create something of this nature, please send it in. This will assure us that this section is always going to be incomplete.

Sample 1. The random number generator

The following program illustrates how RND(X) differs when X=Ø and when X<>Ø.

```
  1 PRINT "MODIFY THIS PROGRAM BY ADDING LINE '8 Y=RND(1)'"
  2 PRINT "IF YOU WANT THE SAME LIST OF RANDOM NUMBERS."
  3 PRINT
  5 X=0:Z=0
 10 FOR A=1 TO 10
 20 Y=RND(0)
 25 PRINT y;
 30 IF Y<.5 THEN 60
 40 X=X+1
 50 GOTO 100
 60 Z=Z+1
100 NEXT A
150 PRINT
200 PRINT Z;"LESS THAN .5     ";X;"GREATER THAN .5"
500 END
```

Explain why PRINT INT((B-A+1)*RND(Ø)+A) will print a random number between A and B.

Sample 2. The "LINE=" statement

This program illustrates how the LINE statement controls the length of the line that is printed.

```
 1 LINE=72
 4 PRINT "MODIFY LINE ONE TO CHANGE THE LENGTH OF THE LINE THAT IS"
 5 PRINT "PRINTED.  YOU MAY WANT TO TRY 10 AND 80."
10 FOR X=1 TO 14
20 PRINT "1234567890";
30 NEXT X
40 END
```

Sample 3. Substrings

(Try running all these sample programs.) Are all the groups of characters, printed out by this program, words found in the dictionary?

```
 10 PRINT "THIS PROGRAM ILLUSTRATES SUBSTRING FUNCTIONS.  LIST IT!"
 20 PRINT
1000 LET A$="EXPONENTIATE"
1010 B$=LEFT$(A$,8)
1020 C$=MID$(A$,4,3)
1030 PRINT A$,B$,C$,RIGHT$(A$,3)
1040 PRINT
1100 D$=MID$(A$,3,2)+MID$(A$,8,1)+MID$(A$,6,3)
1110 PRINT D$
9999 END
```

Sample 4. String arrays

Multiple statements per line appear on line 2000.

```
 10 PRINT "THIS PROGRAM ILLUSTRATES STRING ARRAYS.  LIST IT'"
1000 DIM B$(3,2)
1010 FOR I=1 TO 3
1020 FOR J=1 TO 2
1030 INPUT "ENTER THE FIRST NAME OF ONE OF YOUR FRIENDS",A$
1035 B$(I,J)=A$
1040 NEXT J
1050 NEXT I
2000 FOR K=1 TO 3:FOR L=1 TO 2:PRINT B$(K,L);TAB(25);:NEXTL:PRINT:NEXTK
9999 END
```

Sample 5. Formatting

The function in line 20 is an interesting method to use when you want to print a column of numbers with the decimal points lined up.

```
 10 INPUT X
 20 DEF FNA(X)=LEN(STR$(INT(X)))
 30 PRINT TAB(20-FNA(X));X
 40 GOTO 10
100 END
```

Sample 6. The PEEK function

PEEK returns the decimal number equivalent of the number stored at address X, also given as a decimal number. The ASTRAL Monitor will list contents of memory locations in hexadecimal.

The PATCH statement is also illustrated here.

```
  1 PRINT "ENTER '57344' AND SEE WHAT IS STORED AT HEX ADDRESS E000."
  2 PRINT "(57344=E000 HEX)  WHILE IN THE MONITOR, ENTER TE000,1"
  3 PRINT "TO SEE THE HEX EQUIVALENT OF THE NUMBER STORED AT E000."
  4 PRINT
  5 PRINT "ENTER A 'G' TO GET BACK TO THIS PROGRAM, A CONTROL C TO"
  6 PRINT "RETURN TO BASIC."
  9 INPUT X
 10 Y=PEEK(X)
 15 PRINT Y
 20 PATCH
 40 GOTO 5
100 END
```

Sample 7. The POKE statement

Be careful when using the POKE command. You may destroy part of the BASIC Interpreter if you change a memory location that is less than 13,000.

This program can be used to convert a decimal number between 0 and 255 to hexadecimal.

```
  1 LINE= 100
  3 PRINT "ENTER A NUMBER BETWEEN 0 AND 255.  THE PATCH COMMAND WILL THEN"
  4 PRINT "RETURN CONTROL TO THE MONITOR.  WHILE IN THE MONITOR"
  5 PRINT "ENTER 'T5000,1' TO SEE THE HEX EQUIVALENT OF THE NUMBER THAT"
  6 PRINT "YOU ENTERED.  THEN ENTER A 'G' TO GET BACK TO THIS PROGRAM."
  9 INPUT X
 10 POKE(20480),X)
 20 PATCH
 30 GOTO #
100 END
```

Sample 8. AND and OR

```
100 IF A=B AND C=D THEN 200
130
```

can be accomplished by:

```
100 IF A=B THEN 120
110 GOTO 130
120 IF C=D THEN 200
130
```

```
100 IF A=B OR C=D THEN 200
130
```

can be accomplished by:

```
100 IF A=B THEN 200
110 IF C=D THEN 200
130
```

9. Suppose you have a long program with few print statements and as the program is executing you would like to know how your program execution is progressing.

Stop the program execution by entering CONTROL C. If you want to know the present value of some variable, like X, enter PRINT X. To see what lines are executing, enter TRACE ON. Then restart the program by entering CONT. To stop the trace function, enter CONTROL C, TRACE OFF, and CONT.

10. FILE NOTES 1

In order to use a file in a program, the file must first be opened with a FILES or ASSIGN statement. Opening a file causes the file to be associated with an integer file number from 1 to 8, depending on its position (1-8) in the FILES statement. Once a file is opened, all references to it are through this file number.

```
 10  FILES A,B,*,T
       .
       .
       .
950  ASSIGN F,3
```

In the above example, after line 950 is executed, the file numbers of files A,B,T and F are 1,2,4, and 3, respectively. These files would, of course, have to be created before the program is executed.

If files are not closed during program execution with an ASSIGN statement, they will automatically be closed when the program terminates.

When an end-of-file (EOF) is encountered, the program will terminate with an error message unless an IFEND statement is used to direct the program to another statement.

```
10 FILES NAMES:1
20 IFEND #1 THEN 60
30 READ #1;A$
40 PRINT A$
50 GOTO 30
60 END
```

Note in the above program that the IFEND statement need not be executed each time you check for an EOF. Once the IFEND is executed, the system branches to the specified statement number whenever an EOF is encountered in that file.

ASTRAL DISK BASIC

Reference Manual

1710 Oldridge Ave. North
Stillwater, Minnesota 55082

ASTRAL BASIC - Getting Started

Disk BASIC

1. Make sure the disk drive and the terminal interfaces are plugged into the ASTRAL 2000 and that each unit has its power cord connected.

2. Turn on the disk drive and terminal, then turn on the ASTRAL 2000.

3. Insert the System Diskette into drive unit Ø and your program diskette into unit 1.

4. If you don't have the ASTRAL Monitor prompt (>) showing on your terminal, use the RESET switch on the ASTRAL 2000 front panel.

5. Enter the monitor command, "GFØØØ" to get the ASTRAL Disk Operating System (DOS).
   The DOS prompt (!) will now appear.

6. Enter the DOS Command, "RUN,BASIC" to get BASIC.

7. The BASIC prompt (:) will appear.
   ASTRAL BASIC is now available.

The ASTRAL Disk BASIC commands and statements are documented in this section.

