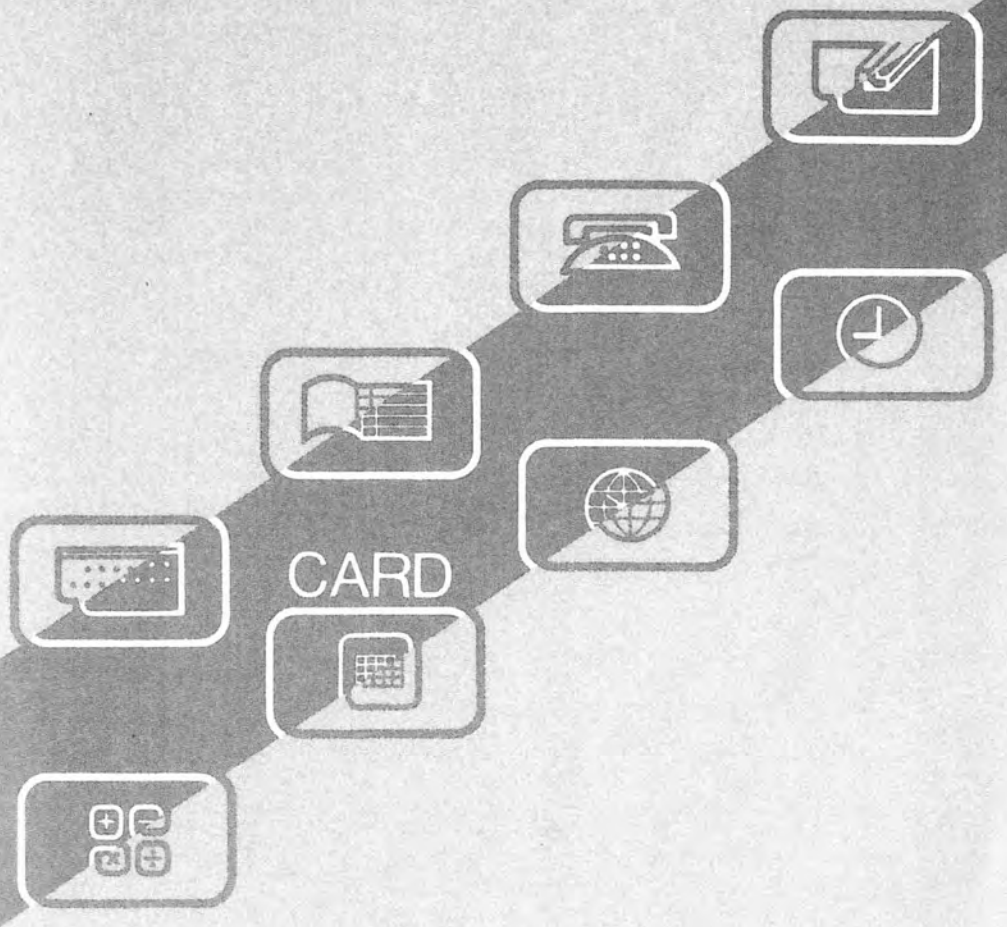
SHARP®

Scientific Computer Card BASIC

MODEL
OZ-707

OPERATION MANUAL



NOTICE

SHARP strongly recommends that separate permanent written records be kept of all important data. Data may be lost or altered in virtually any electronic memory product under certain circumstances. Therefore, SHARP assumes no responsibility for data lost or otherwise rendered unusable whether as a result of improper use, repairs, defect, battery replacement, use after the specified battery life has expired, or any other cause.
SHARP assumes no responsibility, directly or indirectly, for financial losses or claims from third persons resulting from the use of this product and all of its functions, such as the loss of or alteration of stored data, etc.
The information provided in this manual is subject to change without notice.

TABLE of CONTENTS

FOR YOUR RECORDS....
For your assistance in reporting this product in case of loss or theft, please record below the model number and serial number which are located on the back side of the unit. Please retain this information.

Model Number _____ Serial Number _____

Date of Purchase _____ Place of Purchase _____

INTRODUCTION

Thank you for your purchase of SHARP products.

Your new SHARP OZ-707 Scientific Computer Card [BASIC] (hereafter referred to as "the Card") was designed to bring you state-of-the-art technology. It incorporates many advanced features:

- Scientific calculations: You can perform scientific calculations with ease and efficiency.
- Statistical and regression calculations: You can perform single- and two-variable statistic calculations or linear regression calculations. Stored data editing lets you confirm or correct data entry.
- Algebraic Expression Reserve (AER) memory: Formulas you frequently use can be stored in memory and conveniently recalled for repeated use.
- High-precision calculations with 20 significant digits: Double-precision calculation capability makes the Card suitable for application software requiring high computation accuracy.
- Existing software resources are available: Programs developed for the PC-E500 may be used in the Card, if modifications are made for differences in the display width, commands or character codes. Enter the programs through the keyboard.
- Easy-to-use editing features: AUTO, RENUM, and DELETE commands simplify program editing.
- A RAM disk: Part of the internal memory can be used as a RAM disk, which lets you save and load your programs and data just as you would using a diskette.
- A serial I/O interface: Allows direct Card-to-computer communication.

NOTICE

SHARP strongly recommends that separate permanent written records be kept of all important data. Data may be lost or altered in virtually any electronic memory product under certain circumstances. Therefore, SHARP assumes no responsibility for data lost or otherwise rendered unusable whether as a result of improper use, repairs, defect, battery replacement, use after the specified battery life has expired, or any other cause.
SHARP assumes no responsibility, directly or indirectly, for financial losses or claims from third persons resulting from the use of this product and all of its functions, such as the loss of or alteration of stored data, etc.
The information provided in this manual is subject to change without notice.

TABLE of CONTENTS

PART 4 APPENDICES

USING THE CARD FOR THE FIRST TIME

Installing the Battery

Install the battery before using the Card for the first time. If the Card is used without the battery, all stored data will be lost when the Card is removed from the Organizer.

1. Turn off the Organizer power before inserting the Card. Insert the Card and then turn on the Organizer. (Refer to the section Installing the Various IC Cards in the Organizer Operation Manual.)
2. Press the tab on the Card down with a coin or similar object and pull out the battery holder. (See figure 1.)
3. Wipe the supplied battery with a dry cloth, and insert it in the battery holder with the + side up. (See figure 2.)
4. Fully insert the battery holder into the Card by sliding the battery holder back until it clicks in place.

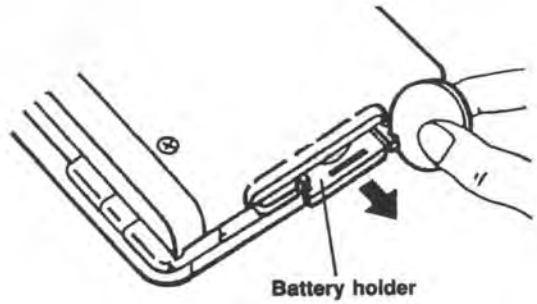


Figure 1

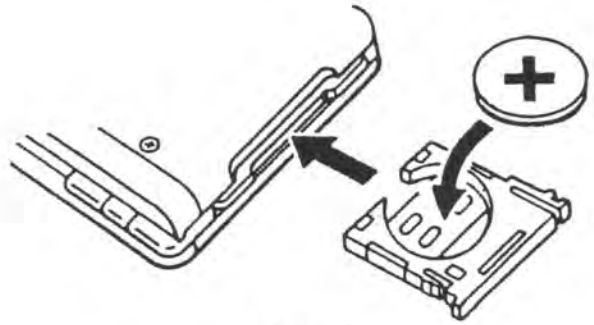


Figure 2

Notes:

- If the Card is removed from the Organizer, and the battery holder is then removed, any data stored in the card will be lost. If data is stored on the Card before a battery has been inserted, you can retain that data by removing the battery holder from the Card while the Card is still in the Organizer.

- As a reminder for the next battery replacement, turn off the Organizer, remove the Card and write the date on the battery replacement label on the back of the Card. (Refer to Battery Replacement in the Appendix C.)

Initializing the Card

When the power is turned on with the Card inserted for the first time, the following is displayed:

```
This IC CARD is
not initialized,
or memory has
been impaired.
Enter "Y" to
INITIALIZED and
CLEAR IC CARD
memory contents.
```

Press the [Y] key to initialize the Card memory. The Card enters the RUN mode.

USING THIS MANUAL

This manual has been designed to introduce you to the capabilities and features of your Card and to serve as a reference tool. The Card is a powerful tool and has many valuable and time-saving functions that even the seasoned computer buff will be pleased to discover!

Part 1: Basic Operation
Part 1 introduces the BASIC programming language as implemented on the Card. Even if you have programmed in BASIC before, please read this part thoroughly, since the BASIC language has many dialects. This part also contains time-saving information in the form of programming shortcuts and debugging techniques.

Part 2: Algebraic and Statistical Operations
Part 2 describes the use of the Card's AER and STAT modes. These modes allow algebraic expressions to be used for repetitive calculations, and for single- or two-variable statistical calculations.

Part 3: Basic Reference
Part 3 is an alphabetic listing of the numeric functions and BASIC commands used in programming the Card. Many of these commands can be used in the direct input modes of the Card.

Part 4: Appendices
Part 4 contains mainly reference material such as code tables, error messages and specifications. You will also find tips on how to take care of your Card.

This manual is not intended to be a self-teaching course in BASIC, the complete description of which is beyond the scope of this manual. If you have never programmed in BASIC, you should buy a separate book or attend a class on the subject before trying to work through this manual.

Key Notation in this Manual

In this manual, keys are described as [ ] for the Card and [ ] for the Electronic Organizer.

Example:
[BASIC], [SIN] for Card keys
[CARD], [▼] for Organizer keys.

sin⁻¹ / sin
- [SIN]: sin key
- [2nd F] [SIN⁻¹]: $\sin^{-1}$ key (or [SHIFT])

The [SHIFT] and [2nd F] keys are functionally identical.

When numerals, characters, or symbols printed on the keys or just above each key are referred to in this manual, only the pertinent letters will appear, with key boxes or the [SHIFT] key omitted, as shown in the following example:

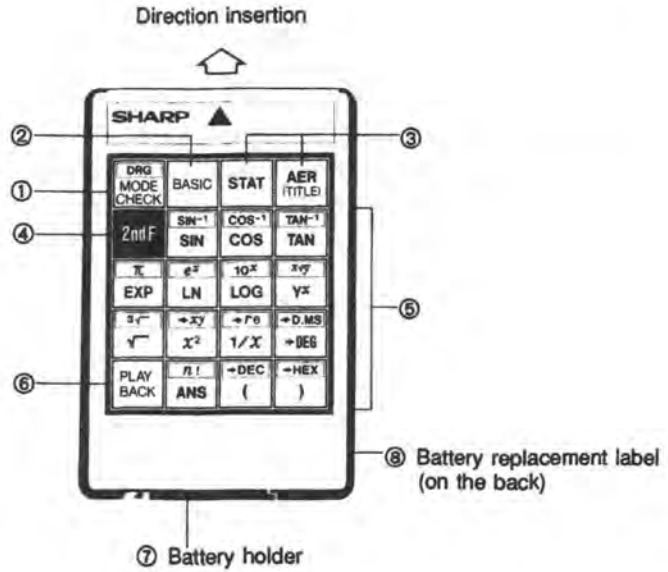
[S] [H] [A] (EDIT) [R] [P] →	SHARP
[SMBL] [SECRET] 3 [4] [5] →	$45

- Where a space must be entered with the [SPC] key, it is indicated by the symbol ␣ in this manual, for example:
  "SHARP␣ EL-865␣␣WN-104"

Keys may appear in their full boxed images in this manual, such as [sin⁻¹ sin] whenever needed.

To discriminate the number zero from the capital letter "O," the zero appears as "Ø" on the display. In the descriptions in this manual, the number zero will also appear as "Ø" whenever it may be confusing. Display symbols will be described when necessary.

PART NAMES



1. MODE CHECK key
Holding this key displays the currently selected modes, the number of free bytes, and the capacity of RAM disk E if reserved.

2. BASIC key
This key toggles the BASIC modes between PRO and RUN.

3. Mode keys
These keys select the STAT (STATistics) and AER (Algebraic Expression Reserve) modes.

4. Second function key
This key is used to specify the functions at the top part of the scientific function keys.

5. Scientific function keys
These keys specify the scientific functions.

6. PLAYBACK key
This key displays the previous menu in the STAT or AER mode, and also redisplays an expression entered in the RUN mode.

7. Battery holder
A battery is installed to backup the Card memory when the Card is removed from the Organizer.

8. Battery replacement label (on the back)
Write the date of battery replacement on this label as a reminder for the next battery replacement.

SELECTING MODES

Before starting to use your Card, you must decide which mode to use. The Card has the following operational modes:

BASIC mode:	The BASIC mode is divided into the RUN and PRO (program) modes.
PRO mode:	Allows you to write or correct a BASIC program
RUN mode:	Allows you to calculate scientific functions or to execute a BASIC program or BASIC commands.
STAT mode:	Allows you to do statistical and regression calculations.
AER mode:	Allows you to enter or use algebraic expressions.

Pressing the [CARD] key after the power is turned on automatically enters the RUN mode.
To select any other Organizer mode, press the corresponding Organizer mode key. To select the Card again, press the [CARD] key.
When using this Card, refer to the Organizer Operation Manual whenever necessary.

When the CARD is selected:
Pressing the [STAT] key, selects the STAT mode.
The STAT menu is displayed.

```
* STATISTICAL
     ANALYSIS *
1.SINGLE-
     VARIABLE
2.TWO-VARIABLE
3.DISPLAY (SNG)
9.QUIT
SELECT No. ?
```

Pressing the [AER] key, selects the AER mode.
The AER menu is displayed.

```
    * AER *
1.EXECUTE
2.CORRECT
   & DELETE
3.RESERVE
9.QUIT

SELECT No. ?
```

Pressing the BASIC key toggles the BASIC mode between PRO and RUN. "RUN MODE" or "PRO MODE" will be displayed followed by the prompt (>).

```
PRO MODE
>
```

To check which modes are currently selected, press the MODE CHECK key (inoperative in the STAT or AER mode). While this key is held, the following are displayed:

```
    MODE CHECK
RUN PRO
DEG RAD GRAD
SNG DBL
NP  PRINT
 50126bytes free
RAMFILE"E:10K"
```

① RUN or PRO mode
② DEG (degree), RAD (radian) or GRAD (gradient) mode
③ SNG (single-precision) or DBL (double-precision) mode
④ NP (non-print) or PRINT mode
⑤ Number of bytes free
⑥ The capacity of RAM disk E, if reserved

The selected modes are highlighted on the display.

PART 1

BASIC OPERATION

Part 1 describes to the use of the BASIC programming language as implemented on the Card. RUN mode and PRO (program) mode are described in this section. Part 1 ends with some suggestions for shortcuts in programming, and for tracing bugs in your program.

1. RUN MODE

The RUN mode is a very versatile operation mode, which allows calculations and operations available in the Organizer CALC mode, and in the AER and STAT modes, calculations using results from the STAT mode, and has the ability to run BASIC programs written in the PRO mode.

Selecting RUN Mode

When the power is turned on, the Card is usually in the RUN mode ("RUN MODE" is displayed).
If "PRO MODE" is displayed, press the [BASIC] key to select the RUN mode. The prompt (>) tells you that the Card is awaiting entry.

```
RUN MODE
>
```

Some Helpful Hints

If you make an error during entry and get an error message, the simplest way to clear the error is to press the [C•CE] key and reenter. If the system "hangs up" (you cannot get it to respond at all), refer to the Organizer Operation Manual to clear the error. Pressing the [C•CE] key also clears the display, but does not erase anything stored in the Card memory.

The prompt (>) tells you that the Card is awaiting entry. As you enter data the prompt disappears and the cursor (_) moves to the right, indicating the next available location in the display.

Press the [ENTER] key to tell the computer that you have finished entering data and to signal the computer to perform the indicated operations. **You must press the [ENTER] key at the end of each line of entry or your calculations will not be acted upon by the Card. The [←┘] key has no effect.**

Do not used dollar signs or commas when entering calculations. These characters have special meanings in the BASIC programming language.

When using the [2nd F] key to implement another key's second function, press the [2nd F] key and then press the other key. The [SHIFT] key may also be used.

For symbols which are not available on the keyboard, press the [SMBL] key followed by the number beside the desired symbol. Other symbols are shown by pressing [▲] or [▼].

- Recalling a formula after execution will cause a value such as "5E3" or "5E-3" to be displayed as "5000" or "0.005". A value such as "5.000" will be displayed as "5". If the value is outside the normal floating point display range, it will be displayed in scientific notation.
- Pressing the [SHIFT] (or [2nd F]) and [◄] (♪) keys will toggle the beep sound for key entry. Setting the beep while in the BASIC mode is effective for other operation modes.
- The Card has an 8 byte input buffer to hold up to 8 key entries while a computation is being performed.
- 8 bytes are used for a numeric value in single-precision mode while 13 bytes are used in double-precision mode. It is not always possible to store up to 254 characters in one line.

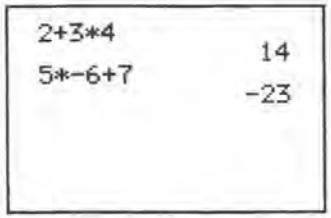
Simple Calculations

The Card performs calculations in the RUN mode with 10-digit precision (unless set to the double-precision mode, which will be discussed later).
Turn the power on and try these simple examples.

Example:
2 + 3 × 4 =
[C•CE] 2 [+] 3 [×] 4 [ENTER]

```
2+3*4
                14
```

Example:
5 × (–6) + 7 =
5 [×] [–] 6 [+] 7 [ENTER]



In the RUN mode, the [ENTER] key must be pressed to obtain the calculated result instead of the [=] key.

The Organizer display consists of 8 lines (16 characters per line). Key entries and calculated results are displayed from the top line of the display. If the characters to be displayed exceed 8 lines, the displayed contents will be moved up by 1 line (the first line will move off the top of the display).

Compound Calculations and Parentheses

You can combine several operations into one step. For example, you may enter:

675 + 6750/45000

Compound calculations, however, must be entered very carefully to avoid ambiguity. When performing compound calculations, the Card has specific rules of expression evaluation and operator priority (see page 28). Use parentheses to clarify your expressions:

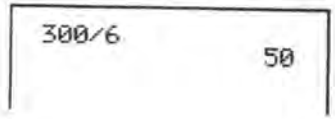
(675 + 6750)/45000 or 675 + (6750/45000)

Recalling Entries

Even after the Card has displayed the results of your calculation, you can verify that the entry was made correctly, and edit it if necessary. To edit, use the left arrow [◄] and right arrow [►] keys.
Use the left arrow [◄] key to position the cursor after the last character. Use the right arrow [►] key to position the cursor over the first character.

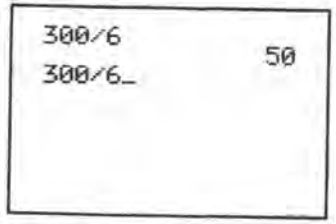
Remember that the left and right arrows are also used to position the cursor. The right and left arrows are very helpful in editing entries without having to retype the entire expression.

300 [÷] 6 [ENTER]



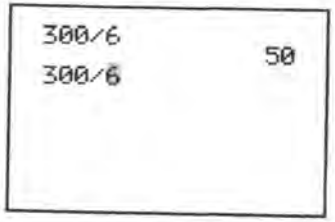
Change this operation to 300/5.
Recall your last entry using the [◄] key.

[◄]



Because you recalled the expression using [◄], the cursor is positioned at the end of the display. Use [◄] to move the cursor one space to the left.

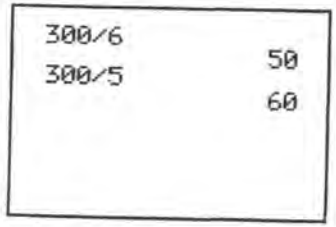
[◄]



Notice that after you move the cursor, it becomes a flashing block. Whenever you position the cursor over an existing character, it will flash.

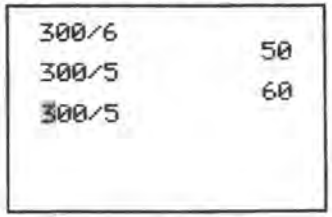
Enter a 5 to replace the 6. An important point in replacing characters — once you enter a new character over an existing character, the original is gone forever! You cannot recall an expression that has been erased.

5 [ENTER]



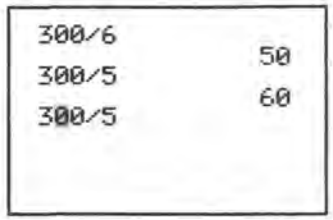
You can also insert or delete characters in an entry. Change the previous calculation to 3000/25. Recall your entry using the [►] key.

[►]

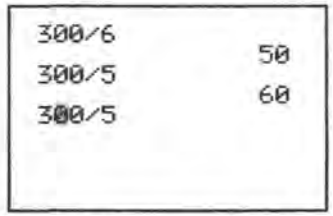


Because you recalled using the [►] key, the flashing cursor is now over the first character. To make the correction you must insert a zero. Using the [►] key, move the cursor until it is over a zero. When making an INSert, position the flashing cursor over the character before which you wish to make the insertion.

[►]

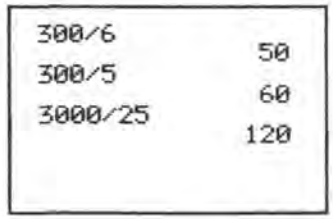


[INS]



Pressing the [INS] key changes the shape of the flashing cursor (◄), which points to the location where your entry will be inserted. Enter the zero. Move the cursor over the 5 and enter 2. Once the entry is corrected, display your new result.

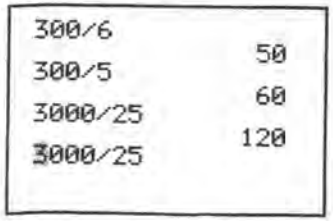
0 [►] [►] [►] 2 [ENTER]



Pressing the [INS] key enters the insert mode and pressing the [INS] key again or the [ENTER] key exits the insert mode.

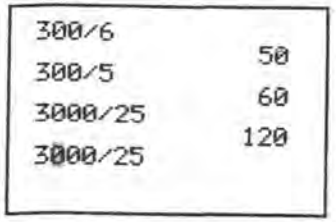
To DELete a character, use the [DEL] key. Change the previous calculation to 3/5. Recall your entry using [►].

[►]



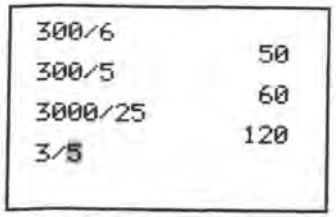
The flashing cursor is now positioned over the first character in the display. To correct this entry, eliminate the zeros. Using [►], move the cursor to the first zero. To delete a character, always position the cursor over the character to be deleted.

[►]



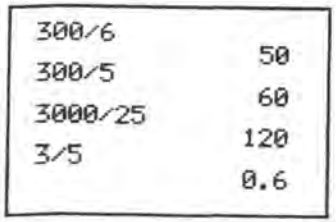
Now use the DELete key to delete the zeros and the two.

[DEL] [DEL] [DEL] [►] [DEL]



Pressing the [DEL] key deletes the character under the cursor and shifts all the following characters one space to the left. Since you have no other changes to make, complete the calculation by displaying the result.

[ENTER]



Note:
Pressing the [SPC] key when the cursor is positioned over a character replaces the character, leaving a blank space. DELete eliminates the character and the space it occupied.

You can also use the [BS] key to delete errors. Note that pressing the [BS] key moves the cursor back one position and deletes the character there, while pressing the [DEL] key deletes the character the cursor is positioned over.

Errors

Recalling your last entry is essential if you get an error message. Suppose you typed this into the Card:

[C•CE] 3 0 0 [÷] [÷] 5 [ENTER]

```
300//5
Syntax error
```

"Syntax error" is the Card's way of saying, "I don't know what you want me to do here". Press the [◄] or [►] key to move the flashing cursor to where the error occurred.

[◄] (or [►])

```
300//5
Syntax error
300//5
```

Use the [DEL] key to correct this error.

[DEL] [ENTER]

```
300//5
Syntax error
300/5
                60
```

If, upon recalling your entry after a syntax error, you find that you have omitted a character, use the INSert sequence to insert it.

When using the Card as a calculator, the majority of errors you encounter will be syntax errors. For a complete listing of error messages, see Appendix A.

Serial Calculations

The Card allows you to use the results of one calculation as part of the following calculation.

Example:
What is 15% of 300 ✳ 150?

[C•CE] 3 0 0 [X] 1 5 0 [ENTER]

```
300*150
                45000
```

In serial calculations it is not necessary to retype your previous results, but DO NOT press the [C•CE] key between entries.

[X] . 1 5 [ENTER]

```
300*150
                45000
45000*.15
                 6750
```

Notice that as you type in the second calculation (✳.15), the computer automatically displays the result of your first calculation at the left of the screen and includes it in the new calculation. In serial calculations the entry must begin with an operator. As always, you end the entry with the [ENTER] key.

Note:
The [%] key cannot be used in percent calculations in RUN mode.
The [%] key should be used as a character only.
For example, 4 5 0 0 0 [X] 15 [%] [ENTER] → Syntax error

To change the sign of the previous result, multiply by –1:

[×] [–] 1 [ENTER]

```
300*150
                45000
45000*.15
                 6750
6750*-1
                -6750
```

Using Variables in Calculations

The Card can store up to 26 simple numeric variables under the alphabetic characters A to Z. If you are unfamiliar with the concept of variables, they are more fully explained in Chapter 2. Variables are designated with an assignment statement:

A = 5 [ENTER]
B = –2 [ENTER]

You can also assign the value of one variable (right) to another variable (left).