See the BASIC examples given in another section.

Statements

Statement numbers must be between 1 and 9999.
Multiple statements can be entered on one line if they are separated by colons (:).
Keyboard or tape input may be 80 characters per line including the carriage return.

Numbers

Numbers can be represented within the range of 1.0E-99 to 9.99999999E+99.
There are nine digits of significance.
Numbers may be entered and displayed in three formats: integer, decimal and exponential.

Variable Names

A numeric variable may be a letter or a letter followed by a single digit. A string variable is a letter followed by a $.
Both numeric and string arrays can be used. The maximum one dimensional array is 255. The maximum two dimensional array is 255X255, which would require more than 65K of memory.

Calculator Mode

ASTRAL BASIC may be used as a sophisticated calculator by means of its direct statement execution capability.
While BASIC is in command mode some BASIC statements can be entered without line numbers and are executed immediately. The most often used direct statements are the LET and PRINT.

String Concatenation

Strings can be concatenated up to a maximum 128 characters.
Examples:
```
10 A$="GOOD MORNING"
20 B$="PETER"
30 C$=A$+B$
```

Input/Output Statements

INPUT, PRINT, LOAD, and TAPE may reference an input or output port number (1-8). PRINT <3> A,B would send the values A and B to port three, which may have a line printer attached. A serial or parallel interface card is required.

BASIC restart

If for some reason your BASIC program drops you out to the monitor (the prompt given will be ">"), enter G103 to get back into BASIC. Your program should still be intact.

Special Characters

The backspace is the control H.
Control X cancels the current line.
The CONTROL C is used to stop execution of a program. The user may have to send the CONTROL C several times.

ASTRAL BASIC

Commands

1710 Oldridge Ave. North
Stillwater, Minnesota 55082

Notice

Commands flagged with an asterisk "*" can be used as statements.

Statements flagged with an asterisk "*" can be used as commands.

Square brackets "[]" are used to enclose optional portions of expressions.

ASTRAL BASIC COMMANDS

APPEND

APPEND program name

The APPEND command loads from disk the program whose name is given. (Remember to attach :1 to the end of the program name if the program is on the diskette in disk drive unit 1.)

The program is merged with the program already in memory.

TAPPEND

TAPPEND [<n>]

The TAPPEND is similar to APPEND but the program is read from the tape reader. The port number is n.

CAT

CAT and CAT:1

The CAT (catalog) command gives a listing of the programs and files stored on diskettes in either disk drive.

The attributes of BASIC programs are 1Ø and 11, and 9Ø and 91 if they have been CSAVEd. BASIC files have attributes 2Ø and 21.

Programs and files with attributes of 11, 91 and 21, respectively cannot be purged.

CLOAD

CLOAD program name

CLOAD is used to load CSAVEd programs.

CONT

CONT

The CONT command is used to continue the execution of a program after it has been halted by a STOP statement. The next statement executed is the one following the STOP statement that caused the program to halt.

After you stop the execution of a program by using the CONTROL C key, you can continue its execution by entering CONT, also. To help track down a looping bug in your program you may want to set TRACE ON before entering CONT.

CREAT*

CREAT* filename,filelength

CREAT* creates a file. It can be used within a program, however a FILES or ASSIGN statement must still be used to open the file.

The file name can be a source string whose value becomes the file name. File names are five characters or less in length and should consist of letters of the alphabet and/or the digits zero through nine.

The file length is a numeric expression that is evaluated and truncated to an integer. It specifies the number of 128 byte records. A file may vary from one record to 1973 records, which is 252,544 bytes.

CSAVE

CSAVE program name

CSAVE saves a program on diskette in a semi-interpreted state. A CSAVEd program cannot be listed. This is especially useful to assist software development groups that want to protect the proprietary nature of their product.

The attribute of a CSAVEd program is 90. If you want to protect the program from being purged or deleted use the protect (PRO) command. The attribute will then be 91.

CSAVEd programs must be loaded with the CLOAD command.

DELETE

DEL beginning statement number, ending statement number

The DELETE command deletes all statements between and including the specified statements.

DIGITS*

DIGITS=numeric expression

The DIGITS command sets the number of digits printed to the right of the decimal point.

Any digits to the right of those printed will be truncated.

The numeric expression is evaluated and truncated to an integer. Its value should be between 0 and 9 inclusive.

DIGITS=0 resets the printing to floating point mode.

FDOS

FDOS

The FDOS command returns the system to the Floppy Disk Operating System mode.

LINE*

LINE=numeric expression

The LINE command specifies the number of print positions that will be used in a line. To use a full 80 characters set line equal to at least 100. This is necessary because the printing of a space within the last 25% of the line causes the rest of the data to be printed on the next line. This prevents a word or number from being split up by the end of the line.

Default line length is 72.

LIST

LIST [statement m] [,statement n]

The LIST statement causes the statements of the current program to be displayed on the user's terminal. If no argument is given the entire program will be listed. Only one statement will be listed if one argument, m, is given. The statements m through n will be listed if both parameters are given. If n is less than m, the statements from m to the end of the program will be listed.

LOAD

LOAD filename

The LOAD statement loads a program from diskette. The program must have a 10, 11, 90, or 91 attribute. Qualify the filename with a :1 if the diskette is placed in disk drive unit 1.

NEW

NEW

The NEW or SCRATCH command deletes the current program.

PATCH*

PATCH

The PATCH command causes control to be returned to the ASTRAL monitor. If no changes are made in memory where BASIC is stored, you may enter a "G" to go back to BASIC with your program still intact. If PATCH is used within a program, entering a "G" will return control back to the BASIC user program to the statement immediately following the PATCH statement.

PORT

PORT=numeric expression

The PORT command defines the computer I/O port which will serve as the control port.

If you define a port without a terminal as the control port, you will lose control of the system. You will have to use the reset and start over again.

PRO

Pro filename

Protect places the named program or file in the "protected" state. This means that the program or file cannot be purged within BASIC and cannot be deleted within DOS.

The attributes for programs, files, and CSAVEd programs are changed to 11, 21, and 91, respectively.

PURGE*

PURGE program name or file name

The PURGE command is used to delete BASIC programs and files. PURGE may be used as a statement within a program.

PURGE will not pack the diskette. The diskette can only be packed using the DELET command within DOS. An attribute 80 will be shown within DOS until the diskette is packed.

Remember to append ":1" if the file or program is on the diskette in unit 1.

RENUMBER

```
REN    [S#]
       [S#,I]
       [S#,I,BS#]
       [S#,I,BS#,ES#]
```

The RENUMBER command is used to renumber statements of the program currently in RAM.

S# represents what the statement number of the first affected statement will be.

I is the interval you want between statement numbers.

BS# and ES# represent the beginning and ending old statement numbers at which the renumbering is to begin and end.

If no parameters are given, the entire program will be renumbered starting with 10 with intervals of 10.

REN 1000 will renumber the entire program with an interval of 10. The first line number in the renumbered program will be 1000.

REN 1000,20 will renumber as the example above but with an interval of 20.

REN 1000,20,100 will renumber as the last example but will not renumber any statements smaller than 100.

REN 1000,20,100,300 will renumber statements from 100 through 300. Statement 100 will become 1000 with intervals of 20.

RUN

RUN

RUN executes the current program resident in memory. RUN always begins at the lowest statement number. All numeric variables are initialized to zero.

SAVE

SAVE program name

SAVE is used to save your program on diskette. Append ":1" to the program name to save it on the diskette in disk drive unit 1. The LOAD command is used to load the program back into memory.

Program names consist of five characters or less. It is suggested that only alphabetic characters or digits be used to construct program names.

SCRATCH

SCRATCH

The SCRATCH or NEW command deletes the current programs.

STRING

STRING=numeric expression

STRING is used to set the maximum string length for all strings in the program. The default value is 32, the maximum is 128.

TAPE

TAPE[<n>]

The TAPE command was implemented specifically to assist in the loading of paper tapes punched by timeshare systems. If n is omitted the tape unit must be connected to the terminal.

The tape command changes [and] to (and), respectively as the program is loaded. This assists in loading Hewlett-Packard tapes since HP programs will contain the square brackets in DIM statements even though the programmer used parentheses. The ASTRAL BASIC only accepts parentheses.

If n is given, the tape device must be connected to port n.

TRACE ON*

TRACE ON

Prints each line number as the line is executed. TRACE ON and TRACE OFF can be used within a program to help debug a part of the program.

TRACE OFF*

TRACE OFF

Turns the trace function off.

TSAVE

TSAVE [<n>]

TSAVE saves the current program onto tape. If n is given, the program is saved to port n.

UNPRO

UNPRO filename

The unprotect command places the program or file in the unprotected status that it had before the PRO command was executed. Programs and files are saved, csaved, and created in the unprotected status.

ASTRAL BASIC

STATEMENTS

ABS Function

ABS(numeric expression)

The ABS function is a numeric valued function which returns the absolute value of the numeric expression.

ADVANCE # Statement

ADVANCE #filenumber;skipcount

The ADVANCE statement causes the pointer for the specified file to be advanced past the number of items specified by the skip count.

The skip count is a numeric expression which is evaluated and truncated to an integer specifiying the number of items to be skipped. An invalid file number, negative skip count, or trying to advance past the EOF will cause the program to terminate.

ASC Function

ASC(source string)

The ASC function returns the numeric decimal ASCII code value of the first character in the specified source string. For example, ASC("A") would return 65. ASC or a zero length string return Ø.

This is the inverse of the CHR$ function.

ASSIGN Statement

```
ASSIGN filedesignator,filenumber
```

or

```
ASSIGN *,filenumber
```

The ASSIGN statement is used to assign a filedesignator to a file number reserved in the files statement by an "*". The filedesignator may be a string variable.

The specified file is opened. Any other file associated with that file number will be closed.

If the filedesignator is an "*", the file associated with the filenumber will be closed.

Example:

```
  10  FILES PAY,*,INVEN
   .
   .
   .
1000  INPUT "WHAT FILE ARE YOU USING",A$
1010  ASSIGN A$,2
1020  READ #2;X,Y
```

ATAN Function

```
ATAN (numeric expression)
```

ATAN is a numeric valued function which returns the arctangent of the numeric expression. The value returned is the angle in radians.

CHAIN* STATEMENT

CHAIN program name

The CHAIN statement causes the current program to terminate and the program referenced by program name to be loaded and executed.

Do not forget to append ":1" to the program name if it is on the diskette in drive unit 1.

CHAIN used as a command will load a program from disk and execute it.

CHR$ FUNCTION

CHR$(numeric expression)

The CHR$ function returns the single ASCII character that has the value of the numeric expression.

The numeric expression is evaluated and truncated to an integer which should be in the range of from Ø to 255. If the value is not in this range the program terminates with an Error 1.

One reason for using the CHR$ function is to have certain control characters, that cannot be entered from the keyboard, printed out. For example, using CHR$(X) in a print statement with X=8,1Ø, and 13 will make the terminal backspace, line feed, and carriage return, respectively.

Since the ASCII codes are between Ø and 127, the numbers 128 to 255 are evaluated modulo 128.

This is the inverse of the ASC function.

COM Statement

COM variable list

The COM statement lists variables to be passed betweed programs linked by the CHAIN statement.

The variable list is one or more common elements separated by commas. Common elements may be simple numeric or string variables or arrays. Arrays must be dimensioned in the COM statement, if used, and not in DIM statements. Corresponding variables in chained-to programs must also be dimensioned in COM statements to the same values as those in the first program.

COM statements must be the lowest numbered statements in the program.

The transfer of data values between variables in chained programs is done according to the order of the variables in the COM statements. For example, suppose the first program has the statements

```
10 COM S,A$(10),X1
20 COM B$(5)
```

and the second program has

```
5 COM A,C$(10),E,D,$(5)
```

and the first program chains to the second. The current values of S,A$(10),X1, and B$(5) will be used to initialize the variables A,C$(10),E, and D$(5) respectively. Variables S,X1,A$(10), and B$(5) could even be used in the second program for a different purpose.

COS Function

COS(numeric expression)

The COS function is numeric valued and returns the cosine of the numeric expression.

The numeric expression is interpreted as being in radians.

DATA STATEMENT

DATA constant list

The DATA statement specifies data for READ statements. The constant list is one or more constants separated by commas.

DATA statements may be placed anywhere in the program. The data items will be read in sequence as required by the READ statements as they are executed.

The RESTORE command can be used to reset the pointer back to the first data item and the TYP function may be used to determine the type of the next item in the list.

DEF FN STATEMENT

DEF function name(variable)=numeric expression

The DEF statement allows the user to define his own functions.

The function name is always FN followed by another letter of the alphabet. The variable is a non-subscripted numeric variable whose purpose is only to indicate where the actual argument of the function will be used within the expression to the right of the equal sign. If

```
110 DEF FNB(X)=3.14159*X*X
```

is followed in the program by

```
500 A=FNB(6.5)
```

Then A will equal 3.14159*6.5*6.5

Each function name can only be used once in a program.

DIM STATEMENT

DIM dimension list

The DIM statement allocates memory space for an array.

The dimension list consists of one or more array names, each followed by parameters inside parentheses showing the number of rows or number of rows and columns. These one or two dimensional arrays can be either numeric arrays or string arrays. Maximum dimensions are 255 and 255X255 for one and two dimensional arrays, respectively.

All array elements are set to zero by the DIM statement.

If an array is not explicitly defined by a DIM statement, the default dimensions are ten elements for a one dimensional array and ten by ten for a two dimensional array.

An array name may appear in only one DIM statement in a given program.

10 DIM A$(255,255) would require more than 65K of RAM!

10 DIM X(5),C$(3),Y(20) allocates space for 5 numbers in the X array, 20 numbers in the Y array, and 3 strings in the C$ array. Thirty-two characters are reserved for each string unless changed by the STRING command.

END Statement

END

The END statement terminates execution of the program and returns control to the BASIC command mode.

Every program should terminate with an END statement. A space must separate the statement number and the END.

EXP Function

EXP(numeric expression)

The EXP function is a numeric valued function which returns the mathematical constant "e" raised to the power of the numeric expression.

The constant e is approximately 2.718282

FILES STATEMENT

FILES file list

The FILES statement opens files for use in the program. Eight files can be declared within FILES statements.

The file list is made up of file names separated by commas. The files are numbered from 1 to 8 in the order in which they are listed. These file numbers are used by READ, PRINT, ASSIGN, ADVANCE, UPDATE and IF END statements and the REC and TYP functions.

A "*" may be used in the file list to reserve a position for a file to be named later in the program by an ASSIGN statement.