C = A + 3 [ENTER]
D = C [ENTER]

As you press [ENTER], the Card performs the calculation and displays the new value of the variable. You can display the current value of any variable by entering the alphabetic character it is stored under:

[C•CE] C [ENTER]

```
C                       8
```

Variables will retain their assigned values even if the Organizer is turned OFF or undergoes an Auto OFF. Variables are lost only when:

- You assign a new value to the same variable.
- You enter CLEAR [ENTER] (not the [C•CE] key).

There are certain limitations on the assignment of variables, and certain programming procedures that cause them to be changed. See Chapter 2 for a discussion of assignment. See Chapter 2 for a discussion of the use of variables in programming.

Single-Precision, Double-Precision

The largest number that the Card can handle in the single-precision mode is ten significant digits, with a two-digit exponent.
In the double-precision mode, it is increased to 20 significant digits (the capital E, indicating the exponent, is replaced by a capital D in double-precision mode).

Selecting Double-Precision Mode

1. Enter RUN mode and press the [C•CE] key to clear the display.
2. Enter DEFDBL and press the [ENTER] key.
   DBL is highlighted on the MODE CHECK display (press [MODE CHECK]), indicating that the Card is now in the double-precision mode.

Although the exponent is indicated by a capital letter D in the double-precision mode, you can still use either the E or D as a character when entering the exponent in an expression. See page 39 for more details.

Canceling Double-Precision Mode

1. In the RUN mode, press the [C•CE] key to clear the display.
2. Enter DEFSNG and press the [ENTER] key.
   SNG is highlighted on the MODE CHECK display, indicating that the Card is now in the single-precision mode.

The double-precision mode is automatically canceled if:

1. The power is turned OFF.
2. The RUN, NEW or CLEAR command is executed.

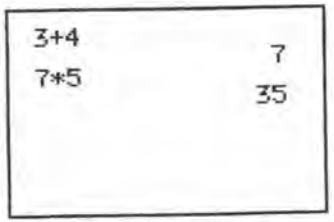
Last Answer Feature

In a simple calculation, the result of the previous calculation can only be used in continuous calculations as the first number.

Example:

C•CE 3 + 4 ENTER

× 5 ENTER

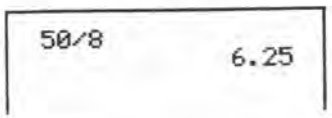


However, the Card has a feature that lets you recall the result of the previous calculation and use it in any location in the current calculation. This is called the last answer feature. It allows the previous answer to be recalled any number of times by pressing the ANS key. If you entered the last example, press C•CE then ANS and you will see "35" displayed.

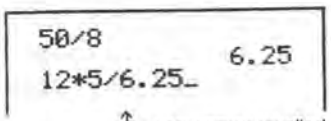
Let's look at an example where a previous result is used twice in the current calculation. Note that in this example, the last answer changes and is updated with the current answer each time ENTER is pressed.

Example:
Use the result (6.25) of the operation 50 ÷ 8 to compute
12 × 5/6.25 + 24 × 3/6.25 =

C•CE 5 0 ÷ 8 ENTER

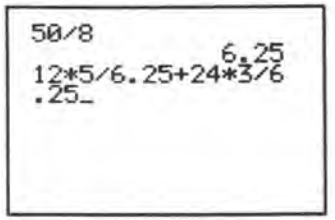


1 2 × 5 ÷ ANS

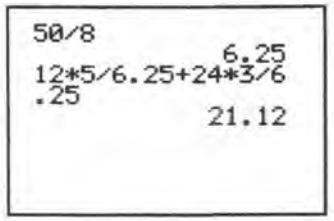


↑— Last answer recalled

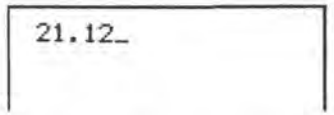
+ 2 4 × 3 ÷ ANS



ENTER



C•CE ANS



Pressing ENTER causes the previous "last answer" to be replaced with the result of the latest calculation. The last answer is not, however, cleared by pressing the C•CE or ON key, but is cleared when the power is turned off.

The last answer cannot be recalled when the Card is not in the RUN mode. The last answer is replaced when a program is executed.

Maximum Calculation Length

The length of the calculation that can be entered is limited to 254 key strokes before the ENTER key is pressed. If you exceed this limit and an error occurs, press the PLAY BACK, ◄, or ► key to recall the entry. The cursor flashes at the last permitted character position. Break the calculation into two or more steps.

Eight bytes are used for each numeric value in single-precision mode, so that it is not always possible to enter 254 keystrokes. (13 bytes per numeric value in double-precision mode.)

Scientific Calculations

The Card has a wide range of numeric functions for use in scientific calculations. PART 3 contains an alphabetical listing of these functions. Note that the notation of the functions in BASIC may differ from conventional mathematical notations.

All scientific functions may be entered in the RUN mode either by pressing the appropriate function key or entering the BASIC command.

The Card enables specification of angular units in degrees, radians or gradient using the DEGREE, RADIAN or GRAD commands.

Angular unit	Command	Description
Degrees	DEGREE	Represents a right angle as 90[°].
Radians	RADIAN	Represents a right angle as π/2[rad].
Gradients	GRAD	Represents a right angle as 100[g].

Angular units may also be specified by pressing [2nd F] [DRG]. Each time these keys are pressed, the angular units change from ...DEG → RAD → GRAD → ...

For practice, use these instructions to specify angular units when required in the following calculation examples.

Press the [C•CE] key before performing a calculation.

Example: sin 30° =
Operation:
DEGREE [ENTER] (Specifies "degree" for angular unit.)

S I N 3 0 [ENTER]
or
[SIN] 30 [ENTER]

```
DEGREE
SIN30
                0.5
```


Example: tan π/4 =
Operation:
RADIAN [ENTER] (Specifies "radian" for angular unit.)

T A N [(] P I [÷] 4 [)] [ENTER]

```
RADIAN
TAN(PI/4)
                1
```

Example: $\cos^{-1}(-0.5)$ =
Operation:
DEGREE [ENTER] (Specifies "degree" for angular unit.)

A C S [−] 0 . 5 [ENTER]

```
DEGREE
ACS-0.5
              120
```

Example: log 5 + ln 5 =
Operation:

L O G 5 [+] L N 5 [ENTER]

```
LOG5+LN5
     2.308407917
```

Example: e^{2+3} =
Operation:

E X P [(] 2 [+] 3 [)] [ENTER]

```
EXP(2+3)
     148.4131591
```

Example: $\sqrt{4^3 + 6^4}$ =
Operation:

SQR [(] 4 [Y^x] 3
[+] 6 [Y^x] 4 [)] [ENTER]

```
SQR(4^3+6^4)
     36.87817783
```

Example:
Convert 30 deg. 30 min. in sexagesimal notation to decimal notation.
Operation:
D E G 3 0 . 3 0 [ENTER]

```
DEG30.30
                30.5
```

Example:
Convert 30.755 deg. in decimal notation to sexagesimal notation.
Operation:
D M S 3 0 . 7 5 5 [ENTER]

```
DMS30.755
              30.4518
```

Example:
Conversion from rectangular to polar coordinates: Determine the polar coordinates (r, θ) for the point P(3, 8) in rectangular coordinates:
Operation:
DEGREE [ENTER] (Specifies "degrees" for angular unit.)

P O L [(] 3, 8 [)] [ENTER]

```
DEGREE
POL(3,8)
      8.544003745
```

Z [ENTER]

```
DEGREE
POL(3,8)
      8.544003745
Z
      69.44395478
```

- The value of θ is stored in variable Z, and the value of r in variable Y.

Example:
Conversion from polar to rectangular coordinates: Determine the rectangular coordinates (x, y) for the point P(12, 4π/5) in polar coordinates.
Operation:
RADIAN [ENTER] (Specifies "radians" for angular unit.)

R E C [(] 1 2 , [(] 4
[÷] 5 [X] P I [)] [)]
[ENTER]

```
RADIAN
REC(12,(4/5*PI))
     -9.708203933
```

Z [ENTER]

```
RADIAN
REC(12,(4/5*PI))
     -9.708203933
Z
      7.053423028
```

- The values of y and x are stored in variables Z and Y, respectively.

Note:
For coordinate conversion, the conversion results are stored in variables Z and Y. Therefore, the previous contents of Z and Y will be cleared.

Example:
Convert the hexadecimal number CF8 to its decimal equivalent.
Operation:
& H C F 8 [ENTER]

```
&HCF8
              3320
```

- "&H" represents a hexadecimal value.
- If you attempt decimal-to-hex conversion on a negative decimal number, the Card internally performs "two's complement" calculation and shows the result in 16's complement.
- The [−] key may be used to reverse the sign of the numeric data now in the display. If the sign of a positive hex number is reversed, the complement of the positive number will be displayed.
- When you wish conversion between decimal and hex numbers, use the following assignment statements:
  From decimal to hex: A$=HEX$ N
  From hex to decimal: N=VAL ("&H"+A$) or N=&H4F3
  The allowable ranges of hex numbers are: $0 \leqq x \leqq$ 2540BE3FF and FDABF41C01 $\leqq x \leqq$ FFFFFFFFFF.

— Reference —

Expressions composed of relational operators (=, >, <, > =, < =, < >) can take on the values listed in the following table:
x and y represent numeric values.

=*	−1 if $x = y$ 0 if $x \neq y$	> =	−1 if $x \geqq y$ 0 if $x < y$
>	−1 if $x > y$ 0 if $x \leqq y$	< =	−1 if $x \leqq y$ 0 if $x > y$
<	−1 if $x < y$ 0 if $x \geqq y$	< >	−1 if $x \neq y$ 0 if $x = y$ ("< >" means "≠")

* If, for example, "A = numeric value" or "B = formula" is used in a logical equation, the computer will not treat it as a logical equation but as an assignment statement for variables. When using an equal (=) sign in a logical equation, use it in the form of "numeric value = A" or "formula = B", with the exception of conditional expressions used in IF statements.

Direct Calculation Feature

In the manual calculations described up to now, the [ENTER] key has always been used to terminate a formula and obtain the calculation result of the formula. However, you can directly operate the functions of the computer with the desired function key (without operating the [ENTER] key) when the objective numeric data is in the display.

Example:
Determine sin 30° and 8!.
Operation:
DEGREE [ENTER]
[C•CE] 3 0 [SIN]

```
SIN 30
                0.5
```

Operation:
[C•CE] 8 [2nd F] [n!]

```
FACT 8
              40320
```

Example:
For $\tan^{-1}\frac{5}{12}$, first check the result of $\frac{5}{12}$, then determine $\tan^{-1}\frac{5}{12}$.
Operation:
DEGREE [ENTER]
5 [÷] 12 [ENTER] [2nd F] [TAN⁻¹]

```
DEGREE
5/12
 4.166666667E-01
ATN 4.166666667E
-01
      22.61986495
```

Note that this "direct" calculation mode is not available for functions requiring the entry of more than one numeric value (binominal functions) such as power, root, or coordinate conversion.
The direct calculation feature is not effective for formulas:
(e.g.) [C•CE] 5 [×] 4→ 5*4_
[LOG] → 5*4LOG_

If no data is on the display, pressing a function key will display the corresponding BASIC command.
The direct calculation feature is effective only for numeric values. Therefore, if hexadecimal numbers A to F are entered for hex to decimal conversion, the direct calculation feature will remain inoperative. In such a case, perform an ordinary manual calculation using the [ENTER] key.

Priority in Direct Input Calculations

You can enter formulas in the exact order in which they are written, including parentheses or functions. The order of priority in calculation and treatment of intermediate results will be taken care of by the Card.

The internal order of priority in manual calculation is as follows:

1. Recalling variables or PI
2. Function (sin, cos, etc.)
3. Power ($\wedge$), root (ROT)
4. Sign (+, −)
5. Multiplication or division ($\times$, /)
6. Addition or subtraction (+, −)
7. Comparison of magnitude (>, > =, <, < =, < >, =)
8. Logical AND, OR, XOR

Note:

- If parentheses are used in a formula, the operation given within the parentheses has the highest priority.
- Composite functions are operated from right to left (sin $\cos^{-1}$ 0.6).
- Chained power (3^{4^2} or $3 \wedge 4 \wedge 2$) is operated from right to left.
- For items 3) and 4) above, the last entry has higher priority.
  (e.g.) $-2 \wedge 4 \rightarrow -(2^4)$
  $3 \wedge -2 \rightarrow 3^{-2}$

Printing for Direct Input Calculations

The calculation steps and results can be printed if the optional printer is connected and switched on, and the [SHIFT] [ENTER] keys are pressed (Print mode).
If a printout is not desired, either switch off the printer, or press [SHIFT] [ENTER] again (Non-Print mode).

Calculation Errors

The following types of errors occur in ordinary calculators, pocket computers, personal computers and also this Card:

Errors due to Least Significant Digit Processing
Usually, the maximum number of digits that can be calculated in a computer is fixed. For example, 4/3 results in 1.33333333333···. In a computer with a maximum of 8 digits, the 8 digits are significant digits; other least significant digits are either truncated or rounded.

Example:
Computer with 10 significant digits

10 significant digits

4 [÷] 3 [ENTER] → 1.3333333333...

Truncated, rounded

Therefore the calculated result differs from the true value by the amount truncated or rounded. (This difference is the error.)

In the Card, a 12-digit calculated result is obtained (or a 24-digit result in double-precision mode). This result is rounded and specially processed to minimize error in the displayed value.

Example in single-precision mode: 4/3 × 3

4 [÷] 3 [×] 3 [ENTER] → 4 — Calculated in succession
4 [÷] 3 [ENTER] → 1.333333333 | Calculated independently
[×] 3 [ENTER] → 3.999999999 |

When calculated in succession, the result of 4/3 is obtained internally in 12 digits and is used for calculation and then rounded.

When calculated independently, the displayed value (10 digits) is used for the calculation.
Example in double-precision mode: 4/3 × 3

4 [÷] 3 [×] 3 [ENTER] → 4# — Calculated in succession

4 [÷] 3 [ENTER] → 1.3333333333333333# | Calculated independently
[×] 3 [ENTER] → 3.9999999999999999# |

When calculated in succession, the result of 4/3 is obtained internally in 24 digits and is used for calculation and then rounded.

When calculated independently, the displayed value (20 digits) is used for the calculation.

Errors due to Function Determining Algorithms
The Card uses a variety of algorithms to calculate the values of functions, such as power and trigonometric functions. When calculations use such functions, an additional source of error is introduced. This error factor increases with the number of functions used in the calculation. The actual error for each function varies according to the values used and is greatest around singularities and inflection points (e.g., when an angle approaches 90 degrees, the tangent approaches infinity).

2. CONCEPTS AND TERMS OF BASIC

In this chapter we will examine some concepts and terms of the BASIC language.

String Constants

In addition to numbers, there are many ways that the Card uses letters and special symbols. These letters, numbers, and special symbols are called characters.

In BASIC, a collection of characters is called a string. In order for the Card to tell the difference between a string and other parts of a program, such as commands or variable names, you must enclose the characters of the string in quotation marks (").

The following are examples of string constants:

"HELLO"
"Goodbye"
"SHARP BASIC CARD"

The following are not valid string constants:

"ORGANIZER	No ending quote
"VALUE OF "A"IS"	Quote cannot be used within a string

Hexadecimal Numbers

The decimal system is only one of many different systems to represent numbers. Another that is quite important when using computers is the hexadecimal system. The hexadecimal system is based on 16 instead of 10. To write hexadecimal numbers you use the familiar 0 to 9 and six more "digits": A, B, C, D, E, and F. These correspond to 10, 11, 12, 13, 14, and 15. When you want the Card to treat a number as hexadecimal, put an ampersand (&) character and "H" in front of the numeral:

```
&HA     = 10
&H10    = 16
&H100   = 256
&HFFFF = 65535
```

Variables

Computers are made up of many tiny memory areas called bytes. Each byte can be thought of as a single character. For instance, the word "card" requires four bytes of memory because there are four characters in it. To see how many bytes are available for use, simply enter FRE0 and press [ENTER] . The number displayed is the number of bytes available for writing programs.

This technique works well for words, but is very inefficient when you try to store numbers. For this reason, numbers are stored in a coded fashion. Thanks to this coding technique, the Card can store large numbers in only 8 bytes (or 13 bytes in double-precision mode). The largest number that can be stored is +9.999999999E + 99 (or 9.99999999999999999D + 99 in double-precision). The smallest number is 1.E-99. This gives you quite a range to choose from. However, if the result of a calculation exceeds this range, the computer will let you know by displaying an error message on the screen (see Appendix A). To see it right now, enter:

9 [EXP] 9 9 [X] 9 [ENTER]

```
9E99*9
Overflow
```


To get the Card working properly again, just press the [C•CE] key. But how do you go about storing all these numbers and strings? It's really very easy. The Card uses names for different pieces of data. Let's store the number 556 in the Card. You may call this number by any name you wish, but for this exercise, let's use the letter R. The statement LET can be used to instruct the Card to assign a value to a variable name, but only in a program statement. However, the LET command is not necessary, so we will not use it often. Now, enter: R = 556 and press [ENTER]. The Card now has the value 556 associated with the letter R. These letters that are used to store information are called variables. To see the content of the variable R, press the [C•CE] key, the [R] key and the [ENTER] key. The Card responds by showing you the value 556 on the right of your display. This is useful when writing programs and formulas.

Next, let's use the R variable in a simple formula. In this formula, the variable R stands for the radius of a circle whose area we want to find. The formula for the area of a circle is: $A = \pi R^2$. Type in the following:

R [Y^x] 2 [X]
[2nd F] [π] [ENTER]

```
R^2*PI
      971179.3866
```

The result is 971179.3866.
This technique of using variables in equations will become more understandable as we get into writing programs.

So far, we've only discussed numeric variables. What about storing alphabetic characters? Well, the idea is the same, but, so that the Card will know the difference between the two kinds of variables, add a $ to the variable name. For instance, let's store the word BYTE in the variable B$. Notice the $ after the B.

This tells the Card that the contents of variable B$ are alphabetic, or string data. To illustrate this, enter the following:

B$ = "B Y T E" [ENTER]

```
B$="BYTE"
>
```

The string BYTE is now stored in the variable B$. To make sure of this, press the [C•CE] key and enter the following:

B $ [ENTER]

```
B$
BYTE
```

The contents of character strings or character variables are displayed from the left edge of the next line.

Variables handled by the Card are divided into the following:

Variables

Numeric variables:
- Fixed numeric variables (A to Z)
- Simple numeric variables (AB, C1, etc.)
- Numeric array variables

String variables:
- Simple string variables (A$, BB$, C2$, etc.)
- String array variables

Numeric variables are further divided into single-precision and double-precision variables. These will be discussed later.

Fixed Numeric Variables

The first type, fixed numeric variables, are always used by the Card for storing numerical data. They can be thought of as pre-allocated variables. In other words, no matter how much memory your program uses, you will always have at least 26 variables to choose from to store numerical data in. Fixed memory locations are eight bytes long.

Simple Variables

Simple variable names are specified by alphanumeric characters, such as AB, B$, C8$. Unlike fixed variables, simple variables have no dedicated storage area in memory. The area for simple variables is automatically set aside (within the program and data area) when a simple variable is first used.

Since separate memory areas are defined for simple numeric variables and simple string variables even if they have the same name, variables such as AB, AB$ and AB#, for example, may be used at the same time.

While alphanumeric characters are usable for simple variable names, the first character of a variable name must always be a letter. **Up to 40 characters** may be used to define a variable name.

Notes:

1. Variable names must not begin with a BASIC command (e.g. PRINTOUT, ONPRINT), but may contain BASIC commands if desired (e.g. APRINT, BONPRINT).
2. Each simple string variable can hold up to 254 characters or symbols.

Array Variables

Sometimes, it is useful to deal with numbers as an organized group, such as a list of scores or a tax table. In BASIC these groups are called arrays. Arrays can be one-dimensional, like a list, two-dimensional, like a table, or multi-dimensional up to 120 dimensions.

Use the DIM (short for dimension) statement to define an array. Arrays must always be declared before they are used (unlike the single-value variables we have been using). The form for the DIMension statement is:

DIM array-variable-name (size)

where:

array-variable-name is a variable that conforms to the normal rules for numeric or array variable names previously discussed.

size is the number of storage locations. Note that when you specify a number for the size, you get one more location than you specified.

Examples of legal numeric and string DIMension statements are:

```
DIM X(5)          → X (0), X (1), X (2), X (3), X (4), X (5)
DIM AA(24)
DIM QUITE5(0)
DIM X$(5)
DIM AA$(24)
DIM QUITE5$(0)
```

The first statement creates an array X with 6 storage locations. The second statement creates an array AA with 25 locations. The third statement creates an array with one location and is actually illogical since (for numbers at least) it is the same as declaring a single-value numeric variable.

It is important to know that an array-variable X and a variable X are separate and distinct to the Card. The former denotes a series of numeric storage locations, and the latter denotes a single and different location.

Now that you know how to create arrays, you might be wondering how we refer to each storage location. Since the entire group has only one name, the way in which we refer to a single location (called an "element") is to follow the group name with a number in parentheses. This number is called a "subscript". For example, to store the number 8 in the fifth element of our array X (declared previously) we would write:

X(4) = 8

If the use of 4 is puzzling, remember that the numbering of elements begins at zero and continues through to the number of elements declared in the DIM statement.

The real power of arrays lies in the ability to use an expression or a variable name as a subscript.

An n-dimensional array is declared by the statement:

DIM array-variable-name (size 1, size 2, ..., size n)

where:

size n specifies the number of elements in the nth dimension of the array. Note that when you specify the number of elements, you get one more element than indicated by the specification.

The following diagram illustrates the storage locations that result from the declaration DIM T(2, 3) and the subscripts (now composed of two numbers) that pertain to each location:

	column 0	column 1	column 2	column 3
row 0	T (0, 0)	T (0, 1)	T (0, 2)	T (0, 3)
row 1	T (1, 0)	T (1, 1)	T (1, 2)	T (1, 3)
row 2	T (2, 0)	T (2, 1)	T (2, 2)	T (2, 3)

Note:
Two-dimensional arrays can rapidly use up storage space. For example, an array with 25 rows and 35 columns uses 875 storage locations!

An n-dimensional array has n indices into the array.
For example, DIM Z$(3,3,3,4) generates a 4-dimensional array with 320 elements.

The following table shows the number of bytes used to define each variable and the number used by each program statement.

Variable type	Number of bytes used	
	Variable name	Data
Single-precision numeric variable	(Length of variable name + 4) bytes	7 bytes
Double-precision numeric variable	(Length of variable name + 4) bytes	12 bytes
String (array) variable	(Length of variable name + 8) bytes	The number of stored string (array) data (bytes)

Element	Line number	Statement & function	ENTER
Number of bytes used	3 bytes	2 bytes	1 byte

Double-Precision Variables
Existing single-precision variables can be converted to double-precision variables by appending a sharp mark (#). For example:

A#, AB#, X#(10), Y#(2,3) and X1#(5,6).

Double-precision variables have 20 significant digits and a 2 digit exponent from –99 to 99.

Note:
Single-character variables to which the sharp mark is appended (e.g. A#) are not fixed numeric variables, but are treated as double-precision simple numeric variables.

The following types of variables are stored in separate memory areas:

A	and	A#
AB	and	AB#
X(10)	and	X#(10)

Variables can be specified as single-precision (10-significant-digit) variables by appending an exclamation mark (!), or as double-precision (20-significant-digit) variables by the sharp mark (#). However, the Card makes it possible to treat any numeric variable as a single- or double-precision variable by the DEFSNG (define single) and DEFDBL (define double) statements. This is especially useful if your program contains numerous double-precision variables.

Storing Values in Double-Precision Variables

1. Using Declarative Signs (! and #)

   AB! (or AB) = 123456789123456789 1234 → $1.234567891 \times 10^{21}$

   The value is stored using 10 significant digits in the single-precision variable AB! (or AB).

   AB# = 123456789123456789 1234 → $1.2345678912345678912 \times 10^{21}$

   The value is stored using 20 significant digits in the double-precision variable AB#.

2. Using Declarative Signs and Declarative Statements (DEFSNG and DEFDBL)

   These are two BASIC commands used as variable definition statements. See Chapter 3 for a description of programming.

Mixing Double- and Single-Precision Values
If a calculation includes double-precision variables, the Card will automatically select double-precision mode where necessary.

Double-precision mode is automatically selected in the following cases:

Calculations are executed on values containing 11 or more significant digits:
Ex. 1234567891234 × 5
The letter D is used in formulas to specify an exponent:
Ex. TAN 7.43D05
Values are identified using the sharp mark (#):
Ex. 4#/7
Double-precision variables are used:
Ex. AB# + BC
The DEFDBL statement is used:
Ex. In RUN mode, enter DEFDBL. DBL is highlighted in the MODE CHECK display.

DEFDBL [ENTER]
5 [÷] 9 [ENTER]

```
DEFDBL
5/9
5.55555555555555
55556D-01
```

If the calculation formula contains a mixture of single- and double-precision values, each individual calculation within the formula is executed according to the degree of precision valid at that time.