If a file in the file list does not exist the program will terminate with error 53(File not found).

All files in the file list must have been previously created since the files are opened when program execution begins. If you want to create a file within a program, open it using as ASSIGN statement.

FOR AND NEXT STATEMENTS

```
FOR forvariable = initialvalue TO finalvalue [STEP stepsize]
NEXT forvariable
```

FOR and NEXT are used together to set up program loops forcing the group of inbetween statements to be repeated a specified number of times.

The forvariable is a simple numeric variable. The initialvalue, finalvalue, and the stepsize are all numeric expressions.

The forvariable is first set to the initialvalue and the statements following the FOR are executed. When the NEXT statement is encountered, the stepsize is added to the forvariable and execution is resumed with the statement following the FOR. If the addition of the stepsize forces the value of the variable to be greater than the finalvalue, the next instruction executed will be the one following the NEXT statement. If the stepsize is negative, this iterative process will cease when the variable becomes less than the finalvalue.

If no STEP is specified, a value of one is assumed.

If the finalvalue is less than the initialvalue with a positive stepsize, the FOR - NEXT loop will still be executed once.

Numeric expressions used for initialvalue, finalvalue and stepsize are only evaluated once, the first time the loop is entered.

FOR - NEXT loops may be nested one entirely inside the other. In this case, the forvariables cannot be the same.

When the statement after the NEXT statement is executed, the forvariable is equal to the last value used within the loop.

GOSUB AND RETURN STATEMENTS

GOSUB statement number
RETURN

The GOSUB statement transfers program control to the beginning of a subroutine.

A subroutine is a sequence of BASIC statements which perform some task you want executed from more than one location in a program. There is no explicit indication in the program as to which statements constitute the subroutine. The statement number indicated in the GOSUB is the first statement of the subroutine and a RETURN statement is the last.

The RETURN statement returns program control to the statement following the GOSUB statement. There may be more than one RETURN statement in a subroutine however the last statement must be a RETURN.

Subroutines may be nested. GOSUB can be used within a subroutine to call another subroutine, which in turn calls another subroutine, etc.. This subroutine nesting is limited to eight levels.

GOTO* STATEMENT

GOTO statement number

The GOTO statement transfers program control to the specified statement number.

The statement number must be an existing statement in the current program. If the GOTO transfers control to a non-executable statement, such as REM, DIM, COM or DEF, control passes to the next sequential statement.

Program execution can be started at any line number by using the GOTO as a command.

IFEND STATEMENT

IFEND #file number THEN statement number

The IFEND statement causes a branch to the specified statement number when an End-of-File (EOF) mark is detected:

1. during a file read operation
2. when trying to write beyond the EOF
3. when a direct file write exceeds the amount of space in the record.

The branch is not taken if an EOF is encountered during the execution of an ADVANCE statement.

It is not necessary to execute the statement prior to every file read or write because the IFEND only sets a flag which stays set until another IFEND statement is executed for the same file or until the file is closed.

If the IFEND statement is not used and an EOF condition occurs, an error will occur terminating the program.

IF ... THEN STATEMENT

IF relational expression THEN statement number
IF relational expression THEN BASIC statement

The IF ... THEN statement transfers control to the specified statement number or executes the BASIC statement following the THEN if the relational expression is true. If it is not true, then the statement on the line following the IF ... THEN statement is executed.

A relational expression consists of two arithmetic expressions or a string variable and a string expression separated by a relational operator.

The relational operators are:

=	equal
<> or#	not equal
<	less than
>	greater than
< =	less than or equal
> =	greater than or equal

Because numeric values may be rounded during computation, the "=" operator must be used carefully. When possible use the "<=" or ">=" instead.

Strings are compared character by character. A character is "less than" another character if its ASCII code value is the smaller of the two values. Two strings are equal only if they contain identical characters.

INPUT STATEMENT

INPUT [<n>] read variable list

The INPUT statement allows users to enter data from the terminal during program execution.

A read variable list is one or more numeric or string variables separated by commas.

If n is given, data is read from the device attached to port n.

When INPUT X,A$,Y,B$(2,3) is executed, the system prints a "?" and the user is expected to input four data items separated by commas. The first and third would have to be numeric; the second and fourth would be string characters. The first string would be stored in variable A$, the second string in the second row third column of the B$ array. If the expected number of data items is not entered, BASIC will bring out another "?" and wait for more data.

When entering a string, leading blanks are ignored, trailing blanks are not ignored. String items are separated by commas. Quotes are not used as string delimiters.

Entering a CONTROL C when the system is waiting for input will terminate the program.

A literal string can be used in an INPUT statement to print out a message explaining what to enter. For example:

```
INPUT "ENTER YOUR VALUE NOW",X
```

Note that a comma must be used between the literal string and the read variable list.

INT FUNCTION

INT(numeric expression)

The INT function is a numeric valued function which returns the greatest integer less than the numeric expression.

ITM Function

ITM(numeric expression)

The ITM function returns the number of data items between the beginning of the currently accessed file record and the position of the file pointer.

The numeric expression is evaluated and truncated to an integer which is used as the file number designating which file to use.

LEFT$ FUNCTION

LEFT$ (source string,numeric expression)

The LEFT$ function returns the leftmost X characters of the source string, where X represents the value of the numeric expression. The numeric expression is truncated to an integer.

LEN FUNCTION

LEN(source string)

The LEN function returns the number of characters included in the specified string.

The STRING command specifies the maximum string length. The LEN function allows you to check the actual number of characters assigned to a string varuable at any time.

LET* FUNCTION

[LET] variable = expression

The LET statement is used to assign a value to a variable. LET is optional.

The expression can be numeric or string. A string must be enclosed within quotes. e.g. LET A$="MONDAY"

The equal sign is, in this case, an assignment operator. It assigns the value on the right of the equal sign to the variable on the left.

LOG Function

LOG(numeric expression)

The LOG function returns the natural logarithm of the numeric expression.

The numeric expression must evaluate to a number greater than zero. The program will terminate with an error otherwise.

MID$ Function

MID$(sourcestring,numeric expression[,numeric expression])

The MID$ function returns a string of characters from the source string starting with the Xth character from the left and continuing for Y characters, where the first numeric expression evaluates to X and the second to Y.

The second numeric expression is optional. If it is omitted the MID$ function will return a string starting with the Xth character and continue through the end of the source string.

The numeric expressions are truncated to integers.

ON ... GOSUB Statement

ON numeric expression GOSUB statement number list

The ON ... GOSUB transfers control of the statement numbers of the first statements of each subroutine you want to branch to. The statement numbers are separated by commas.

The numeric expression is evaluated and then truncated to an integer n. Control is then transferred to the n th statement number in the list.

If there is no statement number corresponding to the value of the numeric expression, the program will terminate with an error.

ON ... GOTO Statement

ON numeric expression GOTO statement number list

The ON ... GOTO statement transfers control to the statement as defined by the value of the numeric expression.

The statement number list consists of statement numbers separated by commas.

The numeric expression is evaluated and then truncated to an integer n. Control is then transferred to the n th statement number in the list.

ON X GOTO 15Ø,2ØØ,25Ø is the same as:

```
IF X=1 THEN 15Ø
IF X=2 THEN 2ØØ
IF X=3 THEN 25Ø
IF X>3 an error will result
```

PEEK FUNCTION

PEEK(numeric expression)

The PEEK function returns the decimal value contained in the memory location whose decimal address equals the numeric expression.