Example:

6 * 5 + 4#/7

Single-precision Double-precision

Double-precision calculation

If double-precision values are converted to single-precision variables, the double-precision value is rounded to 10 significant digits.

Example:

[C•CE] A B # [=] 5 # [÷] 9

```
AB#=5#/9_
```

[ENTER]

```
AB#=5#/9
 5.555555555555555
55556D-01
```

A B [=] A B # [ENTER]

```
AB#=5#/9
 5.555555555555555
55556D-01
AB=AB#
  5.5555555556E-01
```

If a double-precision value is used in conjunction with a function of which the argument is single-precision, functional calculations are performed after the double-precision value is rounded off to a 10-digit value.

The following functions are performed in single-precision mode only:
DECI, HEX, POL, REC

Program and Data Files

Programs and data files are fundamental in the use of your Card. A choice of media for storing program and data files is available.

Part of the Card memory can be used as a RAM disk (E:), and is the most readily available storage device. Up to 64 files can be stored in RAM disk E. The RAM disk has 768 bytes of system area and files are stored in blocks of 256 bytes. Other media may be used through the 4-pin option jack or the 15-pin option jack, such as a cassette tape.

Filenames

When saved to a storage medium such as RAM disk E in the Card memory, 4-pin I/O device, cassette tape, or serial I/O device, a file must be given a name. This name is used to load program files into the Card memory, or to access data files on the medium. The filename may be any name up to 8 characters long and include the following characters:

A – Z, a – z, 0 – 9, #, $, %, &, ', (,), {, }, –, ^, _, @, space

Extension

A file extension is an additional way of identifying the type of file (e.g., BASIC program file or text file). The extension consists of three characters added to the end of the filename and separated from it by a period. The extension is specified when the file is saved.

BASIC programs are automatically given the extension .BAS when saved using the SAVE command. When reloaded into memory using the LOAD command, you do not need to specify the .BAS extension.

When the FILES or LFILES commands are used to list the files on the RAM disk, BASIC programs will appear with the .BAS extension unless some other extension has been specified by the user when the file was saved. The .BAS extension must always be specified when using the COPY command.

Device Name

Since files can be stored on tape, Card memory, or the 4-pin I/O device, the device must also be specified. The device name must be followed by a colon (:). The following are the device names used on the Card.

E: RAM disk E (Card memory)
PACOM: 4-pin I/O device (via 4-pin option jack)
CAS: Cassette tape (via 4-pin option jack)
COM: Serial I/O device (via 15-pin option jack)

Note:
Device name PACOM refers to the IC card (Scientific Computer Card [BASIC] or Program Card [BASIC]) installed in another Organizer to or from which a program or data is transferred. The optional CE-200L Data Transfer Cable is required to connect the two Organizers for data transfer.

The complete file descriptor thus consists of the device name, filename, and extension:

d: filename.ext

File Numbers

File numbers are used with certain commands (e.g., OPEN, INPUT# and PRINT#) to read data from or write data to files.
File numbers can be specified with #1 - #255.
A maximum of two files on any two devices may be opened at the same time. The following device combinations are possible:

	E	PACOM	CAS	COM
E	O	O	O	O
PACOM	O	×	×	O
CAS	O	×	×	O
COM	O	O	O	×

O: may be opened at the same time.
×: may not be opened at the same time.

All files are closed before executing the LOAD or MERGE command.

Data Files

There are two types of data file; sequential data files and random access data files. The Card supports sequential data files only. Data is written to a sequential file as a series of ASCII characters stored one item after another (sequentially) in the order sent. The data is read back sequentially when later accessed.

Expressions

An expression is some combination of variables, constants, and operators that can be evaluated to a single value. The calculations that you entered previously were examples of expressions. Expressions are an intrinsic part of BASIC programs. For example, an expression might be a formula that computes an answer to some equation, a test to determine the relationship between two quantities, or a means to format a set of strings.

Numeric Operators

The Card has five numeric operators.

+ Addition
− Subtraction
* Multiplication
/ Division
^ Power

A numeric expression is constructed in the same way that you entered compound calculations. Numeric expressions can contain any meaningful combination of numeric constants, numeric variables, and the numeric operators:

(A * B) ^ 2
A(2,3) + A(3,4) + 5.0 − C
(A/B) * (C + D)

String Expressions

String expressions are similar to numeric expressions except that there is only one string operator — concatenation (+). This is the same symbol used for addition. When used with a pair of strings, the + attaches the second string to the end of the first string and makes one longer string. You should take care in making more complex string concatenations and other string operations because the work space available for string calculations is limited to 254 characters.

Note:
String quantities and numeric quantities cannot be combined in the same expression unless one of the functions that convert a string value into a numeric value or vice versa is used:

"15" + 10 is illegal
"15" + "10" is "1510", not "25"

Relational Expressions

A relational expression compares two expressions and determines whether the stated relationship is true or false. The relational operators are:

>	Greater than
> =	Greater than or Equal to
=	Equal to
< >	Not equal to
< =	Less than or Equal to
<	Less than

The following are valid relational expressions:

A < B
C(1,2) > = 5
D(3) < > 8

If A was equal to 10, B equal to 12, C(1,2) equal to 6, and D(3) equal to 9, all of these relational expressions would be true.

Character strings can also be compared in relational expressions. The two strings are compared character by character according to their ASCII value starting at the first character (see Appendix B). If one string is shorter than the other, a 0 or NULL will be used for any missing positions. All of the following relational expressions are true:

"ABCDEF" = "ABCDEF"
"ABCDEF" < > "ABCDE"
"ABCDEF" > "ABCDE"

Relational expressions evaluate to true or false. The Card represents true by a –1; false is represented by a 0.

Logical Expressions

Logical operations use the Boolean algebra functions AND, OR, XOR and NOT to build connections between relational expressions. The logical operations in a single expression are evaluated after arithmetic and relational operations.

In this way, logical operators can be used to make program decisions based on multiple conditions using the IF ... THEN ... ELSE statement.

Example:

IF A < = 32 AND B > = 90 THEN 150

This statement causes execution to jump to line number 150 if the value of the numeric variable A is less than or equal to 32 and at the same time, the value of numeric variable B is greater than or equal to 90.

IF X < > 13 OR Y = 0 THEN 50

This statement causes execution to jump to line 50 unless variable X has the value 13, or if variable Y is equal to 0.

In a logical operation involving two numbers in the range –32768 to +32767, the two numbers are converted into 16-bit binary integers (in two's complement form) and the logical connection is then evaluated for each corresponding pair of bits in the two numbers.

The results returned by the logical operators for these bit evaluations are listed here:

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

NOT

X	NOT X
1	0
0	1

After each bit pair has returned the corresponding result (a 1 or a 0) according to the above tables, the resulting 16-bit binary number is converted back to a decimal value. This number is the result of the logical operation.

Example:

41 AND 27 →
equals
9

```
41 = 101001
27 = 011011   AND
   ← 001001
```

41 OR 27 →
equals
59

```
41 = 101001
27 = 011011   OR
   ← 111011
```

41 XOR 27 →
equals
50

```
41 = 101001
27 = 011011   XOR
   ← 110010
```

NOT 3 →
equals
–4 (two's complement form)

```
3 =  0000000000000011   NOT
   ← 1111111111111100
```

NOT X can generally be calculated by the equation NOT X = –(X+1).

Parentheses and Operator Precedence

When evaluating complex expressions, the Card follows a predefined set of priorities that determine the sequence in which operators are evaluated. This can be quite significant:

5 + 2 * 3 could be

5 + 2 = 7 or 2 * 3 = 6
7 * 3 = 21 6 + 5 = 11

The exact rules of "operator precedence" are given on page 28.

To avoid having to remember all these rules and to make your program more precise, always use parentheses to determine the sequence of evaluation. The above example is clarified by writing:

(5 + 2) * 3 or 5 + (2 * 3)

3. PROGRAMMING

In the previous chapter, we examined some of the concepts and terms of the BASIC programming language. In this chapter, you will use these elements to create programs. Let us remind you, however, that this is not a manual on how to program in BASIC. What this chapter will do is familiarize you with the use of BASIC on your Card.

Programs

A program consists of a set of instructions to the Card. It will perform the exact operations that you specify. You, the programmer, are responsible for issuing the correct instructions.

BASIC Statements

The Card interprets instructions according to a predetermined format. This format is called a statement. You must always enter BASIC statements in the same pattern. Statements must start with a line number:

```
10: INPUT A
20: PRINT A*A
30: END
```

Line Numbers

Each line of a program must have a unique line number — any integer between 1 and 65279. Line numbers are the reference for the Card. They tell the Card the order in which to run the program, and can be used to tell the Card at which line to start. You need not enter lines in sequential order (although if you are a beginning programmer, it is probably less confusing for you to do so). The computer always begins execution with the lowest line number and moves sequentially through the lines of program in ascending order.

You can use the AUTO command to automatically insert line numbers for you. Each time you press the [ENTER] key, a new line number, with the correct increment, will be automatically inserted. See the BASIC COMMAND DICTIONARY for a full description of this useful function.

It's wise to allow increments of several numbers in your line numbering (10, 20, 30, ... 10, 30, 50, etc.). This enables you to insert additional lines if necessary.

If you use the same line number, the old line with that number is deleted when you enter the new line.

Labelled Programs

Often you will want to store several different programs in the memory at one time. (Remember that each must have unique line numbers). Normally, to start a program with a RUN or GOTO command, you need to remember the beginning line number of each program. However, there is an easier way. You can label each program with alphanumeric characters and run the program.

Label the first line of each program that you want to reference. The label consists of a letter and up to 19 alphanumeric characters in quotes or with * in front of it, followed by a colon:

```
10: *A: PRINT "FIRST"
20: END
80: "B": PRINT "SECOND"
90: END
```

Although both *label and "label" forms may be used, *label is recommended, since it executes more quickly and is more visible in the program listing.

BASIC Commands

All BASIC statements must contain commands. They tell the Card what action to perform. A command is contained with a program, and as such is not acted upon immediately.

Some statements require or allow an operand:

```
10: DATA "HELLO"
20: READ B$
30: PRINT B$
40: END
```

Operands provide information to the Card telling it what data the command will act upon. Some commands require operands, while with other commands they are optional. Certain commands do not allow operands. (See the BASIC COMMAND DICTIONARY for BASIC commands and their uses.)

Note:
Commands, functions and variables entered in lowercase characters will be converted to uppercase characters.

Direct Commands

Direct commands are instructions to the Card that are entered outside of a program. They instruct the Card to perform some immediate action or set modes that affect how your programs are executed.

Direct commands have immediate effect — as soon as you complete entering direct commands (by pressing the [ENTER] key), the command will be executed. Direct commands are not preceded by a line number.

```
RUN
NEW
RADIAN
```

Modes

You will remember that you can use the Card as a calculator in the RUN mode. The RUN mode is also used to execute the programs you create. The PRO mode is used to enter and edit your programs.

Beginning to Program

After all your practice in using the Card as a calculator, you are probably quite at home with the keyboard. From now on, when we show an entry, we will not show every keystroke. Remember to use the [2nd F] or [SHIFT] key to access characters above the keys and to end every line by pressing the [ENTER] key.

Now you are ready to program!
To enter program statements into the Card, the Card must first be placed in the PRO (program) mode using the [BASIC] key.
"PRO MODE" will be displayed followed by the prompt.

Enter the NEW command.

```
PRO MODE
NEW
>
```

The NEW command clears the memory of all existing programs and data. The prompt appears after you press the [ENTER] key, indicating that the Card is awaiting input.

Example 1 — Entering and Running a Program
Make sure the computer is in the PRO mode and enter the following program:

1 0 P R I N T " H E L L O "

```
PRO MODE
NEW
10PRINT"HELLO"_
```

Notice that the Card automatically inserts the colon between the number and the command when you press the [ENTER] key.

Check that the statement is in the correct format and then change the mode to RUN by pressing the [BASIC] key.

[C•CE] R U N [ENTER]

```
RUN
HELLO
>
```


Since this is the only line of the program, the Card will exit the program and return to the BASIC prompt " > ".

Example 2 — Editing a Program
Suppose you wanted to change the message that your program was displaying. That is, you wanted to edit your program. With a single line program you could just retype the entry, but as you develop more complex programs, editing becomes a very important component of your programming. Let's edit the program you have just written.

Are you still in the RUN mode? If so, switch back to the PRO mode.

You need to recall your program in order to edit it. Use the up arrow key [^] to recall your program. If your program was completely executed, the [^] key will recall the last line of the program. If there was an error in the program, or if you used the [ON] key to stop execution, the [^] key will recall the line in which the error or break occurred. To make changes in your program, use the [^] key to move up in your program (recall the previous line) and the [v] key to move down in your program (display the next line). If held down, the [^] or [v] key will scroll vertically (up or down) through your program.

Remember that to move the cursor within the program line you use the [►] (right arrow) and [◄] (left arrow) keys.

Using the [►] key, position the cursor over the first character you wish to change:

[^]

```
10:PRINT "HELLO"
```

[►] [►] [►] [►] [►]
[►] [►] [►]

```
10 PRINT "HELLO"
```

Notice that the cursor is now in the flashing block form, indicating that it is on top of an existing character. Enter:

G O O D B Y E " !

```
10 PRINT "GOODBY
   E"!_
```

Remember to press the [ENTER] key at the end of the line. Change to the RUN mode.

R U N [ENTER]

```
10:PRINT "GOODBY
   E"!
RUN
GOODBYE
Syntax error in
10
```

The error message indicates the type of error, and the line number in which the error occurred.

Press the [C•CE] key to clear the error condition, and return to the PRO mode. You must be in the PRO mode to make changes in a program. Using [^] (or [v]), recall the line in which the error occurred.

[^] (or [v])

```
10 PRINT "GOODBY
   E"!
```

The flashing cursor is positioned over the problem area. You learned, that when entering string constants in BASIC, all characters must be contained within quotation marks. Use the DELete key to eliminate the "!".

[DEL]

```
10 PRINT "GOODBY
   E"_
```

Now let's put the ! in the correct location. When editing programs, DELete and INSert are used in exactly the same way as they are in editing calculations. Using [◄], position the cursor on top of the character that will be the first character following the insertion.

[◄]

```
10 PRINT "GOODBY
   E"
```

Press the INSert key. A "◄" will indicate where the new data will be entered:

Enter the !.

[INS] !

```
10 PRINT "GOODBY
   E!"
```

Remember to press the [ENTER] key so the correction will be entered into the program.

Notes:

1. If you wish to DELete an entire line from your program, just enter the line number and the original line will be eliminated. The DELETE command can be used to delete more than one line at a time.
2. In the PRO mode, if keys are pressed when the cursor is not displayed, their corresponding characters are usually displayed from the leftmost column of the display. However, if the [►] or [◄] key is pressed when the cursor is displayed, successive key entries are displayed starting from the cursor position.

Example 3 — Using Variables in Programming

If you are unfamiliar with the use of numeric and string variables in BASIC, reread the appropriate sections in Chapter 2.

Using variables in programming allows more sophisticated use of the Card's abilities.

Remember, you assign simple numeric variables using any letter from A to Z:

```
A = 5
```

To assign string variables you also use a letter, followed by a dollar sign.

```
A$ = "TOTAL"
```

The values assigned to a variable can change during the execution of a program, taking on the values entered or computed during the program. One way to assign a variable is to use the INPUT command. In the following program, the value of A$ will change in response to the data typed in answer to the inquiry "WORD?".

Enter this program:

```
10: INPUT "WORD?";A$
20: B=LEN(A$)
30: PRINT "THE WORD (";A$;") HAS"
40: PRINT B;" LETTERS"
50: END
```

The second new element in this program is the use of the END statement to signal the completion of a program. END tells the Card that the program is completed. It is always good programming practice to use an END statement.

As your programs get more complex you may wish to review them before you begin execution. To look at your program, use the LIST command. LIST, which can only be used in the PRO mode, displays programs beginning with the lowest line number.

Try listing this program:
L I S T [ENTER]

```
10:INPUT "WORD?"
  ;A$
20:B=LEN (A$)
30:PRINT "THE WO
  RD (";A$;") H
  AS"
40:PRINT B;" LET
  TERS"
```

Use the [^] and [v] keys to move through your program until you have reviewed the entire program. After checking your program, change to the RUN mode and run it:

[C•CE] R U N [ENTER]

```
RUN
WORD?_
```

H E L P

```
RUN
WORD?HELP_
```

[ENTER]

```
RUN
WORD?HELP
THE WORD (HELP)
HAS
 4  LETTERS
>
```


This is the end of your program. Of course you may begin it again by entering RUN. However, this program would be a bit more entertaining if it presented more than one opportunity for input. We will now modify the program so it will keep running without entering RUN after each answer.

Return to the PRO mode and use the [^] or [v] key (or LIST) to reach line 50, or enter:

L I S T 5 0 [ENTER]

```
50:END
```

You may enter 50 to delete the entire line or use the [►] key to position the cursor over the E in END. Change line 50 so that it reads:

```
50: GOTO 10
```

Now RUN the modified program.

The GOTO statement causes the program to loop (keep repeating the same operation). Since you put no limit on the loop it will keep going forever (an "infinite" loop). To stop this program press the [ON] key.

When you have stopped a program using the [ON] key, you can restart it using the CONT command. CONT stands for CONTinue. With the CONT command the program will restart on the line that was being executed when the [ON] key was pressed.

Example 4 — More Complex Programming
The following program computes N factorial (N!). The program begins with 1 and computes N! up to the limit that you enter. Enter this program:

```
100: F = 1: WAIT 118
110: INPUT "LIMIT?";L
120: FOR N = 1 TO L
130: F = F*N
140: PRINT N,F
150: NEXT N
160: END
```

Several new features are contained in this program. The WAIT command in 100 controls the time that displays are held before the program continues. The numbers and their factorials are displayed as they are computed. The time they appear on the display is set by the WAIT statement to approximately 2 seconds.

Notice that there are two statements in line 100 separated by a colon (:). You may put as many statements as you wish on one line (separating each by a colon) up to a maximum of 254 characters including the [ENTER] key. Multiple-statement lines can make a program hard to read and modify, so it is good programming practice to use them only where the statements are very simple or there is some special reason to want the statements on one line.

In this program we have used the FOR command in line 120 and the NEXT command in line 150 to create a loop. In Example 3 you created an "infinite" loop that kept repeating the statements inside the loop until you pressed the [ON] key. With this FOR...NEXT loop, the computer adds 1 to N each time execution reaches the NEXT command. It then tests to see if N is larger that the limit L. If N is less than or equal to L, execution returns to the top of the loop and the statements are executed again. If N is greater than L, execution continues to line 160 and the program stops.

You may use any fixed numeric variable or single-precision simple numeric variable in a FOR...NEXT loop, you do not have to start counting at 1 and you can increment any amount at each step. See the BASIC COMMAND DICTIONARY for details.

We have labeled this program with line numbers starting with 100. Labeling programs with different line numbers allows you to have several programs in memory at one time. To RUN this program instead of the one at line 10, change to the RUN mode and enter:

[C•CE]
R U N 1 0 0

You could also give the program a name using a label and start the program with RUN *label.

Notes on the PRINT command:
If more than eight lines must be displayed, the first lines will scroll up off the dislay, and cannot be recalled. Use the PAUSE or WAIT command in the program to display data more slowly, or use the printer. (Refer to the PAUSE, WAIT or LPRINT command.)

The WAIT command applies to every PRINT command. Break long PRINT commands into a number of shorter commands if the display scrolls too quickly.

Example:

```
100 PRINT A, B, ..., P
          ↓
100 PRINT A, B, ..., H: PRINT I, J, ..., P
```

Since the WAIT command is not supported by many personal computers, a wait loop such as FOR J=1 TO 500:NEXT J can also be used to extend the display time.

Storing Programs in Memory

You will remember that settings and functions remain in the Card even after it is turned off. Programs also remain in memory after the power is turned off, or it undergoes an Auto OFF. Even if you use the [ON] or [C•CE] key, the programs will remain.

Programs are lost from memory only when you:

- Enter NEW before beginning programming in the PRO mode.
- Create a new program using the SAME LINE NUMBERS as a program already in memory.

Data Files

Following are some programming examples which use data file storage.

Creating a Sequential File
Program 1 is a short program that creates a sequential file, DATA, from information you input on the keyboard.

Program 1 — Creating a Sequential File

```
10: DIM DE$(1)
20: OPEN "E:DATA" FOR OUTPUT AS #20
30: CLS
40: INPUT "NAME: ";NA$
50: IF NA$ = "DONE" THEN 100
60: INPUT "DEPARTMENT: ";DE$(1)
70: INPUT "DATE HIRED: ";HI$
80: PRINT #20,NA$;",";DE$(1);",";HI$
90: GOTO 30
100: CLOSE #20
110: END
```

Before execution, enter: INIT "E:10K" [ENTER] to allocate storage space in the RAM disk E.

```
RUN
NAME: SAMUEL GOLDWYN
DEPARTMENT: AUDIO/VISUAL AIDS
DATE HIRED: 01/12/72
NAME: MARVIN HARRIS
DEPARTMENT: RESEARCH
DATE HIRED: 12/03/65
NAME: DEXTER HORTON
DEPARTMENT: ACCOUNTING
DATE HIRED: 04/27/81
NAME: DONE
```

Reading Data from a Sequential File

Now look at Program 2. It accesses the file DATA that was created in Program 1 and displays the name of everyone hired in 1981.

Program 2 — Accessing a Sequential File

```
10: DIM DE$(1)
20: OPEN "E:DATA" FOR INPUT AS #20
30: INPUT #20,NA$,DE$(1),HI$
40: IF RIGHT$(HI$,2)="81" THEN PRINT NA$
50: GOTO 30
```



```
RUN
DEXTER HORTON
Input past end in 30
```

Program 2 reads, sequentially, every item in the file, and prints the names of employees hired in 1981. When all the data has been read, line 30 causes an ERROR. To avoid this error, use the EOF function, which tests for the end-of-file. The revised program looks like this:

```
10: DIM DE$(1)
20: OPEN "E:DATA" FOR INPUT AS #21
25: IF EOF(21) THEN 60
30: INPUT #21,NA$,DE$(1),HI$
40: IF RIGHT$(HI$,2)="81" THEN PRINT NA$
50: GOTO 25
60: CLOSE #21
70: END
```

As shown by these programs, the following steps are required to create a sequential file and access the data in it:

1. OPEN the file for OUTPUT.
2. Write data to the file using the PRINT# statement.
3. CLOSE the file and reopen it in INPUT mode to read the data.
4. Use the INPUT# statement to read data from the file into the program.

Adding Data to a Sequential File

If you have an existing data file and want to add more data to the end of it, you cannot simply open the file in the OUTPUT mode and start writing data. As soon as you open a sequential file in the OUTPUT mode, you destroy its current contents.

Instead, use the APPEND mode. If the file does not already exist, the OPEN statement will work exactly as it would if the OUTPUT mode had been specified.

The following procedure can be used to add data to an existing file called "FOLKS".

Program 3 — Adding Data to a Sequential File

```
110: DIM AD$(0)
120: OPEN "E:FOLKS" FOR APPEND AS #22
130: REM ADD NEW ENTRIES TO FILE
140: CLS
150: INPUT "NAME?";NA$
160: IF NA$ = "00" THEN 230: REM 00 EXITS INPUT LOOP
170: INPUT "ADDRESS?";AD$(0)
180: INPUT "BIRTHDAY?";BI$
190: PRINT #22,NA$
200: PRINT #22,AD$(0)
210: PRINT #22,BI$
220: GOTO 140
230: CLOSE #22
```

Input 00 in answer to the question NAME? in line 150 to cause the program to jump out of the input loop in line 160. The REM statement can be used to write programming notes to yourself.

This brief introduction to programming should serve to illustrate the exciting programming possibilities of your new Card.

Program Execution

More than one program can be stored in this Card if the memory capacity is not exceeded. Execute the second or subsequent program using one of the following:

1. the RUN command: RUN line number [ENTER]
2. the GOTO command: GOTO line number [ENTER]

Execution begins from the specified line number. If a label such as *AB is entered in the program, the program can be executed by entering RUN*AB [ENTER].

The following lists the differences between the variables and status when a program is executed using the GOTO and RUN commands.

Execution using RUN	Execution using GOTO
• Clears the WAIT setting. • Clears the USING format. • Clears array and simple variables. • Clears double-precision operation mode and variable specifications. • Initializes the DATA statement for the READ statement. • Clears the PRINT=LPRINT setting. • Closes all the files • Clears the position set by LOCATE or GCURSOR. • Clears the error trap function. • Clears ERN and ERL variables.	• Retains the WAIT setting. • Retains the USING format. • Retains array and simple variables. • Retains double-precision operation mode and variable specifications. • Does not initialize the DATA statement for the READ statement. • Retains the PRINT=LPRINT setting. • Does not close all the files. • Retains the position set by LOCATE or GCURSOR. • Retains the error trap function. • Retains ERN and ERL variables.

Note:

When the program is executed using the RUN command, the variables for data are cleared. (Fixed variables are retained.) To retain the data, execute using the GOTO command.

4. DEBUGGING

er entering a new BASIC program, it often does not work the first ıe. Even if you are simply entering a program that you know is rrect, such as those provided in this manual, it is common to make ıleast one typing error. It may also contain at least one logic error as ıll.

llowing are some general hints on how to find and correct your lors. Suppose you run your program and get an error message:

Go back to the PRO mode and use the [^] or [v] key to recall the line with the error. The cursor will be positioned at the place in the line where the Card became confused.

If you cannot find an obvious syntax error, the problem may lie with the values that are being used. Check the values of the variables in either the RUN or PRO mode by entering the name of the variable and pressing the [ENTER] key.

ıppose you run the program and do not get an error message, but ə program does not do what you expect.

Check through the program line by line using LIST and the [^] and [v] keys to see if you have entered the program correctly. It is surprising how many errors can be corrected by just taking another look at the program.

Think about each line as you go through the program as if you were the Card. Take sample values and try to apply the operation in each line to see if you get the result that you expected.

Insert one or more extra PRINT statements in your program to display key values and key locations. Use these to isolate the parts of the program that are working correctly and the location of the error. This approach is also useful for determining which parts of a program have been executed. You can also use STOP to temporarily halt execution at critical points so that several variables can be examined.

4. Use TRON (TRace ON) and TROFF (TRace OFF), either as direct commands or within the program to trace the flow of the program through individual lines. Stop to examine the contents of critical variables at crucial points. This is a very slow way to find a problem, but it is sometimes the only way.

No matter how careful you are, eventually you will create a program that does not do quite what you expect it to. To isolate the problem, BASIC has a special method of executing programs known as the "Trace" mode.

TRON (TRace ON) starts Trace mode. The TRON instruction may be issued as a direct command (in RUN mode) or it may be embedded within a program. Used as a direct command, TRON informs the Card that tracing is required during the execution of all subsequent programs. The programs to be traced are then started in a normal manner, with a GOTO or RUN command.

If TRON is used as a statement, it will initiate the Trace mode only when the line containing it is executed. If, for some reason, that line is never reached, Trace mode will remain inactive.

Debugging Procedures

1. Set the computer to RUN mode.
2. Enter TRON [ENTER] to specify the trace mode.
3. Enter RUN [ENTER] to execute the program. The line number will be displayed at above right of the display for about 0.5 second after each line is executed.
4. Press the [ON] key when the desired line number is displayed. The break message is displayed and execution is interrupted. Press the [^] key to display the last statement executed. To resume execution, press the [SHIFT] [v] keys or enter CONT [ENTER]. However, if execution is interrupted during data entry using the INPUT command, just press the [ENTER] key as for usual program continuation.
5. Press the [v] key to move to the line to be checked. Holding the [v] key will execute the program step by step. Releasing the key will stop program execution.

6. Continue the trace procedure and check if the program is executing properly by confirming program execution order and variable contents after each line is executed. If the program is not executing properly, correct the logic.
7. After debugging, enter TROFF [ENTER] to exit the trace mode.

Example:

```
10 INPUT "A=";A,"B=";B
20 C=A*2
30 D=B*3
40 PRINT "C=";C;"  D=";D
50 END
```

Run the program.

RUN mode

TRON [ENTER]	>	
RUN [ENTER]	A=_	Execute INPUT command
8 [ENTER] (Data entry)	B=_	
9 [ENTER] (Data entry)		
	C= 16 D= 27	Execute PRINT command
	>	End of execution (prompt is displayed)

The executed line number will be displayed at the right top for about 0.5 second.

When execution is interrupted with the [ON] key, recall the variables manually and check that the values are as expected. Pressing the [V] key will execute one statement at a time and entering CONT [ENTER] will execute the statements continuously.

Note:
The trace mode will remain in effect unless TROFF [ENTER] is entered, the [SHIFT] [C•CE] keys are pressed, or the power is turned off.

To debug by interrupting program execution:
Perform one of the following:

- Press the [ON] key during program execution.
- Press the [ON] key in the trace mode.
- Enter the STOP command at the location to be stopped.

The Break message will be displayed and execution will be interrupted. Then

1. Check the variable contents manually.
2. Press the [V] key to execute subsequent statements line by line. Press the [SHIFT] [V] keys or enter CONT [ENTER] to return to previous operation.

- A program interrupted by the [ON] key or the STOP command can be executed line by line by pressing the [V] key. Trace messages will be displayed at each step.

PART 2

ALGEBRAIC AND STATISTICAL OPERATIONS

The Card allows algebraic expressions to be stored for repetitive calculations in the AER mode, and statistical calculations to be performed on single- or two-variable data in the STAT mode.

5. AER MODE

The Algebraic Expression Reserve (AER) mode is convenient for repetitive calculations. Algebraic expressions with multiple variables can be registered using all scientific functions.

Selecting and Cancelling the AER Mode

Selecting the AER Mode

Press the [AER] key to display the AER preparation message and then the AER menu. (The length of time this message is displayed depends on the number of registered AER expressions.)

```
   * AER *
1.EXECUTE
2.CORRECT
  & DELETE
3.RESERVE
9.QUIT

SELECT No. ?
```

Press the [1] key to execute expressions, the [2] key to correct or delete expressions, or the [3] key to register expressions.

Note:
If the AER mode is not entered when the [AER] key is pressed, there is not enough free area. Delete unnecessary data and programs from the memory.

Cancelling the AER Mode

Press the [9] key to exit the AER mode to return the Card to the BASIC mode.

```
RUN MODE
>
```

Note:
The contents of numeric, string or array variables which start with "Z" will not be retained after cancelling the AER mode.

Registering Expressions

Select "3. RESERVE" in the AER menu.

```
   * AER *
01:TITLE?
```

1. 01 ..... Indicates the expression number (1 to 20).
2. TITLE? .. Indicates the title of the expression. Up to 13 alphanumeric characters and symbols excluding quotes (") can be entered for a title.
   Press the [ENTER] key only to enter no title.

Note:
Do not use quotes (") in the titles or expressions. The data following the quotes will be ignored.

Entry format

Expressions are entered in the following format:

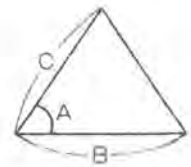
F (parameter, parameter, ...) = expression

- Only single-precision numerical variables can be used and the variable names must be a single character A through Z.
- Function keys such as [SIN], [COS] and [2nd F] [e^x] can be used.
- A comma (,) is required to separate the variables. (The commas are entered by pressing the [SMBL] key followed by the number beside the symbols.)
- The expression is stored by pressing the [ENTER] key at the end.
- When the memory is exceeded, "AER memory over" is displayed.
- Up to 95 characters or 95 bytes of an expression can be stored (8 bytes per value, 2 bytes per function and 1 byte per other character including the [ENTER] key).

After expression entry, the AER menu is displayed.

Example:
Enter the formula for the area of triangles.

$$\text{Area } S = \frac{B \cdot C \cdot \sin A}{2}$$



(Select "3. RESERVE" in the AER menu.)

AREA [ENTER]

```
    * AER *
01:AREA
:?
```

F (A, B, C) = B*C*SIN A/2

```
    * AER *
01:AREA
:F(A,B,C)=B*C*SI
N A/2_
```

[ENTER]

Entry of the formula for the area of triangles is complete, and the AER menu is displayed.

Example:
Enter the formula for the Pythagorean theorem.
(Select "3. RESERVE" in the AER menu.)

PYTHAGORAS [ENTER]
F (A,B) = SQR (A*A + B*B)

```
    * AER *
02:PYTHAGORAS
:F(A,B)=SQR(A*A+
B*B)_
```

[ENTER]

Recalling Expressions
Select "1. EXECUTE" or "2. CORRECT & DELETE" in the AER menu. Pressing the [AER] (title) key displays the currently stored titles and expressions sequentially.
To quit recall, press the [PLAY BACK] key to display the previous menu.

Example:
Recall the currently stored expressions.
(Select "1. EXECUTE" or "2. CORRECT & DELETE" in the AER menu.)

[AER] (title)

```
    * AER *
02:PYTHAGORAS
:F(A,B)=SQR(A*A+
B*B)
```

[AER] (title)

```
    * AER *
01:AREA
:F(A,B,C)=B*C*SI
N A/2
```

Correcting and Deleting Expressions

Correcting Expressions

(1) To correct the entered expression before pressing the [ENTER] key, position the cursor and correct with keys such as [DEL], [INS], [BS].

(2) To correct an expression already stored, select "2. CORRECT & DELETE" in the AER menu. Recall the expression to be corrected with the [AER] (title) key. Correct the entry as follows.

① Title correction
Press the [◄] or [►] key. The cursor moves to the position following the last character of the expression title. Position the cursor and correct the expression title. After the correction is complete, press the [ENTER] key. If the title is not to be corrected, just press the [ENTER] key.

② Expression correction
Next, the expression is displayed with the cursor over the last character of the expression. Position the cursor and correct the expression. After the correction is complete, press the [ENTER] key. If the expression is not to be corrected, just press the [ENTER] key. The AER menu is displayed.

Example:
Correct the No.1 expression title "AREA" to "AREA 1".
(Select "2. CORRECT & DELETE" in the AER menu.)

Recall the title "AREA" by pressing the [AER] (title) key.

```
    * AER *
01:AREA
:F(A,B,C)=B*C*SI
N A/2
```

[◄] [1] [ENTER]

```
    * AER *
01:AREA1
:F(A,B,C)=B*C*SI
N A/2_
```

[ENTER]

Now the correction is complete and the AER menu is displayed.

Deleting expressions
Carry out the following procedure to delete an expression already stored.

(1) Select "2. CORRECT & DELETE" in the AER menu and recall the expression to be deleted with the [AER] (title) key.
(2) Press the [DEL] key.
"01:AER data clear OK? (Y/N)" will be displayed to confirm the deletion. "01" is the expression title number of the expression to be deleted.
(3) Press the [Y] key to delete the expression or press the [N] key not to delete it.
(For now, press the [N] key. The expression stored in "01" will be used in future examples.)

Note:
When an expression is deleted, the deleted storage area is filled with the expression following the deleted expression, so some expression title numbers may change.

Executing an Expression

(1) Select "1. EXECUTE" in the AER menu and press the [AER] (title) key to recall the title and expression.
(2) When the expression to be executed is recalled, press the [ENTER] key.
(3) The entry prompt for the parameter(s) specified in F ( ) are displayed sequentially. Press the [ENTER] key after each numerical data entry.
- If the [ENTER] key is pressed without numerical data entry, the most recently entered value will be used.

(4) After parameter entry is completed, the calculated result will be displayed.
To quit execution, press the [PLAY BACK] key when the calculated result is being displayed.

Example:
Perform calculations using the stored expression "AREA1". Set the RUN mode, and set the angular unit to degrees.
Set the AER mode and select "1. EXECUTE" in the AER menu. Recall the title "AREA1" which expression is to be executed by pressing [AER] (title) key.

```
    * AER *
01:AREA1
:F(A,B,C)=B*C*SI
N A/2
```

[ENTER]
Enter the value for angle A.
30 [ENTER]

```
A=30
B=?
```

Enter the values for sides B and C.
100 [ENTER] 20 [ENTER]

```
A=30
B=100
C=20
                500
```

The area for the triangle is 500.
Pressing the [ENTER] key will execute the same expression again.

If the [PLAY BACK] key is pressed while the result is being displayed, the AER menu wil be displayed.

Error Messages

(1) AER format error
① The expression was entered in an incorrect format (parameter + expression).
② ")=" is missing after "F(".
Press the [C•CE] key to display the title and expression. Position the cursor, and correct the expression.
(2) AER memory over
An attempt was made to enter more than 20 expressions.
The AER menu will be displayed.
(3) Syntax error
An invalid expression or variable has been entered.
(4) Type mismatch
The type of the specified data does not match.
(5) Division by zero
An attempt was made to divide by zero.
(6) Overflow
The calculated result exceeds the calculation range.
(7) Illegal function call
An illegal operation was attempted.
(8) Data out of range
The specified value exceeds the allowable range.
(9) Out of memory
Not enough memory free area is available. Press the [ENTER] key to return to the RUN mode, and delete unnecessary data and/or programs.

Errors (3) to (8) may occur when entering values during the "1. EXECUTE" operation.
(1) If an invalid value is entered and an error occurs, press the [C•CE] key and re-enter the correct value.
(2) If an invalid expression is registered and an error occurs, press the [C•CE] key to display the title and expression where the error occurred. Correct the expression.

6. STAT MODE

The Card can perform statistical and regression calculations on one or two variables. With statistical calculations, you can obtain mean values, standard deviations, and other statistics from sample data. Regression calculation determines the coefficients of linear regression formulas or estimated values from sample data.

Selecting and Cancelling the STAT Mode

Selecting the STAT mode

Press the [STAT] key to display the STAT menu as shown.

```
* STATISTICAL
    ANALYSIS *
1.SINGLE-
    VARIABLE
2.TWO-VARIABLE
3.DISPLAY (DBL)
9.QUIT
SELECT No. ?
```

Press the [1] key to select single-variable statistical calculations, the [2] key to select two-variable statistical calculations or the [3] key to specify the display and print format (single- or double-precision).

Note:
If the STAT mode is not entered when the [STAT] key is pressed, or if "Out of memory" appears, there is not enough free area. Press the [ENTER] key to return to the RUN mode, and delete unnecessary data and programs from the memory.

Cancelling the STAT mode

Press the [9] key to exit the STAT mode. The Card returns to the BASIC mode.

```
RUN MODE
>
```

Note:
The contents of numeric, string, or array variables which start with "Z" will not be retained after cancelling the STAT mode.

Display format
To specify the display and print format, press the [3] key in the STAT menu.

```
* DISPLAY MODE *
 D:DBL Precision
 S:SNG Precision
```

Press the [D] key to specify double-precision format or the [S] key for single-precision format.
If you do not wish to change the format specification, press the [PLAY BACK] key. The STAT menu will be displayed, with (SNG) to indicate single-precision and (DBL) to indicate double-precision.

Note:
All calculations are performed in double-precision. Only the display and print out change.

Single-Variable Statistical Calculations

Single-variable calculations
The following statistics can be obtained from single-variable statistical calculations.

n: Sample size of x
$\bar{x}$: Sample mean x
Σx: Sum of samples x
Σx^2: Sum of squares of samples x
s: Sample standard deviation with population parameter taken to be $n-1$.

$$s = \sqrt{\frac{\Sigma x^2 - n\bar{x}^2}{n-1}}$$

This equation is used to estimate the standard deviation of a population from sample data (x) extracted from that population.

σ: Population standard deviation with population parameter taken to be n.

$$\sigma = \sqrt{\frac{\Sigma x^2 - n\bar{x}^2}{n}}$$

This equation lets you assume the entire population as sample data (x) or determine the standard deviation of the sample data which is taken to be a population.

Selecting single-variable statistical calculations
After displaying the STAT menu, press the [1] key to select single-variable statistical calculations. The single-variable submenu will be displayed.

[STAT] [1]

```
FUNCTIONS (x)
1. INPUT
2. DELETE
3. ANALYSIS
4. PRINTER
SELECT No. ?
```

The following can be selected from the submenu:

[1] ... INPUT: Used for entering data.
[2] ... DELETE: Used for deleting the entered data if the data was incorrect or to start a new calculation.
[3] ... ANALYSIS: Used for obtaining statistics.
[4] ... PRINTER: Used for printing the obtained statistics. Available only if the optional printer is connected to the Organizer.

Press the [PLAY BACK] key to display the STAT menu.

Entering data
Press the [1] key to display the data input prompt. Enter the data.

[1]

```
DATA INPUT(x)
1:x=_
```

Data to be entered.
Indicates the number of data entered.

To enter a single data value,
press: data [ENTER].
To enter multiple identical data values simultaneously,
press: data , frequency [ENTER].
To enter negative values,
press: [−] data [ENTER].

Press the [ENTER] key to end data entry and display the single-variable submenu.

Note:
In statistics, "frequency" is used to define the number of identical data. For example, if three identical data occur consecutively, frequency 3 is used.

Obtaining statistics
Press the [3] key in the single-variable submenu to display the analysis submenu.

[3]

```
ANALYSIS (x)
1:n      2:Σx
3:Σx²    4:x̄
5:s      6:σ

SELECT No. ?
```

The following statistics can be obtained by pressing keys [1] to [6]:

[1]	..... n:	The sample size of data
[2]	..... Σx:	Sum of samples
[3]	..... Σx^2:	Sum of squares of samples
[4]	..... $\bar{x}$:	Sample mean
[5]	..... s:	Sample standard deviation
[6]	..... σ:	Popular standard deviation

Press the [PLAY BACK] key to return to the single-variable submenu.

Note:
If an error occurs, press the [C•CE] key to clear the error. Check the entered data.

Starting a new calculation (clearing the previous data)
Perform one of the followings:
1. Select "9. QUIT" to exit the STAT mode and select the STAT mode again. The previous data will be cleared.
2. Delete data using the delete/clear function. Press the [2] key in the single-variable submenu to display the delete/clear submenu.

[2]

```
DELETE (x)
1. DATA
2. ALL CLEAR

SELECT No. ?
```

The delete/clear submenu will be displayed.

[2]

```
ALL CLEAR (x)
1.YES
2.NO

SELECT No. ?
```

The all clear submenu will be displayed.

Press the [1] key to clear the previous data or the [2] key to retain the previous data.

Example:
The test scores for 35 randomly selected students are as shown. Determine the mean and standard deviation of these scores.

No.	Score	Frequency	No.	Score	Frequency
1	30	1	5	70	8
2	40	1	6	80	9
3	50	4	7	90	5
4	60	5	8	100	2

Select "1. SINGLE-VARIABLE" in the STAT menu.

[1]

```
DATA INPUT(x)
1:x=_
```

Select "1. INPUT" and enter the data.

[1]
30 [ENTER] 40 [ENTER] 50,4 [ENTER]
60,5 [ENTER] 70,8 [ENTER] 80,9 [ENTER]
90,5 [ENTER] 100,2 [ENTER]
Data entry is now complete.

```
2:x=40
3:x=50,4
7:x=60,5
12:x=70,8
20:x=80,9
29:x=90,5
34:x=100,2
36:x=_
```

Display the single-variable submenu.
[ENTER]

Select "3. ANALYSIS".
[3]

```
ANALYSIS (x)
1:n      2:Σx
3:Σx²    4:x̄
5:s      6:σ

SELECT No. ?
```

Obtain the mean.
[4]

```
ANALYSIS (x)
1:n      2:Σx
3:Σx²    4:x̄
5:s      6:σ

x̄= 71.42857143
```

Obtain the population standard deviation.
[6]

```
ANALYSIS (x)
1:n      2:Σx
3:Σx²    4:x̄
5:s      6:σ

σ= 16.23802542
```

Return to the single-variable submenu.
[PLAY BACK]

- Press [1], [2], [3], or [5] key to obtain the sample size, sum, sum of squares, or standard deviation of samples.
  After obtaining intermediate statistics, such as mean and standard deviation, further data can be entered by selecting "1. INPUT" in the single-variable submenu.

Deleting data
This function can be used for correcting the entered data. Press the following keys in the single-variable submenu to display the data clear prompt.

[2]

```
DELETE (x)
1. DATA
2. ALL CLEAR

SELECT No. ?
```

[1]

```
DATA CLEAR(x)
x=_
```

Enter the values of the incorrect data or data to be deleted in the same manner as for data entry. Multiple data values can be deleted using the commas (,).
After deletion, correct data may be entered from the data input prompt.

Printing statistics
The calculated statistics can be printed on the optional CE-50P printer. Connect the printer to the Organizer and turn the power on. Enter the data and select "4. PRINTER" in the single-variable submenu to print the statistics.

```
FUNCTIONS (x)
1. INPUT
2. DELETE
3. ANALYSIS
4. PRINTER
SELECT No. ?
```

[4]

Printing example:

```
n=                35
Σx=             2500
Σx²=          187800
MEAN(x)=
   71.42857143
s=
   16.47508942
σ=
   16.23802542
```

(with single-precision specified)

```
n=                35
Σx=             2500
Σx²=          187800
MEAN(x)=
   71.428571428571
428571
s=
   16.475089420958
279842
σ=
   16.238025415294
653431
```

(with double-precision specified)

After printing, the display will return to the single-variable submenu.

Two-Variable Statistical Calculations

Operations for two-variable statistical calculations are similar to operations for single-variable statistical calculations. Read the section for the single-variable statistical calculations first.

Two-variable calculations

The following statistics can be determined from two-variable statistical calculations.

n, Σx, Σx^2, and $\bar{x}$: Same as for single-variable calculations

sx and σx: Same as s and σ.

$\bar{y}$: Sample mean y

Σy: Sum of samples y

Σy^2: Sum of squares of samples y

sy: Sample standard deviation with population parameter taken to be n–1.

$$\bar{y} = \sqrt{\frac{\Sigma y^2 - n\bar{y}^2}{n-1}}$$

σy: Population standard deviation of samples (y) with population parameter taken to be n.

$$\sigma y = \sqrt{\frac{\Sigma y^2 - n\bar{y}^2}{n}}$$

Σxy: Sum of products of samples x and y

a: $a = \bar{y} - b\bar{x}$

Coefficient for linear regression $y = a+bx$

b: $b = \frac{Sxy}{Sxx}$

Coefficient for linear regression $y = a+bx$

r: $r = \frac{Sxy}{\sqrt{Sxx \cdot Syy}}$

Correlation coefficient

x': $x' = \frac{y-a}{b}$

Estimated value (x estimated from y)

y': $y' = a+bx$

Estimated value (y estimated from x)

Note:

$$Sxx = \Sigma x^2 - \frac{(\Sigma x)^2}{n}$$

$$Syy = \Sigma y^2 - \frac{(\Sigma y)^2}{n}$$

$$Sxy = \Sigma xy - \frac{\Sigma x \cdot \Sigma y}{n}$$

Selecting two-variable statistical calculations

After displaying the STAT menu, press the [2] key to select the two-variable statistical calculations.

Entering data

Press the [1] key on the two-variable submenu to display the data input prompt. Enter the x and y data as shown in the display.

To enter a single pair of data values,

press: data x [ENTER] data y [ENTER].

To enter multiple identical pairs of data values simultaneously,

press: data x [ENTER] data y, frequency [ENTER].

To enter a negative value,

press the [(–)] key before the value.

Press the [ENTER] key to end data entry and display the two-variable submenu.

Obtaining statistics

Press the [3] key on the two-variable submenu to display the analysis submenu. There are two submenus, pressing the [▲] or [▼] key toggles the submenu. (The symbol ↑ or ↓ will appear.) When the [3] key is pressed in the two-variable submenu,

(The first analysis submenu)

```
ANALYSIS (x,y)
1:n      2:Σx
3:Σx²    4:x̄
5:sx     6:σx
7:Σy     8:Σy²

SELECT No. ?      ↓
```

[▼]

(The second analysis submenu)

```
ANALYSIS (x,y)
1:Σxy 2:ȳ
3:sy  4:σy
5:a   6:b
7:r   8:x'  9:y'

SELECT No. ?      ↑
```

Pressing the [▲] key will display the first analysis submenu.

Example:

The following table lists the dates (April) on which migratory birds fly through a certain district, versus the average temperatures in March of the same district. From this table determine the coefficients, a and b, of the linear regression line, $y = a + bx$, and correlation coefficient r. Estimate the date of migration when the mean temperature in March is 9.1°C. Also estimate the mean temperature in March if the date of migration is April 10.

Year	1	2	3	4	5	6	7	8
Mean temp. (x°C)	6.2	7.0	6.8	8.7	7.9	6.5	6.1	8.2
Date of migration (y day)	13	9	11	5	7	12	15	7

Select "2. TWO-VARIABLE" in the STAT menu.

[2]

```
FUNCTIONS (x,y)

1. INPUT
2. DELETE
3. ANALYSIS
4. PRINTER

SELECT No. ?
```


Select "1. INPUT" and enter the data..

[1]

6.2 [ENTER] 13 [ENTER] 7.0 [ENTER] 9 [ENTER]
6.8 [ENTER] 11 [ENTER] 8.7 [ENTER] 5 [ENTER]
7.9 [ENTER] 7 [ENTER] 6.5 [ENTER] 12 [ENTER]
6.1 [ENTER] 15 [ENTER] 8.2 [ENTER] 7 [ENTER]

```
  y=7
6:x=6.5
  y=12
7:x=6.1
  y=15
8:x=8.2
  y=7
9:x=_
```

Data entry is now complete.
Display the two-variable submenu.

[ENTER]

Select "3. ANALYSIS" and display the second analysis submenu.

[3] [▼]

Determine coefficient a.

[5]

```
ANALYSIS (x,y)
1:Σxy 2:ȳ
3:sy  4:σy
5:a   6:b
7:r   8:x'  9:y'

a= 34.44951017    ↑
```

Determine coefficient b.

[6]

```
ANALYSIS (x,y)
1:Σxy 2:ȳ
3:sy  4:σy
5:a   6:b
7:r   8:x'  9:y'

b=-3.425018839    ↑
```

Estimate the date of migration.

[9]

```
ANALYSIS (x,y)
x=_
```

Enter the mean temperature.

9.1 [ENTER]

```
ANALYSIS (x,y)
x=9.1
  y= 3.281838734
x=_
```

(Estimated date: April 3)

Display the second analysis submenu.

[ENTER]

Estimate the mean temperature.

8

```
ANALYSIS (x,y)
y=_
```

Enter the date of migration.

10 ENTER

```
ANALYSIS (x,y)
y=10
  x= 7.13850385
y=_
```

(Estimated mean temperature on March 10: approx. 7.1°C)

Reference:

In Statistical calculations, the following will be stored into double-precision variables Z#, Z0# to Z4#, and be retained even after the STAT mode is exited. Therefore, these statistics can be used in the RUN mode for further calculations.

Variable		Z4#	Z3#	Z2#	Z1#	Z0#	Z#
Statistics	Single variable	—	—	—	Σx^2	Σx	n
	Two variable	Σy^2	Σy	Σxy	Σx^2	Σx	n

These values will be cleared when the STAT mode is selected again.