The numeric expression is evaluated and truncated to an integer.

POS FUNCTION

POS

The POS function returns, in decimal, the current position of the print head or cursor.

The leftmost position is position #1.

POKE FUNCTION

POKE(numeric expression,numeric expression)

The POKE statement takes the decimal value of the second numeric expression and places it in the decimal memory location indicated by the first expression.

This permits the user to modify the contents of memory locations during execution of a BASIC program. Care must be exercised in the use of this command since it is very easy to accidentally modify the BASIC interpreter or the user program.

PRINT* STATEMENT

PRINT [<n>] [print list]

The PRINT statement causes data to be output at the terminal.

If n is given, the print list is output to the device attached to port n.

The print list consists of numeric or string expressions or variables separated by commas or semicolons. The commas and semicolons are called print delimiters. If the print list is omitted, the PRINT causes a skip to the next line.

The comma print delimiter separates the print line into zones sixteen spaces long allowing only five numbers or strings to be printed on an eighty character line.

The semicolon prints numeric values with one space between them and strings with no space between them.

Numeric expressions are evaluated and the results are printed. A literal string is one that is enclosed between quote marks. Literal strings, string variables, and string concatenations are printed as presented.

If the print list is terminated with a semicolon, the line feed/carriage return is suppressed.

PRINT # STATEMENT

PRINT #filenumber [,recordnumber] [;write list] [,END]

The PRINT # statement is used to write data to BASIC files, The write list is any combination of numeric expressions and string expressions separated by commas. The last or only statement in the write list may be "END", which causes an End-of-File (EOF) mark to be written following the last item in the write list.

SERIAL (SEQUENTIAL) ACCESS PRINTING

If a record number is not specified, the data will be written into the file serially without respect to record boundaries. A serial file print leaves the file pointer positioned immediately following the last item printed. Execution of the next PRINT # causes data to be stored one after the other beginning at the current position of the pointer.

Data written serially is usually read serially using successive file reads.

DIRECT (RANDOM) ACCESS PRINTING

If a record number is specified in the PRINT # statement, the data will be stored in that record. Each direct file print causes the items to be written at the beginning of the selected record.

The file pointer is positioned immediately following the last item printed from each PRINT # statement.

The first record of a file is record number 1.

An End-of-File (EOF) condition is generated in a direct file print if the data in the write list exceeds the amount of space available in the record. In both serial and direct access printing an attempt to print data beyond the physical end of file will also create an EOF condition.

Serial and direct file prints can be used to write on the same file. A serial print following a direct print will write data immediately following the previous data.

READ STATEMENT

READ read variable list

The READ statement assigns string and numeric values from DATA statements to the variables in the read list.

The read variable list consists of string and/or numeric variables separated by commas.

The first item in the first DATA statement in the program is the one assigned to the first variable in the first READ statement. Subsequent variables are assigned beginning with the last DATA item not read.

The type of read variable must agree with the data type in the DATA statement. The TYP function can be used to determine the type of the next data item.

READ # STATEMENT

READ #filenumber [,recordnumber] [;read variable list]

The READ # statement reads data from files and can be used to position the file pointer at the beginning of a record.

SERIAL FILE READ

A serial file read is performed if no record number is given.

A serial file read statement reads items from the file specified by the file number into variables specified in the read variable list. The read variable list consists of string or numeric variables separated by commas.

The first item read is the one following the file pointer. Record boundaries are ignored. The items read can begin in the middle of one record and end in another.

DIRECT FILE READ

A direct file read statement is one where the record number is specified. If a record number is specified that is outside the range for the named file an End-of-File (EOF) condition occurs. If the read variable list is omitted, the direct file read causes the file pointer to move to the beginning of the specified record. e.g. READ #3,I positions the file pointer to the beginning of record I.

In both types of READ statements the data type must match the variable type. You may use the TYP function to determine the type of the next item in the file.

An attempt to read beyond an End-of-File will cause an EOF condition to occur. If an IFEND statement is not used to transfer control to another part of the program, the error will cause the program to terminate.

REC FUNCTION

REC(numeric expression)

The REC function returns the current record number being accessed in the file specified.

The numeric expression represents the file number. It is evaluated and truncated to an integer. Specification of an invalid file number will terminate the program with an error.

REM STATEMENT

REM remark

The REM statement allows you to add comments to your program to document it.

The REM statements are not executed.

RESTORE STATEMENT

RESTORE

The RESTORE statement resets the data pointer to the first item in the first DATA statement.

RIGHT$ FUNCTION

RIGHT$(source string, numeric expression)

The RIGHT$ function returns the rightmost X characters of the source string, where X represents the value of the numeric expression truncated to an integer.

RND FUNCTION

RND(numeric expression)

The RND function returns a set of uniformly distributed pseudo-random numbers if the numeric expression evaluates to zero.

If the numeric expression is not zero, a specific number will be returned for each different value.

RND without an argument is the same as RND (Ø)

See the examples in the BASIC Programming Examples section of this manual.

SGN FUNCTION

SGN(numeric expression)

The SGN function returns the number 1 if the sign of the numeric expression after evaluation is positive. If the sign of the expression is negative a -1 is returned; a zero value returns a Ø.

SIN FUNCTION

SIN(numeric expression)

The SIN function is a numeric valued function which returns the sine of the numeric expression. The numeric expression is interpreted as being in radians.

SQR FUNCTION

SQR(numeric expression)

The SQR function is a numeric valued function which returns the square root of the numeric expression. If the numeric expression evaluates to a negative number, the program will terminate with an error.

STOP Statement

STOP

The STOP statement terminates execution of the program and returns control to the command mode. The STOP statement may occur at any point in the program.

The statement number of the STOP statement is displayed upon program termination. The program can be restarted at the statement following the STOP statement that caused the program to terminate by entering CONT. Using GOTO as a command, the program can be restarted at any statement number.

STR$ Function

STR$(numeric expression)

The STR$ function changes the numeric expression into a string of characters that represent the value of the expression.

The STR$ function is the inverse of the VAL function.

TAB FUNCTION

TAB(numeric expression)

The TAB function is used in print operations to move the current print position to a specified column.

The numeric expression is evaluated and truncated to an integer to obtain the destination column.

The first print position on the left is print position one.

The TAB function is ignored if the destination column is to the left of the present print position. If the value of the numeric expression is negative, the program will halt with an error message.

Semicolons should be used as the delimiter after the TAB function. Remember each print variable delimited by a comma requires 16 print positions. For example:

```
PRINT 123,123,TAB(30);456
```

would not place the 4 in column 30.

TAN FUNCTION

TAN(numeric expression)

The TAN function is a numeric valued function that returns the tangent of the numeric expression.

The numeric expression is interpreted as being in radians.

TYP FUNCTION

TYP(numeric expression)

The TYP function is a numeric valued function that returns the data type of the next sequential item in a file or data statement.

The numeric expression represents the file number. It is evaluated and truncated to an integer. The absolute value of the integer must be a valid file number or Ø. If it is not, the program will terminate with an error.

The table below shows the return values for direct and serial files and data statements. Note that the expression must evaluate to a negative number to test item types in a direct or random file.

Return value	Expression=Ø DATA test	Expression>Ø Seq. file test	Expression<Ø Direct file test
1	number	number	number
2	string	string	string
3	end of data	end of file	end of file
4	———	———	end of record

UPDATE Statement

```
     UPDATE #filenumber;numeric expression
or
     UPDATE #filenumber;source string
```

The UPDATE statement replaces the next sequential data item in the file referenced by the file number with the value of the numeric expression or the source string.