PART 3

BASIC REFERENCE

Part 3 contains alphabetical listings of all the BASIC commands supported by the Card and can be used as a ready reference.

The first section contains an alphabetical listing of numeric functions and pseudovariables.

The second section is an alphabetical listing of all other BASIC commands.

7. SCIENTIFIC & MATHEMATICAL CALCULATIONS

The Card has a wide range of built-in functions for scientific, mathematical and statistical calculations. They are listed below alphabetically. All the functions listed below can be used as part of calculations when using the Card in RUN mode. They may also be used as BASIC commands within programs.

For trigonometric functions, entries can be made in degrees, radians or as a gradient value, as appropriate:

DEGREE: Set the Card to degree entry mode by typing DEGREE (DEG is highlighted on the MODE CHECK display), this is the default mode.

RADIAN: Set the Card to radian entry mode by typing RADIAN (RAD is highlighted on the MODE CHECK display).

GRADIENT: Set the Card to gradient entry mode by typing in GRAD (GRAD is highlighted on the MODE CHECK display).

These three modes (DEG, RAD, and GRAD) can also be set from within a program. Once one mode is set, all entries for trigonometric functions must be in the units set (degrees, radians, or gradient values) until the mode is changed either manually or from within a program. The mode setting is preserved even when the power is turned off. The examples given below are all for direct entry of the functions entered in degrees.

Functions marked **DBL** can be used in both double-precision and single-precision modes. Functions marked **SNG** can be used only in single-precision mode, where double-precision arguments are first converted to a single-precision value and then used in calculations.

Many functions can also be implemented by pressing the corresponding function key. Functions marked (*) have no corresponding key and must be entered through the keyboard.

*ABS |x|

Function: Absolute value **DBL**

Remarks: Returns the absolute value of the numeric argument. The absolute value is the magnitude of the number irrespective of its sign. ABS–10 is 10.

Example: ABS –10 [ENTER] 10

ACS $\cos^{-1}x$

Function: Inverse or arc cosine **DBL**

Remarks: Returns the arc cosine of the numeric argument. The arc consine is the angle whose cosine is equal to the argument. The value returned depends on the mode (DEG, RAD or GRAD).

Example: DEGREE [ENTER]
ACS –0.5 [ENTER] 120

*AHC $\cosh^{-1}x$

Function: Inverse hyperbolic cosine **DBL**

Remarks: Returns the inverse hyperbolic cosine of the numeric argument.

Example: AHC 10 [ENTER] 2.993222846

*AHS $\sinh^{-1}x$

Function: Inverse hyperbolic sine **DBL**

Remarks: Returns the inverse hyperbolic sine of the numeric argument.

Example: AHS 27.3 [ENTER] 4.000369154

*AHT — $\tanh^{-1}x$

Function: Inverse hyperbolic tangent **DBL**

Remarks: Returns the inverse hyperbolic tangent of the numeric argument.

Example: AHT 0.7 [ENTER] 8.673005277E–01

ASN — $\sin^{-1}x$

Function: Inverse or arc sine **DBL**

Remarks: Returns the arc sine of the numeric argument. The arc sine is the angle whose sine is equal to the argument. The value returned depends on the mode (DEG, RAD or GRAD).

Example: DEGREE [ENTER]
ASN 0.5 [ENTER] 30

ATN — $\tan^{-1}x$

Function: Inverse or arc tangent **DBL**

Remarks: Returns the arc tangent of the numeric argument. The value returned depends on the mode (DEG, RAD or GRAD).

Example: DEGREE [ENTER]
ATN 1 [ENTER] 45

COS — cosx

Function: Cosine **DBL**

Remarks: Returns the cosine of the angle argument. The value returned depends on the mode (DEG, RAD or GRAD).

Example: DEGREE [ENTER]
COS 120 [ENTER] –0.5

*CUB — x^3

Function: Cube **DBL**

Remarks: Returns the cube of the argument.

Example: CUB 3 [ENTER] 27

CUR — $\sqrt[3]{x}$

Function: Cube root **DBL**

Remarks: Returns the cube root of the argument.

Example: CUR 125 [ENTER] 5

DECI

Function: Hexadecimal to decimal conversion **SNG**

Remarks: Converts a hexadecimal value to its decimal equivalent.

Example: DECI F82 [ENTER] 3970

DEG — dd°mm′ss″ → ddd.dddd°

Function: Deg/min/sec to decimal conversion **DBL**

Remarks: Converts an angle argument in DMS (Degrees, Minutes, Seconds) format to DEG (Decimal Degrees) format. In DMS format the integer portion of the number represents degrees, the first and second digits after the decimal point represent minutes, the third and fourth digits after the decimal point represent seconds, and any further digits represent fractional seconds.

Example: DEG 30.5230 [ENTER] (30°52′30″) 30.875

DMS — ddd.dddd° → dd°mm'ss"

Function: Decimal to deg/min/sec conversion **DBL**

Remarks: Converts an angle argument in DEG format to DMS format (see DEG).

Example: DMS 124.8055 [ENTER] 124.48198 (124°48'19"8)

EXP — e^x

Function: Exponential function **DBL**

Remarks: Returns the value of e (2.718281828... the base of natural logarithms) raised to the value of the numeric argument. The corresponding function key is [e^x].

Example: EXP 1.2 [ENTER] 3.320116923

FACT — n!

Function: Factorial n **DBL**

Remarks: Returns the factorial of the argument.

Example: FACT 7 [ENTER] 5040

*HCS — cosh x

Function: Hyperbolic cosine **DBL**

Remarks: Returns the hyperbolic cosine of the numeric argument.

Example: HCS 3 [ENTER] 10.067662

HEX

Function: Decimal to hexadecimal conversion **SNG**

Remarks: Converts a decimal value to its hexadecimal equivalent.

Example: HEX 7820 [ENTER] 1E8C

*HSN — sinh x

Function: Hyperbolic sine **DBL**

Remarks: Returns the hyperbolic sine of the numeric argument.

Example: HSN 4 [ENTER] 27.2899172

*HTN — tanh x

Function: Hyperbolic tangent **DBL**

Remarks: Returns the hyperbolic tangent of the numeric argument.

Example: HTN 0.9 [ENTER] 7.162978702E–01

*INT

Function: Integer **DBL**

Remarks: Returns the integer portion of the argument. The integer portion of PI is 3.

Example: INT –1.9 [ENTER] –2

LN $\log_e x$

Function: Natural or Naperian logarithm **DBL**

Remarks: Returns the logarithm to the base e (2.718281828 ...) of the numeric argument.

Example: LN 2 [ENTER] 6.931471806E–01

LOG $\log_{10} x$

Function: Common logarithm **DBL**

Remarks: Returns the logarithm to the base 10 of the numeric argument.

Example: LOG 1000 [ENTER] 3

*NCR ${}_nC_r=n!/r!(n-r)!$

Function: Combination **DBL**

Remarks: Enter the values as NCR(n,r).

Example: NCR (6,3) [ENTER] 20

*NPR ${}_nP_r=n!/(n-r)!$

Function: Permutation **DBL**

Remarks: Enter the values as NPR(n,r).

Example: NPR (6,3) [ENTER] 120

PI π

Function: PI **DBL**

Remarks: PI is a numeric pseudovariable that has the value of π. Use of PI is identical to use of the [π] key. The value of PI has 10-digit accuracy in single-precision (DEFSNG) mode, and 20-digit accuracy in double-precision (DEFDBL) mode.

Example: DEFDBL [ENTER]
PI [ENTER] 3.14159265358979323 85#

POL $(x,y) \rightarrow (r,\theta)$

Function: Rectangular to Polar coordinate conversion **SNG**

Remarks: Converts numeric arguments of rectangular coordinates to their polar coordinate equivalents.

The first argument indicates the distance from the y-axis and the second the distance from the x-axis. The values converted indicate the distance from the origin and the angle in the polar coordinates, and are assigned to the fixed variables Y and Z respectively. The angle depends on the mode (DEG, RAD, or GRAD).

Example: DEGREE [ENTER]
POL (8,6) [ENTER]
Z [ENTER] 10 (r = 10)
36.86989765
($\theta \approx 36.9°$)

^ y^x

Function: x^{th} power **DBL**

Remarks: Returns the x^{th} power of the numeric argument. Enter as y [Y^x] x or y ^ x.

Example: 4 ^ 2.5 [ENTER] 32

RCP — 1/x

Function: Reciprocal **DBL**

Remarks: Returns the reciprocal of the numeric argument.

Function: RCP 4 [ENTER] 0.25

REC — (r,θ) → (x,y)

Function: Polar to rectangular coordinate conversion **SNG**

Remarks: Converts numeric arguments of polar coordinates to their rectangular coordinate equivalents.

The first argument indicates the distance from the origin and the second argument the angle. The angle depends on the mode (DEG, RAD or GRAD). The converted values indicate the distances from the y-axis and the x-axis, and are assigned to the fixed variables Y and Z, respectively.

Example: DEGREE [ENTER]
REC (12,30) [ENTER] 10.39230485 (x ≈ 10.4)
Z [ENTER] 6 (y = 6)

*RND

Function: Random number **DBL**

Remarks: See RND and RANDOMIZE in the BASIC COMMAND DICTIONARY.

ROT — $\sqrt[x]{y}$

Function: x^{th} root **DBL**

Remarks: Returns the x^{th} root of the argument y. Enter as y ROT x.

Example: 7776 ROT 5 [ENTER] 6

*SGN

Function: Sign of argument **DBL**

Remarks: Returns a value based on the sign of the argument.

If x > 0, the function returns 1.
If x < 0, the function returns –1.
If x = 0, the function returns 0.

SIN — sin x

Function: Sine **DBL**

Remarks: Returns the sine of the angle argument. The value returned depends on the mode (DEG, RAD or GRAD).

Example: DEGREE [ENTER]
SIN 30 [ENTER] 0.5

SQR — $\sqrt{x}$

Function: Square root **DBL**

Remarks: Returns the square root of the argument.

Example: SQR 3 [ENTER] 1.732050808

SQU — x^2

Function: Square **DBL**

Remarks: Returns the square of the argument.

Example: SQU 4 [ENTER] 16

TAN — tan x

Function: Tangent — **DBL**

Remarks: Returns the tangent of the angle argument. The value returned depends on the mode (DEG, RAD or GRAD).

Example: DEGREE [ENTER]
TAN 45 [ENTER] — 1

TEN — 10^x

Function: Antilogarithm — **DBL**

Remarks: Returns the value of 10 (the base of the common log) raised to the value of the numeric argument.

Example: TEN 3 [ENTER] — 1000

Calculation Ranges

Numerical Calculations:
For a calculation involving x, the number x must be within one of the ranges below:

$-1 \times 10^{100} < x \leq -1 \times 10^{-99}$ for negative x
$1 \times 10^{-99} \leq x < 1 \times 10^{100}$ for positive x
$x = 0$

Functions:

Function	Range of X
sin x cos x tan x	DEG: $\|x\| < 1 \times 10^{10}$ Single-precision $\|x\| < 1 \times 10^{20}$ Double-precision RAD: $\|x\| < (\pi/180) \times 10^{10}$ Single-precision $\|x\| < (\pi/180) \times 10^{20}$ Double-precision GRAD: $\|x\| < (10/9) \times 10^{10}$ Single-precision $\|x\| < (10/9) \times 10^{10}$ Double-precision Also, for tan x only: (n=integer) DEG: $\|x\| \neq 90\ (2n-1)$ RAD: $\|x\| \neq \pi/2\ (2n-1)$ GRAD: $\|x\| \neq 100\ (2n-1)$
$\sin^{-1} x$ $\cos^{-1} x$	$-1 \leq x \leq 1$
$\tan^{-1} x$	$\|x\| < 1 \times 10^{100}$
sinh x cosh x tanh x	$-227.9559242 \leq x \leq 230.2585092$ Single-precision $-227.95592420641052271 \leq x \leq 230.25850929940456840$ Double-precision
$\sinh^{-1} x$	$\|x\| < 1 \times 10^{50}$
$\cosh^{-1} x$	$1 \leq x < 1 \times 10^{50}$
$\tanh^{-1} x$	$\|x\| < 1$
ln x log x	$1 \times 10^{-99} \leq x < 1 \times 10^{100}$
e^x	$-1 \times 10^{100} < x \leq 230.2585092$ Single-precision $-1 \times 10^{100} < x \leq 230.25850929940456840$ Double-precision
10^x	$-1 \times 10^{100} < x < 100$
$\sqrt[3]{x}$	$\|x\| < 1 \times 10^{100}$
1/x	$\|x\| < 1 \times 10^{100}, x \neq 0$
x^2	$\|x\| < 1 \times 10^{50}$

Function	Range of x	
$\sqrt{x}$	$0 \le x < 1 \times 10^{100}$	
n!	$0 \le n \le 69$ (n=integer)	
DMS→DEG DEG→DMS	$\lvert x \rvert < 1 \times 10^{100}$	
y^x $(y = 10^{x \log y})$	when $y > 0$, $-1 \times 10^{100} < x \log y < 100$ when $y = 0$, $x > 0$ when $y < 0$, $\begin{cases} x = \text{integer or } \frac{1}{x} = \text{odd integer } (x \neq 0) \\ \text{and } -1 \times 10^{100} < x \log \lvert y \rvert < 100 \end{cases}$	
$\sqrt[x]{y}$ $(\sqrt[x]{y} = 10^{\frac{1}{x} \log y})$	when $y > 0$, $-1 \times 10^{100} < \frac{1}{x} \log y < 100$, $x \neq 0$ when $y = 0$, $x > 0$ when $y < 0$, $\begin{cases} x \text{ or } \frac{1}{x} \text{ must be non-zero integer,} \\ \text{and } -1 \times 10^{100} < \frac{1}{x} \log \lvert y \rvert < 100 \end{cases}$	
DECI→HEX	$\lvert x \rvert \le 9999999999$, x = integer	
HEX→DECI	$0 \le x \le 2540BE3FF$ (x in hexadecimal) $FDABF41C01 \le x \le FFFFFFFFFF$	
x, y→r, θ	$(x^2 + y^2) < 1 \times 10^{100}$ $y/x < 1 \times 10^{100}$	$\begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}(y/x) \end{cases}$
r, θ→x, y	$r < 1 \times 10^{100}$ $\lvert r \sin\theta \rvert < 1 \times 10^{100}$ $\lvert r \cos\theta \rvert < 1 \times 10^{100}$	$x = r \cos\theta$ $y = r \sin\theta$
nPr	$0 \le r \le n < 1 \times 10^{100}$	n, r integers
nCr	$0 \le r \le n < 1 \times 10^{100}$ when $n-r < r$, $n-r \le 69$ when $n-r \ge r$, $r \le 69$	n, r integers

- The following functions are single-precision functions only. Before using them, double-precision values are converted to single-precision values: DECI, HEX, POL, REC

8. BASIC COMMAND DICTIONARY

The following pages contain an alphabetic listing of the BASIC commands that you can use on the Card.

For simplicity, the following conventions have been adopted in compiling this dictionary:

expression Indicates a numeric value, numerical variable or a formula including numeric values and numerical variables.

variable Indicates a numerical variable or string variable including array variables.

"string" Indicates a character string enclosed in quotation marks.

string variable Indicates a string variable or string array variable.

∗label Indicates ∗label.
(Although both ∗label and "label" forms may be used with this Card, ∗label is recommended. ∗AB and "AB" are different labels.)

d: Indicates a device name.
The following are device names used on the Card:

E:	RAM disk E (Card memory)
PACOM:	4-pin device
CAS:	Cassette tape
COM:	Serial I/O device (communication)

Note:
Device name PACOM refers to the IC Card (OZ-707 or OZ-770) installed in another Organizer to or from which a program or data is transferred. The optional CE-200L Data Transfer Cable is required to connect the two Organizers for data transfer.

[ ] The parameter in square brackets is optional. The brackets themselves are not part of the command entry.

( ) Used to enclose parameter values in certain commands. They should be entered as part of the command.

" " Used to enclose string parameter values in certain commands.

{A / B} A or B can be selected.

P Program execution is possible.
D Direct input operation is possible.

Abbreviation Most of the commands can be abbreviated. The shortest abbreviation allowed is given in this manual.
Example:
Abbreviation: P. (for PRINT)
The following abbreviations are also valid:
PR. PRI. PRIN.

ARUN

P

FORMAT: ARUN [{line number / *label}]

Abbreviation: AR.
See Also: AUTOGOTO, RUN

PURPOSE:
Starts a program automatically when the power is turned on and the RUN mode is specified.

REMARKS:
Include ARUN as the first program statement (in the first line of the program) to start the program as soon as the power is turned on. This has the same effect as if a RUN command had been entered from the keyboard.

If the power is turned off in the CARD mode, the Card will be in the RUN mode when the power is turned on again, and the program will be executed automatically.

*label must be the first statement of a line within the program, and must consist of alphanumeric characters or symbols.

Turning the power on after it has been turned off by the Auto OFF function will not execute the program automatically.

ARUN is similar to AUTOGOTO except that all variables and arrays other than fixed variables are cleared before program execution.

EXAMPLE:
```
 5: ARUN
10: CLS:WAIT 100
20: PRINT "WELCOME TO THE WORLD OF"
30: PRINT "THE CARD"
40: PRINT "YOU HAVE";FRE0;" BYTES FREE"
50: END
```
The program runs automatically when the power is turned on.

ASC

P
D

FORMAT: ASC {"string" / string variable}

Abbreviation: A.
See Also: CHR$

PURPOSE:
Returns the character code for the first character in the specified string.

REMARKS:
Specify the string as the contents of a string variable in the form X$ or as an actual string enclosed in quotes, "XXXX". Only the character code of the first character in the string is returned. See Appendix B for character code tables.

EXAMPLE:
```
10: INPUT "ENTER A CHARACTER ";A$
20: N = ASC A$
30: PRINT "THE CHARACTER CODE IS ";N
40: GOTO 10
```

[10] The user presses a key to enter a character.
[20] ASC finds the code number for the character.
[30] The answer is displayed.
[40] Repeats until the user halts the program by pressing the [ON] key.

AUTO

D

FORMAT: AUTO [[starting line number][,increment]] [ENTER]

Abbreviation:
See Also:

PURPOSE:
Provides automatic insertion of program line numbers in the PRO mode.

REMARKS:
Valid only as direct input in the PRO mode.
Starting line number and incremental value may be specified.
If not specified, the first line number is automatically set to 10 and the increment to 10. However, if the AUTO command has been previously set to other values, those values are used.
An error is generated if the starting line number exceeds 65279.

When the mode is changed from PRO to RUN and then back to PRO mode, entering AUTO [ENTER] assumes the previously set increment and resumes line numbering from the most recently generated line number.
Pressing the [C•CE] key while a line number is displayed will have the same effect.

Turning the power off and then on, or entering an operation mode other than PRO or RUN will exit the AUTO mode.

AUTOGOTO

P

FORMAT: AUTOGOTO {line number / *label}

Abbreviation: AU.
See Also: ARUN, GOTO

PURPOSE:
Starts a program automatically when the power is turned on and the RUN mode is specified.

REMARKS:
Include AUTOGOTO in the first line of a program (the first statement) to start the program when the power is turned on and is in the RUN mode. This functions as if a GOTO command had been entered from the keyboard. AUTOGOTO is similar to ARUN but does not clear all variables and arrays before starting the program. If the power is turned off in the CARD mode, the Card will be in the RUN mode when the power is turned on again, and the program will be executed automatically.

Turning the power on after it has been turned off by the Auto OFF function will not execute the program automatically.

*label must be the first statement of a line within the program, and must consist of alphanumeric characters or symbols.

BASIC

D

FORMAT: BASIC [ENTER]

Abbreviation: BA.
See Also: TEXT

PURPOSE:
Clears the text mode.

REMARKS:
Valid only as direct input in the PRO mode.

Executing this command clears the Text mode and returns the mode to BASIC. As the mode returns to BASIC, the prompt symbol changes from "<" to ">". Changing from the Text mode to the BASIC mode usually converts the text held in the Card memory to a program (internal code).
All lowercase letters other than those in character strings enclosed in quotation marks (" ") or following a REM (') statement are automatically converted to uppercase letters.
Abbreviations such as "P." and "I." are not converted to their respective commands. (To do so, move the cursor to the line and press the [ENTER] key.) Because of the characteristics of the BASIC function, commands and formats included in the text but not found in the Card will not be converted.

During program conversion, "**" is displayed at the bottom right of the display. Approximately 600 bytes are required for work area to convert a program.
If a converted line is too long, an error will occur.
If a password has been set, executing the BASIC command results in an error.

BEEP

P
D

FORMAT: BEEP number [,[tone][,duration]]

Abbreviation: B.
See Also:

PURPOSE:
Generates beeps of the specified tone and duration through the Organizer's internal speaker.

REMARKS:
Number specifies the number of times the beep will sound. Specify a positive value or expression up to 65535.

Tone specifies the frequency of the beep in the range of 255 to 1. As the value of the tone parameter is increased, the frequency is reduced.

Duration specifies the duration of the beep in the range of 1 to 1048575. The beep duration setting varies with the tone parameter. A given duration value will give a relatively longer beep at lower frequencies.

If the duration is omitted, a default value of 19200 is set.
If the tone is omitted, a default value of 12 is set.
If the duration and tone are omitted, the frequency of the beep is set to approximately 4 KHz.

Press the [ON] key to stop a beep.

EXAMPLE:

```
10: FOR I = 1 TO 3
20: FOR J = 5 TO 25 STEP 5
30: BEEP I,J,150
40: NEXT J
50: NEXT I
60: END
```

[10] The outer loop changes the number of beeps from 1 to 3.
[20] The inner loop changes the tone.
[30] The BEEP statement is executed 15 times.

BREAK ON/OFF

P
D

FORMAT: 1. BREAK ON
2. BREAK OFF

Abbreviation:
See Also:

PURPOSE:
Enables/disables program execution break using the [ON] key.

REMARKS:
Format 1 selects the Break On mode, in which pressing the [ON] key breaks program execution.
Format 2 selects the Break Off mode, in which the [ON] key is unable to break program execution.

The Break On mode is set after power on, execution of the RUN, ARUN, END, or STOP command, or an error.

CHAIN

P

FORMAT: 1. CHAIN

2. CHAIN "d: filename"[, {line number / *label}]

d: E, PACOM, CAS, COM

Abbreviation: CHA.
See Also: LOAD, MERGE

PURPOSE:
Loads and starts, from within one program, another program that has been stored on the specified device.

REMARKS:
To use CHAIN, the specified program must be present on the specified device. The currently running program is cleared from memory at the point where a CHAIN command is encountered and the specified program is loaded and started. Entering the CHAIN statement at the end of each program will continue to load programs automatically. An error will occur if the program becomes too large for the program area as a result of programs loaded by the CHAIN command.

Format 1 is for use only with tape and loads the first stored program from tape and starts it from the lowest line number in the program. The effect is the same as having entered LOAD "CAS:" and RUN in the RUN mode.

Format 2 searches the specified device for the program indicated by "filename", loads it, and starts it at the specified line number or *label, or the lowest line number if omitted.

Note:
A file is automatically opened when executing the CHAIN command. If the device name is specified as PACOM, CAS, or COM, files for that device must be closed.

EXAMPLE:
CHAIN "E:PRO1", 100

Searches the RAM disk for a program named PRO1, loads it and begins execution with line number 100.

CHR$

P
D

FORMAT: CHR$ expression

Abbreviation: CH.
See Also: ASC

PURPOSE:
Returns the character that corresponds to the numeric character code of the parameter.

REMARKS:
See Appendix B for a chart of character codes and their relationship to characters, e.g., CHR$ 65 is "A".

A hexadecimal number can be specified with "&H" in front of the character code (eg. A$ = CHR$ &H5A)

A value greater than 255 generates an error.

CLEAR

P
D

FORMAT: CLEAR [variable 1, variable 2, ..., variable n]

Abbreviation: CL.
See Also: DIM, ERASE

PURPOSE:
Erases variables that have been used in the program and resets all preallocated variables to zero or null.

REMARKS:
CLEAR recovers memory space used to store simple numeric variables, and array variables secured using the DIM statement. Also use CLEAR at the beginning of a program to clear space occupied by variables from previously run programs if several programs are in memory.

Do not use the CLEAR command within a FOR...NEXT loop.

Use the ERASE command to clear specific array variables.

EXAMPLE:

```
10: A = 5: DIM C(5)
20: CLEAR
```

[20] Frees up the spaces assigned to C( ) and resets A to zero.

CLOSE

P
D

FORMAT: CLOSE [# file number, # file number, ...]

Abbreviation: CLOS.
See Also: OPEN

PURPOSE:
Closes a file or files on the currently accessed device.

REMARKS:
This command closes files with the specified file numbers. If no file number is specified, all files are closed. The file numbers are then released for use with other files.

All files are closed in the following cases:

- An END or RUN command is executed.
- The power is turned off.
- The Card is changed to the STAT or AER mode.
- A program is written or read (by the LOAD or MERGE command).

EXAMPLE:

```
CLOSE #2, #5, #21
```

Closes files #2, #5, and #21.

CLS

P
D

FORMAT: CLS

Abbreviation:
See Also: LOCATE

PURPOSE:
Clears the display.

REMARKS:
Clears the display and resets the display start position to (0,0).

CONT

D

FORMAT: CONT [ENTER]

Abbreviation: C.
See Also: STOP

PURPOSE:
Continues a program that has been temporarily halted.

REMARKS:
Valid only as direct input in the RUN mode.

Enter CONT to continue running a program that has been stopped with the STOP command. Enter CONT at the prompt to continue a program that has been halted using the [ON] key. CONT can also be used to continue a program interrupted by PRINT or GPRINT. (See WAIT command.)

COPY

D

FORMAT: COPY "d1:filename 1" TO "[d2:]filename 2" [,A] [ENTER]
d1: E, PACOM, CAS, COM
d2: E, PACOM, CAS, COM

Abbreviation: COP.
See Also:

PURPOSE:
Copies the contents of a file from one storage device to another storage device.

REMARKS:
Copies the contents of the file with filename 1 within device d1 to another file with filename 2 within device d2. The following shows a cross reference for device names usable for devices d1 and d2:

		d2			
		E	PACOM	CAS	COM
d1	E	○	○	○	○
	PACOM	○	×	×	○
	CAS	○	×	×	○
	COM	○	○	○	×

○: Usable ×: Not usable

An error occurs if the same name is given for devices d1 and d2 and for filenames 1 and 2.

If ",A" is specified, the system regards filename 1 as an ASCII file and code "1AH" as an EOF (end of file) code.

If PACOM, CAS, or COM is given for d2, the destination file becomes an ASCII file. If PACOM, CAS, or COM is given for d1, the source file must be an ASCII file. If d2 is omitted, the same device as d1 is assumed.

An error occurs if filename 1 is not given. If filename 2 already exists, it will be overwritten.

The extension can be omitted if it is blank.

Notes:
1. An error occurs if device d2 or the file with filename 2 is write-protected (see SET command).
2. The wildcard cannot be used for device name PACOM, CAS, or COM.

EXAMPLE:
Transfer file "TEST" as file "SAMPLE" using device name "PACOM". Before execution, prepare the RAM disk E on the receiver.

Sender
COPY "E:TEST" TO "PACOM: SAMPLE" [ENTER]

Receiver
COPY "PACOM: SAMPLE" TO "E: SAMPLE" [ENTER]

File "TEST", which is stored in the sender's RAM disk E, is transferred to the receiver's RAM disk E in the Organizer's IC card, under the file named "SAMPLE".

DATA

P

FORMAT: DATA list of values

Abbreviation: DA.
See Also: READ, RESTORE

PURPOSE:
Provides values for use by READ.

REMARKS:
When assigning initial values to an array, it is convenient to list the values in a DATA statement and use a READ statement in a FOR...NEXT loop to load the values into the array. When the first READ is executed, the first value in the first DATA statement is returned. Succeeding READs use succeeding values in the sequential order in which they appear in the program, regardless of how many values are listed in each DATA statement or how many DATA statements are used.

A DATA statement may contain any numeric or string values, separated by commas. Enclose string values in quotes. Spaces at the beginning or end of a string should be included in the quotes.

DATA statements have no effect if encountered in the course of regular execution of the program, so they can be inserted wherever appropriate. Many programmers include them after the READ that uses them. If desired, the values in a DATA statement can be read a second time using the RESTORE statement.

DATE$

P
D

FORMAT: DATE$

Abbreviation:
See Also: TIME$

PURPOSE:
Recalls the date currently set in the Organizer.

REMARKS:
The DATE$ command recalls the date currently set in the Organizer in the form of 10 character string data. The date is displayed in the order of month, date, and year irrespective of the format specified by the Organizer. Month and date are given by two digits each, while the year is given by 4 digits. Hyphens are used to separate the digits.
For details on setting the date, see the description under "Setting the standard clock" in the Organizer manual.

EXAMPLE:
```
10: PRINT DATE$
20: A$ = DATE$
30: PRINT A$
```


DEFDBL

P
D

FORMAT: 1. DEFDBL character range
2. DEFDBL

Abbreviation: DEF.
See Also: DEFSNG

PURPOSE:
Defines variable(s) with single-character names as having double precision accuracy or specifies "double-precision" mode calculations.

REMARKS:
In format 1, the variables in the "character range" are designated as double precision. "character range" can be specified as follows:

- DEFDBL C–F
  where variables, C, D, E and F are designated as double-precision, or
- DEFDBL E,F,Z,H–J
  where variables E, F, Z, H, I and J are designated as double-precision.

Variable names followed by the single-precision type declaration character (!) are given type priority over variable names declared by the DEFDBL statement. For example, variables E and F in the statement

```
DEFDBL E,F
```

will be treated as double-precision variables, but E! and F! will be treated as single-precision variables. Variables not declared as double- or single-precision will be treated as single-precision variables.

In format 2, all subsequent calculations are carried out to double precision. The DBL mark is shown on the screen in this mode. The double-precision mode is canceled by the following:

- The DEFSNG statement is executed.
- The power is turned off.
- The RUN or NEW command is executed.
- A program is loaded (except by the CHAIN command).

When using the DIM statement to establish the number of elements allowed in a numeric array, the DEFDBL statement must be used first if those elements are to be treated as double-precision variables:

```
DEFDBL A
DIM A(3,2)
```

In the example below, the elements will be treated as single-precision variables:

```
DIM A(3,2)
DEFDBL A
```


DEFSNG

P
D

FORMAT: 1. DEFSNG character range
2. DEFSNG

Abbreviation: DEFS.
See Also: DEFDBL

PURPOSE:
Defines variable(s) with single-character names as having single precision accuracy or cancels double-precision mode specified by DEFDBL.

REMARKS:
In format 1, the variables in the "character range" are designated as single-precision. "Character range" can be specified as follows:

- DEFSNG C–F
  where variables C, D, E and F are designated as single-precision, or
- DEFSNG E,F,Z,H–J
  where variables E, F, Z, H, I and J are designated as single-precision.

Variable names followed by double-precision type declaration characters (#) are given type priority over variable names declared by the DEFSNG statement. For example, variables E and F in the statement

```
DEFSNG E,F
```

will be treated as single-precision variables, but variables E# and F# as double-precision variables. Variables not declared as double- or single-precision are treated as single-precision variables.

In format 2, all subsequent calculations are carried out with single precision. The DBL mark on the screen is canceled in this mode.

DEGREE

P
D

FORMAT: DEGREE

Abbreviation: DE.
See Also: GRAD, RADIAN

PURPOSE:
Changes the form of angular values to decimal degrees.

REMARKS:
There are three forms for representing angular values — decimal degrees, radians and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The DEGREE function changes the form of all angular values to decimal degree form until GRAD or RADIAN is used. The DMS and DEG functions can be used to convert angles from decimal degree form to degree, minute, second form and vice versa.

DELETE

D

FORMAT:
1. DELETE line number [ENTER]
2. DELETE line number {−, ,} [ENTER]
3. DELETE line number {−, ,} line number [ENTER]
4. DELETE {−, ,} line number [ENTER]

Abbreviation: DEL.
See Also: NEW, PASS

PURPOSE:
Deletes specified program lines in memory.

REMARKS:
Valid only as direct input in the PRO mode.

Format 1 deletes only the specified program line. Format 2 deletes program lines from the line number specified up to the highest program line in memory. Format 3 deletes all program lines between the first specified line number (lower value) and the second specified line number (higher value). Format 4 deletes program lines from the lowest line number in memory up to the specified line number.

Using DELETE in the RUN mode generates an error. If a password has been used, the command is not executed and the prompt is displayed. Only the digits 0-9 can be in the line numbers. Specifying a line that does not exist generates an error. Specifying a start line number that is greater than the end line number also generates an error.
If the first and second line numbers are omitted, an error will occur.

To delete the whole program, use the NEW command.

EXAMPLE:

```
DELETE 150*
DELETE 200,**
DELETE 50-150***
DELETE ,35****
```

* Deletes line 150 only.
** Deletes from line 200 to the highest line number.
*** Deletes all lines between and including line 50 and line 150.
**** Deletes from the lowest line number up to line 35.

DIM

P
D

FORMAT: DIM variable name 1 (size 1[, size 2, size 3, ...])
[, variable name 2 (size 1 [, size 2, size 3, ...])]

Abbreviation: D.
See Also: ERASE, CLEAR, RUN

PURPOSE:
Reserves space for numeric and string array variables.

REMARKS:
DIM must be used to reserve space for an array variable. The size of an array is the number of elements in that array.

A variable name consists of a letter and up to 39 alphanumeric characters. For string variables, "$" is attached to the end of the variable name. Numeric variables may be either single-precision or double-precision variables. Even though two variables may have the same name (one single-, one double-precision), they will be recognized as two different variables.

Size 1, size 2 are called the "subscripts", and specify the number of elements in the nth dimension of the array. An array with one subscript is called a one-dimensional array, with two subscripts, it is called a two-dimensional array and an array with n subscripts is called an n-dimensional array.

Example:
DIM B(3):
one-dimensional array B( ) reserves 4 array elements B(0), B(1), B(2) and B(3)
DIM XA$(2,3):
two-dimensional string array XA$( ) reserves 12 array elements XA$(0,0), XA$(0,1), ..., XA$(2,2), XA$(2,3)

Integers 0–65534 can be used as subscripts, but an error may occur if a variable with the specified size cannot be reserved because of limits in the memory size and conditions of use.

If the subscripts include a decimal point, only the integer part will be recognized (and the fractional part will be ignored).

Example:
X(2.3) recognized as X(2)
Y(0.25) recognized as Y(0)

The subscript may be declared by a numeric variable or expression:

```
10: INPUT A,B
20: DIM X(A), Y#(B-1,A*B)
```

More than one array can be declared using one DIM statement.

Example:

```
DIM V(5), K$(4,3), XB$(5)
```

If an array has been defined, it cannot be defined again. For example, both DIM X(5) and DIM X(3,4) cannot be defined since the variable names are the same. However, DIM X!(5) and DIM X#(3,4) can be defined since one is a single-precision variable and the other is a double-precision variable.

When a program is executed using the RUN or ARUN command, allocated array variables are cleared, but they are not cleared using the GOTO statement. Thus, when a program is to be executed again using the GOTO statement, an error will occur if a DIM statement attempts to reallocate space for an existing array variable. Either GOTO a line following the DIM statement, or add an ERASE statement and redefine the array.

Example:

```
50: ERASE X: DIM X(3,4)
```

Numerical array and string array variables are recognized as different arrays; thus, the arrays Z( ) and Z$( ) can be defined simultaneously.

The DIM statement cannot be used within a FOR...NEXT loop.

DSKF

P
D

FORMAT: 1. DSKF "E:"
2. DSKF (3)

Abbreviation: DS.
See Also:

PURPOSE:
Returns the amount of free space on a RAM disk E.

REMARKS:
DSKF returns the size of the free disk area in bytes.

The RAM disk is used in blocks of 256 bytes.

To store a program formatted in intermediate code, an additional 20 bytes are required for control area. Thus, a 500 byte program will require 768 bytes on RAM disk E.

EXAMPLE:
DSKF(3)

Returns the free space on the RAM disk E.

END

P

FORMAT: END

Abbreviation: E.
See Also:

PURPOSE:
Signals the end of a program.

REMARKS:
The program will be terminated when the END statement is executed. Statements after the END statement in the same line cannot be executed. All opened files are closed.

EOF

P
D

FORMAT: EOF (file number)

Abbreviation: EO.
See Also:

PURPOSE:
Determines if the end of a sequential file has been reached.

REMARKS:
The EOF function checks if all data in a sequential file (with the specified file number) has been read.

If all data has been read, EOF returns –1 (true) as its function value. If not, EOF returns 0 (false). For the device name COM, EOF returns –1 (true) if the 20-character buffer is empty and 0 (false) if not.

An error occurs if a file with the specified number has not been opened for input.

EXAMPLE:
```
10: OPEN "E:A" FOR OUTPUT AS #2
20: PRINT #2, 123,456,789
30: CLOSE
40: OPEN "E:A" FOR INPUT AS #2
50: INPUT #2,A,B
60: X = EOF (2)
70: INPUT #2,C
80: Y = EOF (2)
90: CLOSE:END
```

[60] Not all data has been read in this line. X = 0.
[80] All data has been read. Y = –1.

ERASE

P
D

FORMAT: ERASE array 1 [, array 2, ... array n]

Abbreviation: ER.
See Also: CLEAR, DIM

PURPOSE:
Erases specified arrays.

REMARKS:
Array elements cannot be erased individually; the whole array is cleared and its memory area is freed. To re-define an array size, first ERASE it and then re-specify it in a DIM statement.

Double-precision array variables can be specified. The contents of double-precision variables specified using the DEFDBL command are erased, but the DEFDBL mode is not.

Do not use the ERASE command within a FOR...NEXT loop.

EXAMPLE:
```
 10: DIM AA(10)
   :
   :
200: ERASE AA
```

ERL

P
D

FORMAT: ERL

Abbreviation:
See Also: ERN, ON ERROR GOTO

PURPOSE:
Returns the line number at which an error occurred during program execution.

REMARKS:
The ERL function is used with the ERN function and the ON ERROR GOTO statement in error processing routines to control program flow when an error occurs. A line number is only set in ERL if the error occurred during program execution.

ERL will be cleared when
- a RUN statement is executed.
- the power is turned off.
- a program is loaded.

ERN

P
D

FORMAT: ERN

Abbreviation:
See Also: ERL, ON ERROR GOTO

PURPOSE:
Returns the error code number of the execution error.

REMARKS:
The ERN function is used with the ERL function and the ON ERROR GOTO statement in error processing routines to control program flow when an error occurs.

See Appendix A for a list of error messages.

ERN will be cleared when
- a RUN statement is executed.
- the power is turned off.
- a program is loaded.

EXAMPLE:
```
 10: ON ERROR GOTO 100:WAIT
 20: FOR N = 1 TO 20
 30: READ A
 40: PRINT A
 50: NEXT N
 60: END
100: IF ERL = 30 AND ERN = 53 THEN PRINT "YOU HAVE NO DATA"
```

EVAL

P
D

FORMAT: EVAL {"character string" / string variable}

Abbreviation: EV.
See Also:

PURPOSE:
Calculates the string as an expression.

REMARKS:
EVAL command calculates the character string enclosed in the double quotes by regarding it as an expression. It can also calculate the character string assigned to a string variable or string array variable.

EXAMPLE:
```
10: PRINT EVAL "15+5*2"
20: A$="20*10":C=EVAL A$
30: PRINT C
40: A=10:B=20
50: PRINT EVAL "SIN A+COS B"
```


FILES

D

FORMAT: FILES ["[E:][filename]"] [ENTER]

Abbreviation: FI.
See Also: LFILES, SET

PURPOSE:
Displays names and attributes of specified file(s) on RAM disk.

REMARKS:
FILES displays the filename, the filename extension (.BAS or other extension), and "P" (write-protection) attribute (see SET command).

If no device name is specified, the last device name used will be assumed. If no filename is specified, all files on the specified device will be displayed. If neither device nor filename is specified, all files on the last device used will be displayed. To display a series of filenames, use an ambiguous filename. (See below.) To display a single filename, specify only that filename and its extension.

The number of bytes used is also displayed. (See the SAVE and SET commands.)

A maximum of three filenames will be displayed at one time, and an ⇨ mark will appear to the left of the filenames. Scroll through the files by pressing the [▲] and [▼] keys to move the ⇨ mark up or down, respectively.
Press [SHIFT] [▲] to move to the bottom of the previous page, and [SHIFT] [▼] to move to the top of the next page. Pressing [L] [ENTER] or [L] [Y] when the ⇨ mark is next to a desired filename allows the file to be loaded into memory. Pressing [K] [ENTER] or [K] [Y] kills (deletes) the file where the ⇨ mark is located. Once a file has been deleted, it cannot be recovered, so use this option with care. To avoid loading or killing a file, press any key other than [Y] or [ENTER] when the OK? prompt appears.

Specify an ambiguous filename to list directory information on groups of files with common name forms. There are two wildcard characters available for this purpose. The asterisk "*" stands for any number of characters (including none) in the filename. The question mark "?" stands for a single character in a filename. The following are examples of the use of wildcard characters:

File specification	Files listed
TEST?	TEST, TESTS, TEST1, TESTA
T??T	TEST, TEXT, TORT, TXYT
S?MPLE	SIMPLE, SAMPLE, S2MPLE
A?????	ABCDEF, APPEND, A12345
R*	RATES, R1, RETURNS, RAND2, R

If the [C•CE] or [ON] key is pressed, or if the last filename is being displayed and the [ENTER] key is pressed, the entry prompt ">" is displayed, and the Card waits for the next command.

FILES has no effect if the specified files do not exist on the specified device.

EXAMPLE:

FILES"E:DATA"

Displays information about the file DATA on RAM disk E.

FILES"E:???1"

Lists all files on RAM disk E that have 4-letter names ending in 1.

FOR...NEXT

P

FORMAT:

FOR { fixed numeric variable / single–precision simple numeric variable } = expression 1 TO expression 2 [STEP expression 3]

⋮

NEXT [{ fixed numeric variable / single–precision simple numeric variable }]

Abbreviation: F. N. STE.

See Also:

PURPOSE:

In combination with NEXT, repeats a series of operations a specified number of times.

REMARKS:

FOR and NEXT are used in pairs to enclose a group of statements that are to be repeated. If the variable following NEXT is omitted, the variable following FOR is assumed. The first time this group of statements is executed the loop variable (the variable named immediately following FOR) is assigned its initial value (expression 1).

When execution reaches the NEXT statement, the loop variable is increased by the STEP value (expression 3) and then this value is tested against the final value (expression 2). If the value of the loop variable is less than or equal to the final value, the enclosed group of statements is executed again, starting with the statement following FOR. If expression 3 for step size is omitted, the increment becomes 1. If the value of the loop variable is greater than the final value, execution continues with the statement that immediately follows NEXT. Because the comparison is made at the end, the statements within a FOR...NEXT pair are always executed at least once.

When the increment is zero, FOR...NEXT will continue in an infinite loop.

The loop variable may be used within the group of statements, for example as an index to an array, but care should be taken in changing the value of the loop variable.

Write programs so that the program flow does not jump out of a FOR...NEXT loop before the counter reaches the final value. To exit a loop before it has been repeated the specified number of times, set the loop variable higher than the final value.

The group of statements enclosed by a FOR...NEXT pair can include another pair of FOR...NEXT statements that use a different loop variable as long as the enclosed pair is completely enclosed; i.e., if a FOR statement is included in the group, the matching NEXT must also be included. FOR...NEXT pairs may be "nested" up to six levels deep. Illegally jumping out of an inner loop will generate an error, a nesting error. See Appendix E.

An error results if a double-precision variable is specified as the numeric variable. Double-precision initial values, final values, and increments are treated as single-precision values.

Do not use the CLEAR, DIM, or ERASE command within a FOR...NEXT loop.

FRE

P
D

FORMAT: 1. FRE 0
2. FRE 1

Abbreviation: FR.
See Also:

PURPOSE:
Returns the free space available in the program and data area in bytes.

REMARKS:
FRE returns the byte count of the free space (not occupied by program, array variables, or simple variables) in the program and data area of memory.

To speed up execution, a certain fixed number of bytes are reserved for each string variable even though a shorter string may be assigned to the variable. Thus, the size of the free space is not affected by the lengths of strings assigned to string variables. It is, however, possible to eliminate idle space in each variable to increase free space.

Format 1 returns the free space by eliminating idle space in each string variable, so its execution may take a little more time.
Format 2 returns the free space without eliminating idle space in string variables. It is useful for determining the approximate amount of free space.

The value of free space returned by format 1 may not match that returned by format 2.

GCURSOR

P

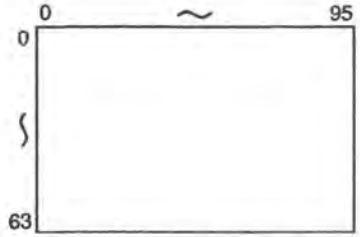
FORMAT: GCURSOR (expression 1, expression 2)

Abbreviation: GC.
See Also: LOCATE, GPRINT

PURPOSE:
Specifies the starting point of dot graphics display.

REMARKS:
GCURSOR specifies the display starting point for the dot pattern to be displayed by the GPRINT command.

The screen consists of 96 columns and 64 rows of dots, which can be addressed by column numbers 0 to 95 and row numbers 0 to 63. Any dot on the screen can therefore be addressed as a starting point by specifying the column number with expression 1 and the row number with expression 2.



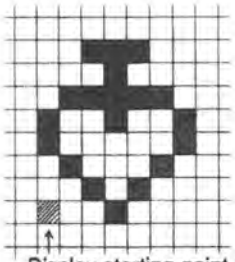
The values of expressions 1 and 2 may range from –32768 to 32767. It should be noted, however, that if the value of expression 1 is outside the range of 0 to 95 or that of expression 2 is outside the range of 0 to 63, the display starting point will become a virtual point which is outside the screen boundaries.

Location (0,7) is automatically assumed as the display starting point if the RUN command is executed or [SHIFT] [C•CE] are pressed.
If a program is started with GOTO, only the row number specified by this command is maintained, with the column number automatically reset to zero.

EXAMPLE:

```
 5: CLS:WAIT
10: GCURSOR (40,30)
20: GPRINT "1824A2F1A22418"
```

This program prints the following dot pattern near the center of the screen (the display starting point indicated by the shaded box is not displayed):



Display starting point (40,30)

GOSUB...RETURN

P

FORMAT: GOSUB {line number / *label}

RETURN {line number / *label}

Abbreviation: GOS. RE.
See Also: GOTO, ON...GOSUB

PURPOSE:
Diverts program execution to a BASIC subroutine.

REMARKS:
When you wish to execute the same group of statements several times in the course of a program, it is convenient to use the BASIC capability for subroutines using GOSUB and RETURN.

The group of statements is included in the program at some location where they are not reached in the normal sequence of execution. A common location is following the END statement that marks the end of the main program.

At each location in the main body of the program where a subroutine is to be executed, include a GOSUB statement with a line number or *label that indicates the starting line number of the subroutine. The last line of each subroutine must be a RETURN.
When GOSUB is executed, the Card transfers control to the indicated line number or *label and processes the statements until a RETURN is reached. Control is then transferred back to the statement following the GOSUB. If a line number or *label follows RETURN, control will return to the line number or *label.

Subroutines may be "nested" up to 36 levels deep. (See Appendix E.)

Since there is an ON...GOSUB structure for choosing different subroutines at given locations in the program, the expression in a GOSUB statement usually consists of just the desired line number or *label.

GOTO

P
D

FORMAT: GOTO {line number / *label}

Abbreviation: G.
See Also: GOSUB, ON...GOTO, RUN

PURPOSE:
Transfers program control to a specified line number or *label.

REMARKS:
GOTO transfers control from one location in a BASIC program to another location. Unlike GOSUB, GOTO does not "remember" the location from which the transfer occurred.

Usually, a program is executed sequentially from the smallest line number. However, execution can be transferred to the line with the given line number or *label. Program execution can be started from the specified line by specifying a GOTO statement as direct input in the RUN mode. The transfer destination is specified by entering the line number or *label after the GOTO command.

Example:

GOTO 40	Jumps to line 40
GOTO *AB	Jumps to the line with label *AB

If the specified line number or *label does not exist, an error occurs. If two or more identical *labels are included in a program, program execution transfers to the line with the lower line number.

EXAMPLE:

```
10: INPUT A$
20: IF A$ = "Y" GOTO 50
30: PRINT "NO"
40: GOTO 60
50: PRINT "YES"
60: END
```

This program prints "YES" if a "Y" is entered and prints "NO" if anything else is entered.

GPRINT

P
D

FORMAT: 1. GPRINT "string"
2. GPRINT expression [; expression; expression; ...]
3. GPRINT

Abbreviation: GP.
See Also: GCURSOR, PRINT

PURPOSE:
Displays the specified dot pattern.

REMARKS:
The GPRINT command displays the specified dot pattern. Each column of bit image data is represented by 8 vertical dots.

In format 1, the 8-dot pattern is divided into an upper group of 4 dots and a lower group of 4 dots. Each group of dots is then represented by a hexadecimal number. The string is enclosed by " ".

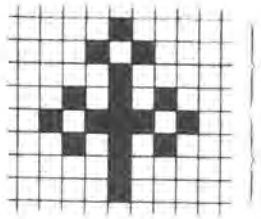
Hex. Digit	Dot Pattern	Hex. Digit	Dot Pattern	Hex. Digit	Dot Pattern	Hex. Digit	Dot Pattern
0		4		8		C	
1		5		9		D	
2		6		A		E	
3		7		B		F	

GPRINT "○○ ○○ ○○ ○○"

Each pair of hexadecimal numbers represents one vertical dot pattern (of 8 dots). The first number represents the upper 4 dots, the second number represents the lower 4 dots.

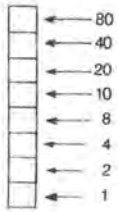
Example:

GPRINT "081448BF481408"
- Represents the upper 4-dot pattern
- Represents the lower 4-dot pattern

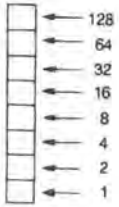


Using format 2, GPRINT 8;20;72;191;72;20;8 produces the same dot pattern.
Specify a semicolon (;) at the end of the string to automatically move the cursor to the next position.

In format 2, the vertical 8-dot pattern is specified using a hexadecimal or decimal value. A "weight" is assigned to each dot in the vertical 8-dot pattern, as shown below.



Weight of each dot (hexadecimal)



Weight of each dot (decimal)

Specify the dot pattern with a numeric value equal to the sum of the "weights" of the dots to be displayed. The value may be any number between 0 and 255.