The data type of the new item must match the type of the item being replaced. If the source string is longer than the one it is replacing, it will be truncated from the right to fit. If the source string is shorter, it will be padded on the right with blanks.

End-of-Record (EOR) marks are ignored. An EOF condition will be created if an attempt is made to update an End-of-File mark.

The items in the file following the one that is being updated will not be affected. This is unlike using the PRINT # statement to print over existing data since the last new item printed will be the last item in the file.

VAL Function

VAL(source string)

The VAL function returns the numeric constant equivalent to the source string.

The source string must contain numeric characters only, except for the letter "E" used in writing numbers in exponential notation.

The inverse of the VAL function is the STR$ function.

ASTRAL BASIC Error Codes

1	Oversize Variable (over 255) in TAB, CHR, subscript or "ON"
2	Input error
3	Illegal character or variable
4	No ending quote in print literal
5	Dimensioning error
6	Illegal arithmetic
7	Line number not found
8	Divide by zero attempted
9	Excessive subroutine nesting (max is 8)
10	RETURN w/o prior GOSUB
11	Illegal variable
12	Unrecognizable statement
13	Parenthesis error
14	Memory full
15	Subscript error
16	Excessive FOR loops active (max is 8)
17	NEXT "X" w/o FOR loop defining "X"
18	Misnested FOR-NEXT
19	READ statement error
20	Error in ON statement
21	Input Overflow (more than 72 characters on Input line)
22	Syntax error in DEF statement
23	Syntax error in FN error, or FN called on function not defined
24	Error in STRING usage, or mixing of numeric and string variables
25	String Buffer Overflow, or String Extract (in MID$,LEFT$, or RIGHT$) too long
26	I/O operation requested to a port that does not have an I/O card installed
27	VAL function error. String starts with a non-numeric character
28	Transcendental function error
31	Media error on read from disk
32	EOF on write to disk
33	Disk not ready
50	File too large
51	File type mismatch
52	File not found
53	Duplicate filename
54	Illegal drive number
55	Wrong data type in PRINT & UPDATE
56	Too many data files declared
57	EOF error
58	Data type mismatch
59	Bad file number
60	Bad data on file
61	COM executed after variable declared

Examples - file related error messages

Commands:

SAVE filename
    Error #50 if file too large.
    Error #53 if filename already exists.
    Error #54 if illegal drive number.

LOAD filename
    Error #51 if wrong file type.
    Error #52 if file not found.
    Error #54 if illegal drive number.

APPEND filename
    See LOAD for possible errors.

CAT (n)
    Error #54 if illegal drive number.

Statements:

CREATE filename,length
    Error #53 if duplicate filename.
    Error #50 if file too large.

PURGE,filename
    Error #52 file not found.
    Error #54 if illegal drive number.

FILES filename,...
    Error #51 if wrong filetype.
    Error #52 if file not found.
    Error #56 if too many files declared.

ASSIGN filename,number
    Error #59 if bad file number.
    See FILES for other possible errors.

IF END #n THEN line no.
    Error #59 if bad file number.
    Error #7 if line number not found.

READ #n,rec.no.
READ #n;varlist
READ #n,recno.;varlist
    Error #57 if End-of-File reached. If record number specified,
      End-of-Record is End-of-File.
    Error #58 if data type and variable type do not match.
    Error #59 if bad file number.
    Error #60 if bad data on file.

```
PRINT #n;varlist
PRINT #n,rec.no.;varlist
     Error #55 if overprint attempted.
     Error #57 if End-of-File reached.
     Error #59 if bad file number.

UPDATE #n;var
     Error #55 if data type and variable type do not match.
     Error #59 if bad file number.

COM var list
     Error #58 if variable types do not match.
     Error #61 if variables have been declared before executing
       the COM statement.

CHAIN filename
     See LOAD for possible errors.

Disk errors:
     Error #31 Media error; unable to read.
     Error #32 End-of-File or end of disk reached on write.
     Error #33 Disk not ready.
```

Function:

```
TYP(m)
     Error #59 if bad file number.
     Error #60 if bad data on disk.
```

COMMANDS

FOR THE

ASTRAL DISK OPERATING SYSTEM

1710 Oldridge Avenue North
Stillwater, Minnesota 55082

ASTRAL DOS Commands

When an exclamation mark (!) is printed on your terminal, DOS is awaiting one of the following directives or commands.

ASMB

ASMB,sourcefilename,objectfilename,passoption

Used to assemble a source program written for the Motorola 6800 microprocessor. The contents of the source file are assembled and the object output is directed to the object file.

All three operands must be specified. If no object file is to be created, any dummy file name (e.g. X or Y etc.) may be entered since no file directory entry will be created.

The pass option field may contain 2, 3, or 4.

- 2 = both an assembly listing and an object output are produced.
- 3 = only an assembly listing is generated to the list device.
- 4 = only an object output is generated to the output object file.

ASMB,PAYS,PAYO,2 produces an object file named PAYO from the source file named PAYS. The assembly listing will appear on the terminal.

ATTR

ATTR,filename,newattributes

Used to change the present attributes of the designated file. A file is created or built within DOS with a zero (Ø) attribute. If you want to prevent the delete command from deleting a file, its attribute must be changed to a one (Ø1), eleven (11), twenty-one (21), or ninety-one (91).

BUILD BUILD,newfilename

Used to construct a new source file from the terminal keyboard. Uses the ASTRAL Editor.

After getting the "@" prompt, use the I(INSERT) command to enter information from the terminal keyboard. Enter two ESC or ALT MODE characters to terminate your insert command. Enter E(END) to signal that you want to get back to DOS.

BUILD,ATTND allows you to enter data into the new file ATTND.

COPY COPY

Used to copy the contents of the diskette in drive unit Ø onto the diskette in unit 1.

This is a one-for-one image copy; therefore, the contents of the diskette need not be of DOS format.

If any sector of the source diskette is determined bad after five read tries, the last data read from that sector, whether good or bad, is written to the new diskette.

CREAT CREAT,newfilename,newfilesize

Used to create a user designated file directory entry with the specified file name and file size in sectors.

The file size is specified in hexadecimal with a minimum size of one sector and a maximum size of 1973(7B5 hex) sectors. The designated disk space is allocated to this file and it is treated as any other DOS created file entry.

CREAT,FAMLY,1A creates a file directory entry named FAMLY, attribute of Ø, and an allocated disk space of 26 (1A hex) sectors. Each sector is 128 bytes long.

DELET DELET:unitnumber,filename1,filename2,...

Used to delete the designated, non-permanent files from the diskette in the specified drive unit. The contents of that diskette's user file area and file directory area are automatically repacked making this disk space available. If the unit number is omitted, unit Ø is assumed.

DELET:1,TEST,GRD,CAI1,CAI2 deletes the specified files from the diskette loaded into drive unit 1.

DELET without any files specified will just pack the diskette.

DIR DIR:unitnumber

Prints out the contents of the file directory on the specified diskette. Lists the filename, attributes, file's starting track and sector, and the file's size in sectors in hexadecimal.

DIR:1 and DIR list the file directory of the diskette in drive unit 1 and unit Ø, respectively.

DUMP DUMP,filename

Dumps the contents of the specified file to the punch device. Leader and trailer are automatically produced on the paper tape.