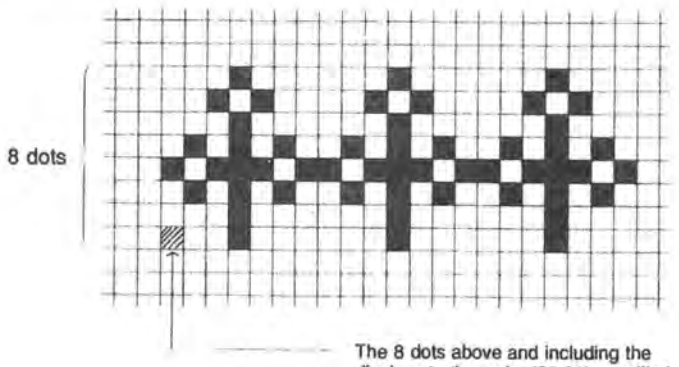
In format 3, the previously specified and displayed pattern is displayed without modification.

The dot pattern specified in the GPRINT command will be displayed beginning with the 8 dots on and above the display starting point specified by the GCURSOR command.

Note:
If the GPRINT command is terminated with " ; ", a subsequent GPRINT command takes effect from the next cursor position (the " ; " concatenates the commands). If the GPRINT command is terminated with " : " or by pressing [ENTER], the horizontal position returns to 0.

EXAMPLE:

```
10: AA$ = "081448BF481408"
20: GCURSOR (30,30)
30: GPRINT AA$;AA$;AA$
```



The 8 dots above and including the display starting point (30,30) specified by the GCURSOR command are used to display the first value given in GPRINT.

GRAD

P
D

FORMAT: GRAD

Abbreviation: GR.
See Also: DEGREE, RADIAN

PURPOSE:
Changes the form of angular values to gradient form.

REMARKS:
There are three forms for representing angular values: decimal degrees, radians, and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results form the ASN, ACS, and ATN functions.

The GRAD function changes the form for all angular values to gradient form until DEGREE or RADIAN is used. Gradient form represents angular measurement in terms of percent gradient, i.e., a 45° angle is a 50 percent gradient.

HEX$

P
D

FORMAT: HEX$ expression

Abbreviation: H.
See Also:

PURPOSE:
Converts a decimal number into its hexadecimal character string equivalent.

REMARKS:
The value of the expression must be in the range of –9999999999 to 9999999999. The resulting hexadecimal character string will be up to 10 digits long.

EXAMPLE:

```
C$ = HEX$12 + HEX$15
```

C$ is assigned the character string "CF".

IF...THEN...ELSE

P

FORMAT:

IF condition THEN { line number / *label / statement } [ELSE { line number / *label / statement }]

Abbreviation: IF T. EL.
See Also: AND, OR, NOT, XOR

PURPOSE:
Conditionally executes a statement at the time the program is run.

REMARKS:
When the condition of the IF statement is true, the statement following THEN is executed; if it is false, the statement following ELSE is executed. When the ELSE statement is omitted and the condition is false, the statement following THEN is skipped.

If THEN or ELSE is followed by a GOTO statement, either THEN or GOTO may be omitted (ELSE statement must be included).

Example 1:

```
IF A<5 THEN C=A*B:GOTO 50
```

If A is smaller than 5, then assign the product, A*B, to C and go to line 50.

Example 2:

```
IF B=C+1 GOTO 60 ELSE 100
or
IF B=C+1 THEN 60 ELSE 100
```

If B equals C+1, then go to line 60; otherwise go to line 100.

The condition (e.g. A<5) of the IF statement can be any relational expression as listed below:

Relational expression	Description
○○ = ××	Equal to
○○ > ××	Greater than
○○ >= ××	No less than
○○ < ××	Less than
○○ <= ××	No greater than
○○ <> ××	Not equal to

Note: ○○ and ×× represent expressions (5*4, A, 8, etc.).

More than one relational expression can be linked with the logical operators "*" or "+".
For example:

```
IF (A>5)*(B>1) THEN ....
```

If A is greater than 5 and B is greater than 1, the statement following THEN is executed. Logical operator "AND" may be used in place of "*".

```
IF (A>5)+(B>1) THEN ....
```

If A is greater than 5 or B is greater than 1, the statement following THEN is executed. Logical operator "OR" may be used in place of "+".

Using Character Strings in Relational Expressions
The magnitudes of character strings can be compared when used in a relational expression of an IF...THEN...ELSE statement. The magnitudes of character codes are compared. For example, characters A, B, and C have codes 65, 66, and 67, respectively. So A is smaller than B, and B is smaller than C.

EXAMPLE:

```
10: INPUT"CONTINUE?";A$
20: IF A$="YES" THEN 10
30: IF A$="NO" GOTO 60
40: PRINT "YES OR NO, PLEASE"
50: GOTO 10
60:
```

Note:
Whenever a variable name is to be followed by a statement, be sure to insert a space between them, for example:

```
100 IF A=B␣THEN 200
          ↑
          A space is needed.
```

Pay special attention to this when you use the IF, FOR, ON...GOTO, or ON...GOSUB command.

INIT

D

FORMAT: 1. INIT "E:?K" [ENTER]
2. INIT "E:" [ENTER]

Abbreviation: INI.
See Also:

PURPOSE:
Initializes the RAM disk E.

REMARKS:
Format 1 specifies the data file area on RAM disk E, which allocates a section of the memory in the Card to store programs and data as if they were stored in external storage.

Specify the size (?) of the data file area in kilobytes. The allowable size for RAM disk E is 2 to the size of the free area.

When this command is executed, the system asks if you are sure you want to delete the existing contents of the RAM disk. If yes, press [Y] . If no, press [N] . When [N] is pressed, the contents of the RAM disk file area are not affected.

When the size of the current disk area is changed, its contents will be cleared.

If "0K" is specified in format 1, the command clears the file area and cancels area definition.

In format 2, the current RAM disk area is initialized.

INKEY$

P

FORMAT: INKEY$

Abbreviation: INK.
See Also:

PURPOSE:
Gives the specified variable the value of the key pressed while the INKEY$ function is executed.

REMARKS:
INKEY$ is used to respond to the pressing of individual keys without waiting for the [ENTER] key to end the entry.

See the following table for a list of applicable keys and the characters that are returned.

The INKEY$ command reads the [SHIFT] or [CAPS] key if it is pressed. Thus, it is unable to read the function or symbol key that is pressed following either of these keys.

EXAMPLE:

```
 5: CLS: WAIT 60
10: IF INKEY$< > " " THEN 10
15: A$=INKEY$
20: B=ASC A$
30: IF B=0 THEN 10
40: IF B ...
```

Lines 40 and beyond contain tests for the key and the actions to be taken (for example: 60: PRINT B).

INKEY$ Character Code Table

H / L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SPC	0		P			MODE CHECK	PLAY BACK						
1	SHIFT			1	A	Q			BASIC	ANS			CARD			
2				2	B	R			STAT	(			HOME			
3		INS		3	C	S			AER	)						
4				4	D	T			2ndF							
5		CAPS	%	5	E	U			SIN				USER DIC			
6				6	F	V			COS				SMBL		<	
7				7	G	W			TAN						>	
8	BS			8	H	X			EXP							
9	↵			9	I	Y			LN							
A			×		J	Z			LOG							
B			+		K				Y^x			CALENDAR				
C	C•CE	▲			L				$\sqrt{}$			SCHEDULE				
D	ENTER	▼	-	=	M				x^2			TEL	R·CM			
E		◀	·		N				$1/x$			MEMO	M−			
F		▶	+		O			DEL	→DEG			CALC	M+			

- The [ON] key functions as a Break key.
- Codes &H80-&H93 are transparent guide key codes.

INPUT

P

FORMAT: 1. INPUT variable [,variable]
2. INPUT "prompt string", variable [[,"prompt string"], variable]
3. INPUT "prompt string"; variable [[,"prompt string"]; variable]

Abbreviation: I.
See Also: INPUT#, INKEY$, READ, LOCATE

PURPOSE:
Allows entry of one or more values from the keyboard.

REMARKS:
When you want to enter different values each time a program is run, use INPUT to enter these values from the keyboard.

Format 1 displays symbol "?" to prompt data entry. If data is entered and the [ENTER] key is pressed at this prompt, the system assigns the data to the variable and resumes program execution.
If more than one variable is specified, the data prompt is repeated the corresponding number of times.

During data prompt, format 2 displays the character string enclosed by double quotes (" ") as entry guidance. The guidance disappears when data is entered.

Format 3 also displays entry guidance during data prompt, but the entered data appears following the entry guidance, which does not disappear.

Formats 1, 2, and 3 may be concurrently used in one INPUT statement:
INPUT "A=";A,B,"C=?",C

The type of the variables given in the INPUT statement must match the type of input data. Assign string data to string variables, and numerical data to numerical variables. If "ABC" is entered in response to a numerical entry prompt, the values assigned to variable ABC is assigned. This allows you to enter such values as SIN30.

If the start position is specified using the LOCATE statement before excecuting the INPUT statement, the prompt string or ? will be displayed at the specified location.

EXAMPLE:

```
10: INPUT A
20: INPUT "A=";A
30: INPUT "A=",A
40: INPUT "X=?";X,"Y=?";Y
```

[10] Puts a question mark at the left margin.
[20] Displays "A=" and waits for data to be entered.
[30] Displays "A=". When data is entered, "A=" disappears and the data is displayed starting at the left margin.
[40] Displays "X=?" and waits for the first entry. After [ENTER] is pressed, "Y=?" is displayed at the left margin.

INPUT$

P
D

FORMAT: 1. INPUT$ (character count)
2. INPUT$ (character count, # file number)

Abbreviation: I.$
See Also: INPUT#, OPEN, PRINT#

PURPOSE:
Allows input of a character string with the specified number of characters from the keyboard or a file.

REMARKS:
Up to 255 characters can be specified as character count.

Format 1 is used to enter a character string with the given number of characters from the keyboard. Execution automatically proceeds to the next statement after a string has been entered.

The [←┘] or [ENTER] key is also counted as one character.
A symbol (e.g. [SMBL] [▼] [▼] 1; four keystrokes) is counted as one character.
When entering a string from the keyboard, the OPEN command need not be executed.

Format 2 reads a character string with the given number of characters from the file with the given file number. An error occurs if the specified file has not been opened.
The INPUT$ command is valid only for a file which has been opened in the INPUT mode.

Numerals are treated as characters when read.

Since the INPUT$ command reads the specified number of characters, the data stored in the file must have the proper format readable by the INPUT$ command.

EXAMPLE:

```
100: A$=INPUT$ (5, #5)
110: AB$=INPUT$(20, #5)
```

This program reads 5 characters into variable A$ and then 20 characters into variable AB$ via buffer No.5.

Values returned by INPUT$ (from keyboard):
Byte 1

L \ H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	*		space	0	@	P	‘	p	Ç	É	á	Ä	Ⓜ	0	α	≡
1			!	1	A	Q	a	q	ü	æ	í	È	Ⓟ	1	β	±
2		INS	"	2	B	R	b	r	é	Æ	ó	Ô	♟	2	Γ	≥
3			#	3	C	S	c	s	â	ô	ú	Á	⚇	3	π	≤
4			$	4	D	T	d	t	ä	ö	ñ	Í	♆	4	Σ	∫
5			%	5	E	U	e	u	à	ò	Ñ	Ú	✓	5	σ	⌡
6			&	6	F	V	f	v	ª	û	º	Ó	┼	6	μ	÷
7			’	7	G	W	g	w	ç	ù	¿	À	⛹	7	τ	≈
8	BS		(	8	H	X	h	x	ê	ÿ	⌐	È	ij	8	Φ	°
9	↵		)	9	I	Y	i	y	ë	Ö	¬	Ô	¨	9	Θ	·
A			*	:	J	Z	j	z	è	Ü	Ã	õ	^	×	Ω	·
B			+	;	K	[	k	{	ï	¢	½	Õ	˜	§	δ	√
C	C•CE	▶	,	<	L	\	l	¦	î	£	¼	φ	¯	‼	∞	n
D	ENTER	◀	−	=	M	]	m	}	ì	¥	¡	Ø	~	→	ø	2
E		▲	.	>	N	^	n	~	Ä	₧	«	Ł	▪	←	∈	■
F		▼	/	?	O	_	o	DEL	Å	ƒ	»	┤	_	◇	∩	

* &H00 indicates the beginning of a 2-byte code.

The key operations listed in the following table cause INPUT$ to return the following codes in the 2nd byte following &H00:

Byte 2

L \ H	0	1	2	3	4	5	6	7	8	9	A	B
0			OFF		CARD			SHIFT ∧	MODE CHECK	PLAY BACK	SHIFT DRG	SHIFT PLAY BACK
1					WORLD			SHIFT ∨	BASIC	ANS	SHIFT BASIC	SHIFT $n!$
2			SHIFT ALARM		LOCAL				STAT	(	SHIFT STAT	SHIFT →DEC
3			SHIFT FUNCTION						AER	)	SHIFT AER	SHIFT →HEX
4			SHIFT OPTION			SHIFT CALENDAR						
5						SHIFT SCHEDULE			SIN		SHIFT ASN	
6						SHIFT TEL	∧		COS		SHIFT ACS	
7			SHIFT 4↔8 LINE			SHIFT MEMO	∨		TAN		SHIFT ATN	
8			SHIFT ANN		SHIFT MARK	SHIFT CALC			EXP		SHIFT π	
9			ALARM EVENT		SHIFT SECRET	SHIFT CARD			IN		SHIFT e^x	
A						SHIFT WORLD			LOG		SHIFT 10^x	
B				CALEN DAR		SHIFT LOCAL			Y^x		SHIFT $\sqrt[x]{y}$	
C			SHIFT CALC DATA	SCHE DULE					$\sqrt{}$		SHIFT $\sqrt[3]{}$	
D			R·CM	TEL					x^2		SHIFT →xy	
E			M−	MEMO					$1/x$		SHIFT →rθ	
F			M+	CALC					→DEG		SHIFT →D.MS	

Note:
ALARM EVENT (&H29) is the code returned when the schedule alarm sounds while waiting for a key to be pressed when the INPUT$ command is being executed.

INPUT#

P
D

FORMAT: INPUT# file number, variable, variable, ..., variable

Abbreviation: I.#
See Also: OPEN, PRINT#

PURPOSE:
Reads items from sequential files on the RAM disk E.

REMARKS:
The file number is the number given to the file when opened for input with the OPEN statement. The file number must be a number specified in an OPEN statement. If the device name is specified as COM in the OPEN statement, it is not necessary to open for input with the OPEN statement.

Specify variables as follows:
- Fixed variables (A, X, etc.)
- Simple variables (CD, EF$, A#, B$, etc.)
- Array elements (B(10), C$(5,5), etc.)

An error occurs if the file contains less data than the number of specified variables. If the file contains more data, the rest of the data is ignored.

The data and variables must be of the same type (e.g., numeric values must be assigned to numeric variables, single-precision values to single-precision variables, double-precision values to double-precision variables, etc.)

Use a comma (,), space (&H20), CR (&H0D), LF (&H0A), or CR + LF as a delimiter when data are read into numeric variables. Spaces preceding data are ignored.

Use a comma (,), CR, LF, or CR + LF as a delimiter to read data into character variables. Spaces preceding data are ignored. If a double quotation mark appears at the beginning of data, data is read up to the next double quotation mark. A comma in a character string enclosed by double quotation marks is assumed not to be a delimiter.

EXAMPLE:

```
 10: A$ = "AB" + CHR$ 34 + "CDE" + CHR$ 34
 20: B$ = CHR$ 34 + "CD,EF" + CHR$ 34
 30: PRINT A$
 40: PRINT B$
 50: OPEN "E:ABC.DAT" FOR OUTPUT AS #2
 60: PRINT #2,A$;",";B$
 70: CLOSE
 80: OPEN "E:ABC.DAT" FOR INPUT AS #2
 90: INPUT #2, C$, D$
100: PRINT C$
110: PRINT D$
120: CLOSE:END
```

Execution

```
RUN [ENTER]  AB"CDE"
             "CD,EF"
             AB"CDE"
             CD,EF
```

KEY (n) ON/OFF

P
D

FORMAT: 1. KEY (key number) ON
2. KEY (key number) OFF

Abbreviation: KE. (n) ON/OFF
See Also: ON KEY GOSUB

PURPOSE:
Enables/disables interrupt requested from a transparent guide key.

REMARKS:
Format 1 enables an interrupt from the transparent guide key with the given key number. The execution of interrupt service is started with the line specified by the ON KEY GOSUB command. The functions printed on the key panel are ignored.
A number from 1 through 20 is allowed as the key number. The key on the upper left corner of the key panel has key number 1, that on the upper right corner has key number 4, and that on the lower right corner has key number 20. The [2ndF] key (key number 5) cannot be used for key interrupt on the OZ-707. To use all transparent guide keys for key interrupt, see the OZ-794A manual.
Format 2 disables the interrupt from the key with the given key number. The key returns to its original function, printed on the key panel.

This command is for one key only. Use a separate command for each key.

EXAMPLE:
See the example for the ON KEY GOSUB command.

KEY 0

P
D

FORMAT: KEY 0, {"character string" / string variable}

Abbreviation: KE.
See Also:

PURPOSE:
Sets the given character string into the key buffer.

REMARKS:
KEY 0 command first clears the key buffer, then sets the given character string into it. Up to 32 bytes of string data can be stored in the key buffer. Characters that cannot be entered directly from the keys can be given with the CHR$ command.

EXAMPLE:
```
10: KEY 0, "ABC"
20: INPUT A$
30: KEY 0, "SHARP" + CHR$ &H0D + "12345" + CHR$ &H0D
40: INPUT B$
50: INPUT C
```

(The string on line 30 has a length of 12 bytes. CHR$ &H0D corresponds to the [ENTER] key.)

KILL

D

FORMAT: KILL "E:filename" [ENTER]

Abbreviation: K.
See Also: SAVE, SET

PURPOSE:
Deletes a file on the RAM disk E.

REMARKS:
Specify the device name and filename. The extension may be omitted if it is blank.

When filenames are displayed using the FILES command, files can be deleted by specifying the file with the [▼] or [▲] key and pressing the [K] and [ENTER] keys.

An error occurs if the specified file does not exist, or is open.
An error occurs if the file attribute is "P".
Change the attribute to "␣" using the SET command to delete a file.

EXAMPLE:
KILL "E:PRO1.BAS"

Deletes the file PRO1.BAS on RAM disk E.

LEFT$

P
D

FORMAT: LEFT$ ("string",expression)

Abbreviation: LEF.
See Also: MID$, RIGHT$

PURPOSE:
Returns the specified number of characters from the left end of a given string.

REMARKS:
LEFT$ returns the number of characters specified by the expression from the left end of the given string.
For example, if A$="ABCD", LEFT$(A$,3) returns the leftmost 3 characters, "ABC".

LEN

P
D

FORMAT: LEN "string"

Abbreviation:
See Also:

PURPOSE:
Returns the number of characters in a string.

REMARKS:
The number of characters in the string includes any blanks or non-printable characters such as control codes or carriage returns.

EXAMPLE:

```
10: INPUT "ENTER A WORD";A$
20: N=LEN A$
30: PRINT "THE WORD LENGTH IS";N
40: END

RUN
ENTER A WORD CHERRY
THE WORD LENGTH IS 6
```

[10] Prompts for a word. In this example, the user enters "CHERRY".
[20] Finds the length of the word.
[30] Prints out the answer.

LET

P
D

FORMAT: 1. LET numeric variable = expression
2. LET string variable = string

Abbreviation: LE.
See Also:

PURPOSE:
Used to assign a value to a variable.

REMARKS:
LET assigns the value of the expression to the designated variable. The type of the expression must match that of the variable; i.e. only numeric expressions can be assigned to numeric variables and only string expressions can be assigned to string variables.

The LET command may be omitted in all LET statements.

LFILES

D

FORMAT: LFILES ["[E:] [filename]"] ENTER

Abbreviation: LF.
See Also: FILES

PURPOSE:
Prints out the names and attributes of the specified file(s) stored on the RAM disk E.

REMARKS:
If no device name is specified, the last device name used will be assumed. If no filename is specified, all files on the specified device will be printed. If neither device nor filename is specified, all files on the last device used will be printed.

Specify a particular filename to print out the name and attributes of that file only. Do not include a filename if you want to print out the names and attributes of all files on disk.

A filename must always be followed by a file extension. Printout appears as "filename.ext attribute." (See the SAVE and SET commands.)

Wildcards (* and ?) can be used to specify filenames.

LINE

P
D

FORMAT: LINE [(expression 1, expression 2)] – (expression 3, expression 4) [,{S / R / X}] [,expression 5] [,{B / BF}]

Abbreviation: LIN.
See Also: GCURSOR, PSET

PURPOSE:
Used to draw a line between two specified points.

REMARKS:
LINE is used to draw a line from the coordinates specified by (expression 1, expression 2) to the coordinates specified by (expression 3, expression 4).

Example:
LINE(0,0) – (95,63)
This statement draws a diagonal line from the upper left corner to lower right corner of the screen.

The values of expressions 1 through 4 should be within the range of –32768 to 32767. To specify points within the screen, use the following range of values:

Expressions 1 and 3: 0 to 95
Expressions 2 and 4: 0 to 63

No error occurs if coordinates outside the screen are specified, provided that the values of the expressions are within the range of –32768 to 32767. In this case only the portion of the line within the range of the screen will appear.

(Expression 1, expression 2) may be omitted. If omitted, a line is drawn from the origin (0, 0) or from the point specified by (expression 3, expression 4) used in a previous LINE statement.

Example:

```
 5: CLS
10: LINE (10,0) – (95,32)
20: WAIT:LINE – (40,63)
```

Note:
Since the screen is made up of a matrix of dots, a diagonal line may appear as a staircase, and curves may not appear as complete curves.

Options S, R, and X are used to set, reset, or reverse the specified line on the screen.

S: Draws a line while activating the corresponding dots on the screen (set).

R: Draws a line while deactivating the corresponding dots on the screen (reset). This option is useful to draw a line in reverse video or to erase an existing line.

X: Draws a line, activating the corresponding dots if they are inactive, or deactivating the corresponding dots if they are already active (reverse).

The default parameter is S.

Expression 5 is used to specify the type of line by a value from 0 to 65535. For example, if the value of expression 5 is 5503 (&H157F), the following line is drawn:

16 dots

The same dot pattern appears repeatedly to draw a dotted line.

Binary representation of 5503 (&H157F) is:
0111111100010101

If you compare this bit pattern with the dot pattern above, you will notice that the dots corresponding to "one" bits are activated while those corresponding to "zero" bits are not. Thus, the type of line is determined by the binary 16-bit pattern of the value of expression 5. If the value is zero, no line appears; and if it is 65535 (&HFFFF), a solid line is drawn. A solid line is also drawn if expression 5 is omitted.

If option R is specified, the dots corresponding to "one" bits are deactivated on the screen; if option X is specified, the status of the dots corresponding to "one" bits are reversed.

Options B and BF are used to draw a rectangle whose opposite corners are specified by (expression 1, expression 2) and (expression 3, expression 4).
Option B is used to draw a line rectangle.
Option BF is used to draw a solid rectangle.

EXAMPLE:

```
10: CLS : WAIT 0
20: AA$ = "081448BF481408"
30: GCURSOR (30,35)
40: GPRINT AA$;AA$;AA$
50: LINE(2,0)-(90,63),&HF18F,B
60: LINE(6,2)-(86,61),X,BF
70: GOTO 60
```

LIST

D

FORMAT: 1. LIST [ENTER]
2. LIST line number [ENTER]
3. LIST *label [ENTER]

Abbreviation: L.
See Also: LLIST, PASS

PURPOSE:
Displays a program.

REMARKS:
Valid only as direct input in the PRO mode.

In format 1, the program is displayed from its first line until the display is full.
In format 2, the program is displayed from the specified line number until the display is full. Use the [∨] key to advance to the next line in the list. If the line for the specified number does not exist, the program will be displayed from the line with the next largest number that does exist.

In format 3, the program is displayed from the line with the specified label until the display is full.

If a password has been set, the LIST command is ignored.
An error will occur if a *label is specified which does not exist in the program or a line number is specified which is past the last line number in program.

LLIST

D

FORMAT: 1. LLIST [ENTER]
2. LLIST { line number / *label } [ENTER]
3. LLIST [line number 1],[line number 2] [ENTER]

Abbreviation: LL.
See Also: LIST, PASS, OPEN

PURPOSE:
Prints out a program on the optional printer.

REMARKS:
Valid only as direct input in the PRO or RUN mode.

When the serial I/O interface has been opened using the OPEN command, the LLIST command outputs the program to the serial I/O interface terminal.
To return the program print command to the printer (CE-50P), execute the CLOSE command.

Format 1 prints or sends all of the programs in memory.
Format 2 prints or sends only the program line for which the number or label is specified.
Format 3 prints or sends the statements from line number 1 through line number 2. There must be at least two lines between the numbers.

Either line number 1 or line number 2 may be omitted. If line number 1 is omitted, the program listing is printed from its first line through line number 2. If line number 2 is omitted, the program listing is printed from line number 1 through the end of the program.

If the line with the line number given in format 2 does not exist or the lines with line numbers 1 and 2 given in format 3 do not exist, the nearest larger numbers are assumed.

If a password has been set, the LLIST command is ignored.