DUMP,STDNT:1 transfers contents of the STDNT file to the punch output device.

EDIT EDIT,inputfilename,newoutputfilename

Enables the editing of the contents of the input file in the Text Editor's RAM buffer by using the Editor's A(APPEND) command. The edit operation is terminated; the edited data transferred to the new output file; the file directory updated; and control returned to DOS by using the Editor's E(END) command. To terminate a command push the ESC or ALT MODE key twice.

See the Text Editor section of this manual for other Editor details.

EDIT,GRD2,GRD3 establishes a new file, GRD3, which will receive the data edited from the contents of the existing file GRD2.

Remember to append ":1" to your filename if your file is on the diskette placed in unit one.

EXIT EXIT

Returns control back to the ASTRAL Monitor.

INIT INIT:unitnumber

Initializes the file directory area on the diskette in drive unit one, two, or three. A diskette cannot be initialized if placed in drive zero. New diskettes should be initialized as soon as you receive them.

ALL existing files, permanent or not, are cleared from the specified file directory. This command must be used to prepare any non-DOS diskette for use by DOS. A User Diskette is the resultant of this command.

INIT:1 initializes the file directory area on the diskette placed in unit one.

LIST LIST,filename

Lists the contents of the specified file to the list device. If a CRT is being used or you just want to examine part of a new file, you may want to use the VIEW command.

LOAD LOAD,objectfilename,offsetbias

Loads the contents of the object file into RAM for execution. The file is loaded into memory at locations which are the sum of the memory address specified in the object file plus the offset bias. The offset bias is specified in hexadecimal. If it is omitted, it is equal to Ø.

The memory wraps around from ØØØØ to FFFF. If an object file memory address is specified as 3ØØØ (hex) and you want it to be 2ØØØ (hex) use an offset bias of EØØØ. (3ØØØ+EØØØ=12ØØØ).

Following the loading of the object file, control will return to the ASTRAL Monitor, if no auto-start address exists in the object file, or to the specified auto-start address if it exits.

LOAD,PROG1 loads the contents of the object file, PROG1, into RAM with an offset bias of zero.

MERGE MERGE,newfilename,filename1,filename2,...

Creates a new file whose contents is the concatenation of the contents of the specified files, in the order in which they appear in the command. Existing files are not affected.

By specifying only one existing file name, one file is copied to another.

MERGE,PROG,PRG1,PRG2,PRG3 creates the new file, PROG, with the contents of files PRG1,PRG2,PRG3 in that order.

MERGE,PAYRL:1,PYTST copies the contents of PYTST into a new file PAYRL, on the diskette placed in unit one.

RENAM RENAM,oldfilename,newfilename

The existing file name of the specified file's file directory entry is replaced by the new file name. Only the file name is affected.

RENAM,PROG2,PROG1 renames the file PROG2 to PROG1.

RUN RUN,hexobjectfilename,inputfilename,outputfilename,n

(This command in DOS is not to be confused with the RUN command in BASIC.)

Loads the contents of the object file into RAM for execution. In addition, inputfilename, and outputfilename are opened and the number n is placed in location PASS. After loading, program control is transferred to memory location ASMB.

The last three parameters are optional and are positional in nature. When you omit an optional parameter, you must still include the commas that would follow the parameter. if some specified parameter follows it in the list.

n may be any hex number from Ø to F.

By default, input file parameters are indeterminate, output file parameters are track=76, sector=1, and n=Ø.

To update the directory following outputs to outputfilename, do a JUMP to location UPDAT in the DOS PROM driver. This JUMP loads the DOS Exec into RAM and updates the output file's directory entry.

To "rewind" the input file, the user's program should perform a CALL RESTR, where RESTR is in the DOS Resident Module.

RUN,BASIC loads the BASIC interpreter, which should be on the system diskette, in RAM and executes it. "READY" will appear on the terminal followed by the prompt "#" indicating that you may now use the BASIC language.

RUN,PAY:1,DOLR:1,CHK:1 loads object program PAY into RAM, opens the input file DOLR and the output file CHK, and transfers program

control to memory location ASMB. All files are stored on the diskette in unit 1.

RUN,GRAD:1,,RECD:1 loads object program GRAD into RAM and opens the output file RECRD.

TAPE

TAPE,newfilename

Creates the specified file entry and transfers the contents of the reader input device into that file.

TAPE,PAT:1 loads the file read in from the reader input device into file PAT on the diskette in unit 1.

VIEW

VIEW,filename,linesperframe,firstline

Displays, on the terminal, the contents of the specified file one frame at a time. Linesperframe and firstline are given in hexadecimal. The default values are 14 and 1 repsectively (2Ø and 1 in decimal). The first line is line zero.

When in the VIEW command, the following keys may be used:

B Causes the beginning frame to be displayed (i.e. that frame whose first line is line number Ø).

F Causes the first frame to be displayed (i.e. that frame whose first line is "firstline").

N Causes the next frame to be displayed.

P Causes the previous frame to be displayed.

CR A carriage return returns control to the DOS Executive.

Summary of the ASTRAL DOS Commands

ASMB,sourcefilename,destinationfilename,p

Assembles the contents of the source file and directs the object to the destinationfile. p is the pass number which determines whether the assembly should produce a listing only, object only, or both.

ATTR,filename,newattributes

Changes the present attributes of the designated file to those specified in the newattributes.

BUILD,destinationfilename

Creates a directory entry and allows the user to enter information into the file on the designated diskette from the terminal keyboard.

COPY

Copies the contents of the diskette in drive unit "Ø" onto the diskette in drive unit "1".

CREAT,filename,size

Creates the designated filename in the directory and allocates disk space equal to size.

DELET:u,filename1,filename2.....filename

Deletes the designated files from the diskette in drive unit u, and then repacks the contents of that diskette, making the disk space available for additional files.

DIR:u

Lists the contents of the file directory on the diskette in drive unit u. Lists the filenames, attributes, and file sizes in sectors.

DUMP,filename

Dumps the contents of the file to the punch output storage device.

EDIT,inputfilename,outputfilename

Enables editing of the input file's contents. Edited data is stored into the output file.

EXIT

Returns control to the ASTRAL Monitor.

INIT:u

Initializes the file directory on the diskette in drive unit "u". Clears all existing user files on that diskette. u cannot be zero.

LIST,filename

Lists the contents of the file on the list output device.

LOAD,filename,offsetbias

Loads the contents of the file into RAM for execution.

MERGE,newfilename,filename1,.....filename

Creates a new file which is a concatenation of filenames 1 to n, in that order.

RENAM,oldfilename,newfilename

Renames the old file with the newfilename.

RUN,hexobjectfilename,inputfilename,outputfilename,n

Sets up the specified input, output, and n parameters, if given; loads the contents of the hex object file into RAM for execution; and then does a branch to location 2ØH.

TAPE,destinationfilename

Loads the contents of the reader device into the specified file on diskette.

VIEW,filename,linesperframe,firstline

Displays the contents of the specified file one frame at a time.

"Lines per frame" and/or "first line number" may be omitted, and if so, are assumed to be 14 and 1 respectively. All numbers are in hex. Line zero is the first line.

When in the VIEW command, the following five keys may be used:

N Causes the next frame to be displayed.

P Causes the previous frame to be displayed.

F Causes the first frame to be displayed.

B Causes the beginning frame to be displayed (i.e. that frame whose first line is Ø.)

CR (carriage return) Returns to DOS Executive.

THE ASTRAL TEXT EDITOR

1710 Oldridge Avenue North
Stillwater, Minnesota 55082

The ASTRAL Text Editor

To initiate the ASTRAL Text Editor from DOS use the EDIT command:

!EDIT,inputfilename,outputfilename

The text to be edited resides in the input file and may have been placed there by the DOS BUILD command. Modified text is placed in the output file. Terminate editing with the E(END) command. The prompt character, "@" reminds you that the system is under control of the Text Editor.

Section I - Notes on Operation

Command Execution and Termination

Text Editor commands are single letters typed in response to the "@" prompt. Commands are terminated and executed by entering two ESCAPE or ALT MODE characters, which are printed as a dollar sign ($) on your terminal.

The Text Editor treats a carriage return (CR) as data, which only terminates a line of text you have entered. The CR does not terminate a Text Editor command.

Some commands use the parameter n, which is an integer between -254 and 255. If not in this range, n is evaluated modulo 256. If n is omitted, it is assumed to be positive one.

Backspace

The backspace character for the Text Editor is the control A. The Editor prints each character that is deleted.

Carriage Return (CR)

The Text Editor divides the contents of the workspace into two classifications: characters and lines. A line consists of the characters between two line feed characters. A character is any ASCII character. The Editor treats carriage return and line feed as a single character which can be manipulated as all other characters are. A line feed is automatic whenever a carriage return is used.

The Buffer Pointer

Following the A(APPEND) command, the buffer pointer is positioned at the beginning of the workspace.

The pointer always resides between two characters. It must be moved to the proper position when deleting or inserting characters. T$$ will always type from the pointer to the end of that line.

Section II - The Commands

The ASTRAL Text Editor commands are divided into four functional groups which are input, buffer pointer manipulation, data manipulation, and output.

Examples using the ASTRAL Text Editor are given in the Text Editor Lab section of this manual.

("Buffer pointer" and "pointer" are used interchangeably as are "buffer" and "workspace").

Input Commands (A,I)

A(Append)

A$$

The Append Command causes text to be read from the file specified in the DOS EDIT command, and append to the workspace until one of the following is achieved:

- End of file
- End of file character ($1A) read
- Workspace full
- Fifty lines are read

The Append Command may be repeated until a complete input file is read if it is a small file. Be careful with large files. More than three or four appends may exceed your system's memory.

When more text is required, typing "255P$$" will write the text in buffer to the specified output file, clearing the workspace for additional appends.

I(INSERT)

Itext$$

The insert command allows text to be input into the workspace at the location of the buffer pointer. After insertion, the buffer pointer is positioned to the right of the last character inserted.

The text may be of any length that does not exceed the number of unused bytes remaining in the workspace, and may be made up of any characters except ESC (ALT MODE) or BREAK.

If the inserted text exceeds space available in the buffer, the Text Editor echoes the BELL character. The command should then be terminated and the text stored by using the P command.

Pointer Manipulation Commands (B,L,M,Z)

B(Beginning of Workspace)

B$$

The B command positions the buffer pointer to the beginning of the workspace.

L(Line) nL$$

The L command moves the buffer pointer the number of lines specified by n, either forward or backward depending upon whether + or - is designated. If no n is specified, a +1 is assumed. If n=Ø, the pointer moves to the beginning of the current line.

If n is greater than the number of lines between the pointer and the end (or beginning) of the workspace, the pointer is moved only to the end (or beginning) of the workspace.

A LINE is a sequence of characters ending in a carriage return (CR).

M(Move) mM$$

The M command moves the buffer pointer forward or backward the number of character positions specified by n, depending upon whether a positive or negative argument is used. The pointer will only move backward to the beginning of the buffer or forward as far as the end of the buffer, as it does with the L command.

Z(End of Workspace)

Z$$

The Z command is used to position the pointer to the end of the workspace and is used prior to appending text to the end of the workspace.

Data Manipulation Commands (S,C,D,K,N)

S(Search)

Sstring$$

The S command causes the Text Editor to search for the first occurrence of a specified character string in the workspace. The search for the character string begins at the pointer.

If a match is found, the pointer is positioned immediately after the last character of the matched character string. If the end of the workspace is reached before locating the character string, the editor prints:

"CAN'T FIND" string

The string being searched for is limited to sixteen characters.

C(Change)

Cstring1$string2$$

The C command causes a specified character string of up to sixteen characters to be searched out and replaced by another string. If the search string is greater than sixteen characters, only the first sixteen characters are used. The two strings need not be of the same length.

Only the first occurrence of string1 following the pointer is affected.

D(Delete)

nD$$

The D command causes the number of characters specified by n to be deleted from the workspace. The positive or negative sign determines whether the n characters will be deleted by moving forward or backward, respectively, from the pointer.

K(Kill) nK$$

The K command is used to delete lines in much the same manner as D deletes characters.

If the pointer resides in the middle of a line and the command 2K is given, the remainder of the line and the following line are deleted. If n=Ø, the characters from the pointer back to the beginning of the line are deleted.

N(Search File)
Nstring$$

The N command causes a search to occur in the same manner as the S command, except that the search continues through the entire file by writing the contents of the workspace to the output file and appending in text from the input file until the search argument is satisfied.

If the argument is not found, the "CAN'T FIND" message is printed on the console and the edit function terminates. Control is then returned to DOS as though the E command had been executed.

Output Commands (P,T,E)

P(Put) nP$$

The P command causes the specified number of lines to be written to the output file starting at the beginning of the workspace. Those lines are then deleted from the workspace.

T(Type) nT$$

The command causes the number of lines specified by n to be listed on the terminal. The listing begins where the pointer currently resides.

If n is negative, the T command causes the number of previous lines to be typed. If n=Ø, the characters from the beginning of the line to the pointer are typed.

The pointer position is not changed by the T command. To verify where the pointer is located use T$$.

E(End) E$$

The E command terminates the edit function and causes the entire workspace to be written to the output file. Any remaining text in the input file is copied to the output file and control is returned to DOS.

Summary of ASTRAL Text Editor Commands

INPUT

The Text Editor accepts input from either the system console device or the specified floppy disk input file. This information is stored in a memory buffer, or "workspace". A buffer pointer specifies the location in the workspace where operations will be performed. The two input commands are:

A - APPEND
I - INSERT

POINTER CONTROL

Four buffer pointer manipulation commands move the buffer pointer to various locations in the workspace, permitting operations to occur at a designated point. The four commands are:

B - BEGINNING
L - LINE
M - MOVE NUMBER OF CHARACTER POSITIONS
Z - END OF WORKSPACE

EDIT

Data Manipulation Commands do the actual editing of the workspace data. They permit characters, or lines, to be deleted, changed or searched out. The five data manipulation commands are:

C - CHANGE
D - DELETE CHARACTERS
K - KILL LINES
S - SEARCH
N - SEARCH FILE

OUTPUT

Output commands cause the text in the workspace to be output to the system console device or the specified output disk file, where it is either printed or recorded, depending upon the command. The three output commands are:

P - PUT
T - TYPE
E - END

COMMAND STRINGS

Command strings enable Text Editor commands to be chained together. Any number of commands may constitute strings and are executed as individual commands from left to right. Four commands however, must be followed by the ESC ($) character. They are:

C - CHANGE
S - SEARCH
I - INSERT
N - SEARCH FILE