LOAD

D

FORMAT: LOAD ["d:filename" [,R]] [ENTER]
d: E, PACOM, CAS, COM

Abbreviation: LOA.
See Also: SAVE, CHAIN, MERGE, LOAD?

PURPOSE:
Loads a BASIC program.

REMARKS:
LOAD loads a program with the specified filename. An error occurs if the program area is exceeded as a result of loading of a program. In such a case, clear unnecessary variables from the data area.
If program filenames have been displayed with the FILES command, the desired program can be simply loaded by first choosing it with the [▼] or [▲] key, then entering [L] [ENTER].

If all options are omitted, the program is loaded through the serial I/O port.
The file extension may be omitted only if it is ".BAS".
When the device name (d:) is PACOM, CAS, or COM, only ASCII code data is valid.

If a load error occurs, a program written in intermediate format will not be loaded at all, but a program written in ASCII format is loaded to the line just before the line where the error occurred.

Specify ",R" to run the program as soon as it is loaded.

While a program is being loaded, all files are closed, and all values or variables specified for the USING, ON ERROR GOTO, WAIT, LOCATE, ERL, and ERN commands are cleared.

Up to 256 bytes of program code can be loaded at a time. An error occurs if no delimiter is encountered before loaded data exceeds 256 bytes.
When a serial I/O device is used for loading, delimiters and the end-of-file code may be set with the OPEN command. (If no end-of-file code was received, use the [ON] key to stop loading.)

When a delimiter is received, the computer translates the loaded data into an intermediate format. An error occurs if a single line exceeds 256 bytes or the first character of a line is not a numeral (line number).

LOAD?

D

FORMAT: LOAD? ["d:filename"] [ENTER]
d: E, PACOM, CAS, COM

Abbreviation: LOA.?
See Also: LOAD

PURPOSE:
Compares a program saved on a device with one stored in memory.

REMARKS:
LOAD? compares the program with the given filename with the one in memory. An error occurs if a mismatch was found during comparison.

If all options are omitted, LOAD? compares the program read from the serial I/O port with the one in memory.
The file extension may be omitted only if it is ".BAS".
When the device name (d:) is PACOM, CAS, or COM, only ASCII format data is valid.

Up to 256 bytes of program code can be loaded at a time. An error occurs if no delimiter is encountered before loaded data exceeds 256 bytes.
When a serial I/O device is used for loading, delimiters and the end-of-file code may be set with the OPEN command. (If no end-of-life code was received, use the [ON] key to stop loading.)

LOC

P
D

FORMAT: LOC file number

Abbreviation:
See Also: OPEN

PURPOSE:
Returns the current pointer position (logical) in a file.

REMARKS:
The LOC command returns the number of records read or written since the file with the specified number was opened. One record is 256 bytes long. If the device name is COM, LOC returns the number of data bytes stored in the 20-character (40-byte) buffer.
An error occurs if a file with the specified number has not been opened.

EXAMPLE:

```
10: OPEN "E:FILE01" FOR INPUT AS #2
20: IF EOF(2) THEN 50
30: INPUT #2,N
40: GOTO 20
50: M=LOC(2)
60: PRINT "THE FILE HAS";M;"RECORDS"
70: CLOSE #2
80: END
```


LOCATE

P
D

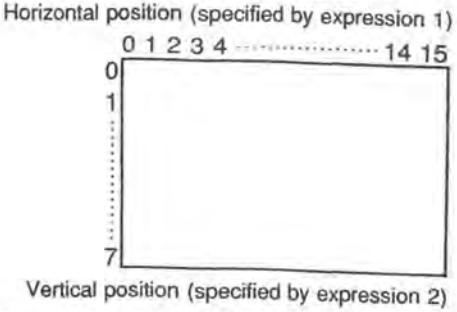
FORMAT: 1. LOCATE [expression 1] [,expression 2]
2. LOCATE

Abbreviation: LO.
See Also: CLS, INPUT, PRINT, PAUSE, GCURSOR, WIDTH

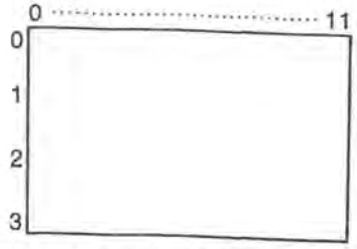
PURPOSE:
Specifies the display start position in character units.

REMARKS:
Specifies the display start position in units of a character position for the contents displayed by the PRINT command, PAUSE command, etc.

When WIDTH,8 has been specified, the display position is specified as follows using format 1:



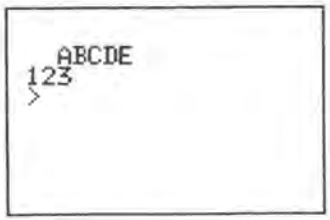
When WIDTH,4 has been specified, the display position is specified as follows using format 1:



If expression 1 is omitted, the horizontal position 0 is assumed.
If expression 2 is omitted, the current cursor vertical position is assumed.

Example:

```
10: CLS
20: LOCATE 2,1: PRINT "ABCDE"
30: LOCATE ,2: PRINT "123"
```



Format 2 clears the display start position.

Using the LOCATE command allows text to be written to any part of the display without affecting existing text except where characters are directly overwritten. Use the CLS command to clear the whole display.

If the number of characters exceed the limits of the display, the display is scrolled to show all the characters, even if the display start position was specified with the LOCATE command.

LOF

P
D

FORMAT: LOF file number

Abbreviation:
See Also: OPEN

PURPOSE:
Returns the size of the specified file.

REMARKS:
The LOF command returns the size of a file with the specified file number. The actual size of the file is displayed in bytes.
If the device name is COM, this command returns the byte count remaining in the 20-character (40-byte) buffer.

An error will occur if the specified file is not open.

RAM disk E is used in units of 256 bytes; the total size of all files will not be equal to the total used RAM disk area (number of bytes).

EXAMPLE:

```
10: OPEN "E:FILE01" FOR INPUT AS #2
20: N=LOF(2)
30: PRINT "FILE01 FILE SIZE IN BYTES IS ";N
40: CLOSE #2
50: END
```

[10] Opens the file FILE01 for input.
[20–30] Finds the size of the file and prints out the value.
[40] Closes the file.

LPRINT

P
D

FORMAT:

1. LPRINT {expression / string} [, {expression / string}] [,]
2. LPRINT {expression / string} [; {expression / string}] [;]
3. LPRINT USING "format";
   {expression / string} [{, / ;} {expression / string}] [{, / ;}]
4. LPRINT

Abbreviation: LP.
See Also: PRINT, USING, OPEN, WIDTH

PURPOSE:
Outputs given data to the printer.

REMARKS:
When the serial I/O interface is opened with the OPEN command, the LPRINT command outputs the program to the serial I/O interface terminal. To return the printing command to the CE-50P Printer, execute the CLOSE command.

When a comma (,) or semicolon (;) is placed at the end of the statement, the next LPRINT command in the program will print its data directly after the data printed by the first LPRINT.

The CE-50P prints in the same format as the display specified with the PRINT command. If a numerical value fits on one line of the paper, it will be printed from the right margin. If it does not fit on one line, it will be printed from the left margin and continued on the following line(s). Character strings will be printed from the left margin and continued on the following line(s).

For output to the serial I/O terminal, both numerical values and strings are printed from the left margin.

Format 2 prints out data in succession from the left margin. When a semicolon (;) is placed at the end of the statement, the next LPRINT command in the program will print its data in succession to the data printed by the first LPRINT.

Format 3 prints out data in the exact format specified in the statement. Either a comma (,) or semicolon (;) may be used as a separator. For the format for USING, see the USING command.

Format 4 prints only delimiter codes. If the preceding LPRINT statement is terminated with a semicolon (;) and unprinted data is left in the buffer, format 4 prints that data.

MDF

P
D

FORMAT:

1. MDF (expression)
2. MDF (expression, threshold number)

Abbreviation: MD.
See Also: USING

PURPOSE:
Rounds up the value of an expression.

REMARKS:
The MDF function rounds the value of an expression to the number of decimal places specified by the USING command.

MDF is effective only when the number of decimal places is specified for a value by the USING command.

Format 1 uses the standard default threshold number of 4. This means that if the first digit of the truncated part is more than 4, one is carried to the non-truncated part. This threshold number can be specified using format 2.

MERGE

D

FORMAT: MERGE "[d:] filename" ENTER
d: E, PACOM, CAS, COM

Abbreviation: MER.
See Also: LOAD, CHAIN

PURPOSE:
Loads a program file from the specified device and merges it with a program in memory.

REMARKS:
MERGE retains the program in memory and then loads the specified program.
The program file to be merged must have ASCII format (see the SAVE command).
Note:
When a program in intermediate format is loaded, an abnormal condition may occur.

If the line numbers overlap, the lines of the specified file replace the lines of the program in memory.
If the device name is omitted, the device name used in the last file command is assumed. For example, if the last file command is FILES"E:", RAM disk E is assumed as the device name.

An error occurs if the program overflows the program area as a result of a MERGE command. Clear unnecessary variables, then try MERGE again.

EXAMPLE:

Program in memory
```
10: INPUT A, B
15: PRINT A, B
20: C = SQR (A * A + B * B)
25: PRINT C
30: END
```

Program to be merged (ASCII format)
```
10: INPUT A, B, C
20: S = (A + B + C)/2
30: AREA = SQR (S * (S – A) * (S – B)
    * (S – C))
40: PRINT AREA
50: END
```

Program after merge
```
10: INPUT A, B, C
15: PRINT A, B
20: S = (A + B + C)/2
25: PRINT C
30: AREA = SQR (S * (S – A) * (S – B) * (S – C))
40: PRINT AREA
50: END
```

The contents of lines 10, 20, and 30 are replaced with those of the same lines of the loaded program.

MID$

P
D

FORMAT: MID$(string,N,M)

Abbreviation: M.
See Also: LEFT$, RIGHT$

PURPOSE:
Returns a string of M characters from inside a string starting from the Nth character in the string.

REMARKS:
If N is greater than the number of characters in the string, a null string is returned. If N is less than 1, an error occurs. M must be in the range of 0 to 254 and N in the range of 1 to 254. Fractions will be truncated.

NAME

D

FORMAT: NAME "E:old filename" AS "new filename" [ENTER]

Abbreviation: NA.
See Also:

PURPOSE:
Renames files on the RAM disk E.

REMARKS:
The NAME command renames the disk file "old filename" as "new filename" on the RAM disk.

An error occurs if "old filename" does not exist, or a file with "new filename" already exists. An error occurs if "old filename" is protected with the "P" (write-protection) function. An error occurs if "old filename" is open.

The extension can be omitted if it is blank.

EXAMPLE:
NAME "E:OLDNAM" AS "E:NEWNAM"

Names file OLDNAM on the RAM disk E as NEWNAM.

NEW

D

FORMAT: NEW [ENTER]

Abbreviation:
See Also: CLEAR, PASS

PURPOSE:
Clears existing programs and data.

REMARKS:
The NEW command clears all programs and data that are currently in memory. (Programs with passwords cannot be cleared.)

ON ERROR GOTO

P

FORMAT: 1. ON ERROR GOTO {line number / *label}
2. ON ERROR GOTO 0

Abbreviation: O. ERR. G.
See Also: ERN, ERL, RESUME

PURPOSE:
Sets up an error trapping routine.

REMARKS:
When the error trap function is enabled, control is transferred to the error routine if an error is detected. An error message is not displayed.

Within an error routine, control can be branched depending on the value of ERN and ERL. The routine is terminated by the RESUME command.

Format 2 clears declaration of error trapping. Declaration of error trapping is also cleared in any of the following cases:
(1) RUN command is executed.
(2) The power is turned off.
(3) A program is loaded.

The program stops executing if an error occurs within an error routine.

ON...GOSUB

P

FORMAT: ON expression GOSUB
{line number 1 / *label 1} , {line number 2 / *label 2} ,...

Abbreviation: O. GOS.
See Also: GOSUB, GOTO, ON...GOTO

PURPOSE:
Executes one of a set of subroutines, depending on the value of a control expression.

REMARKS:
When ON...GOSUB is executed, the expression between ON and GOSUB is evaluated and reduced to an integer. If the value of the integer is 1, control is transferred to line number 1 or *label 1 in the list, as in a normal GOSUB. If the expression is 2, control is transferred to line number 2 or *label 2 in the list, and so forth.

Note:
Be sure to place a space just before the GOSUB command. Otherwise it may be regarded as a variable.

If the expression is zero, negative, or larger than the number of line numbers provided in the list, no subroutine is executed and execution proceeds with the next statement or line of the program.

An error occurs if the value of the expression is –32769 or less or 32768 or more.

Use commas (,) to separate line numbers or *labels in the list.

EXAMPLE:

```
 10: INPUT A
 20: ON A GOSUB 100,200,300
 30: END
100: PRINT "FIRST"
110: RETURN
200: PRINT "SECOND"
210: RETURN
300: PRINT "THIRD"
310: RETURN
```

An entry of 1 displays "FIRST"; 2 displays "SECOND"; 3 displays "THIRD". Any other entry does not produce any display.

ON...GOTO

P

FORMAT: ON expression GOTO {line number 1 / *label 1} , {line number 2 / *label 2} ....

Abbreviation: O. G.
See Also: GOSUB, GOTO, ON...GOSUB

PURPOSE:
Transfers control to one of a set of locations, depending on the value of a control expression.

REMARKS:
When ON...GOTO is executed the expression between ON and GOTO is evaluated and reduced to an integer. If the value of the integer is 1, control is transferred to line number 1 or *label 1 in the list. If the expression is 2, control is transferred to line number 2 or *label 2 in the list, and so forth.

Note:
Be sure to place a space just before the GOTO command. Otherwise it may be regarded as a variable.

If the expression is zero, negative, or larger than the number of line numbers provided in the list, execution proceeds with the next statement or line of the program.

An error occurs if the value of the expression is –32769 or less or 32768 or more.

Use commas (,) to separate line numbers or *labels in the list.

EXAMPLE:

```
 10: INPUT A
 20: ON A GOTO 100,200,300
 30: GOTO 900
100: PRINT "FIRST"
110: GOTO 900
200: PRINT "SECOND"
210: GOTO 900
300: PRINT "THIRD"
310: GOTO 900
900: END
```

An entry of 1 displays "FIRST"; 2 displays "SECOND"; 3 displays "THIRD". Any other entry does not produce any display.

ON KEY GOSUB

P

FORMAT: ON KEY GOSUB {line number 1 / *label 1}, {line number 2 / *label 2}, ... {line number 20 / *label 20}

Abbreviation: O. KE. GOS.
See Also: KEY (n) ON/OFF, ON...GOSUB, GOSUB...RETURN

PURPOSE:
Defines the interrupt service line for a transparent guide key.

REMARKS:
ON KEY GOSUB command passes execution to the line with the line number given by this command when a transparent guide key with the key number given in the KEY (key number) ON command is pressed.

Line number 1 gives the first line of the interrupt service routine to which execution is passed when transparent guide key 1 is pressed; line number 2 gives the line to which execution is passed when transparent guide key 2 is pressed, and so on.

Use commas to separate line numbers. No line numbers need be specified for keys which are not ON, but the comma separator is necessary. If guide key n is the highest guide key used, n-1 commas are required. Line numbers may also be specified with *labels. An error occurs if a specified line was not found.

Since interrupt service is executed in a subroutine, the last line of every routine must be a RETURN statement.

EXAMPLE:
```
10: CLS: A=0:WAIT
20: KEY (2)ON :KEY (4)ON
30: KEY (10)ON :KEY (16)ON
40: ON KEY GOSUB ,80,,90,,,,,,100,,,,,,110
50: A=A+1
60: WAIT :LOCATE 2,0:PRINT A
70: GOTO 50
80: WAIT 10:LOCATE 0,2:PRINT "key 2":RETURN
90: WAIT 10:LOCATE 0,2:PRINT "key 4":RETURN
00: WAIT 10:LOCATE 0,2;PRINT "key10":RETURN
10: WAIT 10:LOCATE 0,2:PRINT "key16":RETURN
```


OPEN

P
D

FORMAT: 1. OPEN "d:filename" FOR mode AS # file number
d: E, PACOM, CAS
2. OPEN "baud rate, parity, word length, stop bit, type of code, delimiter, end-of-file code, XON, shift code" AS # file number
3. OPEN

Abbreviation: OP.
See Also: CLOSE, OPEN$

PURPOSE:
Opens a file or the serial I/O interface for data transfer.

REMARKS:
Format 1 opens the file specified by "d:filename" for use with the specified file number. Subsequent input/output to the file is accomplished by referring to the file number.
Formats 2 and 3 allow data to be transferred through the serial I/O interface (COM).

The file number must be from 1 to 255.

A maximum of two files on any devices may be opened at the same time. Only the following device combinations are possible:

	E	PACOM	CAS	COM*
E	O	O	O	O
PACOM	O	×	×	O
CAS	O	×	×	O
COM*	O	O	O	×

O: may be opened at the same time.
×: may not be opened at the same time.
* To open files on the device COM and another device at the same time, open the file on the device COM first.

Note:
When executing the SAVE, LOAD, CHAIN, or MERGE statement, the Card opens one file automatically.

In format 1, "mode" specifies the method of access to the file, as follows:

INPUT	Specifies sequential input from an existing file.
OUTPUT	Specifies sequential output to a device or file.
APPEND	Specifies addition to a sequential file.

If OUTPUT is specified using an existing filename, that file is erased before the new one is created.

An error occurs when using the APPEND or INPUT mode if the specified file does not exist.
An error occurs when using the APPEND or OUTPUT mode if the specified file has the "P" attribute set (write-protection: see SET command).
An error occurs if an attempt is made to open a file which has already been opened, or to allocate a file number which has already been allocated.

Example:
For RAM disk E
OPEN "E:PRO1" FOR OUTPUT AS #21
Creates a new file on the RAM disk E named PRO1 with file number 21.

In format 2, the following parameters can be selected:
Baud Rate: 300, 600, 1200, 2400, 4800, 9600
Specifies the modulation rate (transfer rate).
(1 baud = 1 bit/sec)
Parity: N, E, O
Specifies the type of parity.
N: No parity bit is transmitted nor received.
E: Specifies even parity.
O: Specifies odd parity.
Word Length: 7, 8
Specifies how many bits to be transmitted or received per character.
Number of Stop Bits: 1, 2
Type of Code: A
Always specify A, since only ASCII codes can be sent or received.
Delimiter: C, F, L
Specifies the type of delimiter to indicate the end of data, end of a program line, etc.
C: Specifies the CR (carriage return) code.
F: Specifies the LF (line feed) code.
L: Specifies the CR code + LF code.
End-of-file Code: &H00–&HFF
Specifies the end-of-file code used to indicate the end of the program, etc.
(May be required when using the SAVE or LOAD commands.)

XON: N, X
Specifies communication control using XON and XOFF.
N: Specifies no control using XON and XOFF.
X: Specifies control using XON and XOFF.
Shift code: S, N
When the word length is specified as 7 characters, the SO, SI switching enables transfer of characters whose codes are 128 or greater.
S: Specifies to switch SO and SI
N: Specifies not to switch SO and SI

Example:
OPEN "COM:1200,N,8,1,A,C,&H1A,N,S" AS #22
1200.....Baud rate (1200 baud)
N.....Parity (none)
8.....Word length (8 bits)
1.....Number of stop bits (1 bit)
A.....Type of code (ASCII)
C.....Delimiter (CR code)
&H1A....End-of-file code (&H1A)
N.....XON (none)
S.....Shift code (Yes)

The device name "COM:" can be omitted.

The conditions in the example above are set after the batteries have been replaced or after the RESET button is pressed.

Any condition specified in the OPEN command can be omitted. If omitted, the current condition remains unchanged.

OPEN ",,,2"
Only the number of stop bits is changed.

In format 3, all conditions set previously are retained. This format enables data to be transferred through the I/O interface. File number 1 is always used.

Note:
To open files on the device COM and another device at the same time, open the file on the device COM first. If the file on another device is opened first, the files may not be opened properly.

OPEN$

P
D

FORMAT: OPEN$ [ENTER]

Abbreviation: OP.$
See Also: OPEN

PURPOSE:
Obtains the currently set I/O conditions.

REMARKS:
The currently set I/O conditions are obtained as a character string.

EXAMPLE:
OPEN$ [ENTER]
1200, N, 8, 1, A, C, &H1A, N, S

PASS

D

FORMAT: PASS "character string" [ENTER]

Abbreviation: PA.
See Also: SAVE, LOAD, DELETE, LIST, NEW, RENUM

PURPOSE:
Sets and cancels passwords.

REMARKS:
Passwords are used to protect programs from listing or editing by unauthorized users. A password consists of a character string that is no more than 8 characters long. The 8 characters must be alphanumeric characters or symbols. The character string cannot be a null string.

Once a PASS command has been given, the programs in memory are protected. A program protected by a password cannot be examined or modified in memory. It cannot be saved to tape or disk, or listed with LIST or LLIST. Nor is it possible to add or delete program lines. The only way to remove this protection is to execute another PASS command with the same password.

If a password is set in the program to be loaded, that password is also set within the Card. If not, no password is set within the Card.
If PASS is executed when no program is in the Card, an error occurs and no password is set.
A password-protected program is protected against the NEW or DELETE command as well.

Press [ENTER] immediately after the password.
Writing characters or symbols after the password results in an error and the password cannot be canceled.

Example:
PASS "ABCDEFG":A = 123 [ENTER] → An error occurs.

EXAMPLE:
PASS "SECRET" [ENTER]

Establishes the password "SECRET" for the program in memory.

PAUSE

P
D

FORMAT:

1. PAUSE {expression / string} [, {expression / string}] [;]
2. PAUSE {expression / string} [; {expression / string}] [;]
3. PAUSE USING "format";
   {expression / string} [{, / ;} {expression / string}] [{, / ;}]
4. PAUSE

Abbreviation: PAU.
See Also: PRINT

PURPOSE:
Briefly shows information on the display.

REMARKS:
PAUSE is used to display prompt information, results of calculations, etc. The operation of PAUSE execution of the program continues following a preset interval of about .85 seconds.

This command is provided to ensure compatibility with other PC models. It is recommended that you replace it with the PRINT command wherever possible.

For the format with the USING command, see the USING command.

POINT

P
D

FORMAT: POINT (expression 1, expression 2)

Abbreviation: POI.
See Also: GCURSOR, PSET, PRESET

PURPOSE:
Returns the status of a specified dot.

REMARKS:
POINT returns 1 if the dot specified by coordinates (expression 1, expression 2) is set, and returns zero if it is cleared. If the specified dot is outside the display boundaries, the command returns –1.

The values of expressions 1 and 2 may be within the range of –32768 to 32767. A dot within the display boundaries is addressed only if the value of expression 1 is 0 to 95 and that of expression 2 is 0 to 63.

EXAMPLE:

```
10: CLS : WAIT 3:A = 45
20: LINE (20,10) - (20,41)        Draws two vertical lines.
30: LINE (80,10) - (80,41)
40: PSET (A,26)                   Sets a dot between the two lines.
50: B = POINT (A + 1,26)          Tests whether the dot on the right
                                  side of the active dot is active or not.
60: IF B THEN 150                 If it is set, go to line 150.
70: PSET (A + 1,26)               If it is cleared, set it.
80: PRESET (A, 26)                Inactivates the dot which was first set.
90: A = A + 1                     Increments the coordinate to address
                                  the next dot position.
100: GOTO 50                      Returns to line 50.
150: B = POINT (A - 1,26)         Tests whether the dot on the left side
                                  of the active dot is active or not.
160: IF B THEN 50                 If it is set, go to line 50.
170: PSET (A – 1,26)              If it is cleared, set it.
180: PRESET (A,26)                Clears the dot which was first set.
190: A = A – 1                    Decrements the coordinate to address
                                  the preceding dot position.
200: GOTO 150                     Go to line 150.
```

Executing this program causes a dot to move back and forth between the two vertical lines.

PRESET

P
D

FORMAT: PRESET (expression 1, expression 2)

Abbreviation: PRE.
See Also: PSET, GCURSOR, POINT

PURPOSE:
Clears (resets) a dot at the specified coordinates on the display.

REMARKS:
PRESET clears the dot at the specified (expression 1, expression 2).

The values of expressions 1 and 2 may be within the range of –32768 to 32767. A dot within the display boundaries is addressed only if the value of expression 1 is 0 to 95 and that of expression 2 is 0 to 63.

EXAMPLE:

```
10: CLS
20: LINE (6,0) - (90,63), BF
30: FOR I = –1 TO 1 STEP 2
40: FOR X = –25 TO 25 STEP 0.5
50: Y = I * SQR ABS (25 * 25 –X * X)
60: PRESET (X + 48, Y + 31)
70: NEXT X : NEXT I
80: WAIT : GPRINT
```

Executing this program draws a circle within a solid rectangle.

PRINT

P
D

FORMAT:

1. PRINT $\left\{\begin{matrix}\text{expression}\\\text{string}\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{expression}\\\text{string}\end{matrix}\right\}\right][,]$
2. PRINT $\left\{\begin{matrix}\text{expression}\\\text{string}\end{matrix}\right\}\left[;\left\{\begin{matrix}\text{expression}\\\text{string}\end{matrix}\right\}\right][;]$
3. PRINT USING "format";
   $\left\{\begin{matrix}\text{expression}\\\text{string}\end{matrix}\right\}\left[\left\{\begin{matrix},\\;\end{matrix}\right\}\left\{\begin{matrix}\text{expression}\\\text{string}\end{matrix}\right\}\right]\left[\left\{\begin{matrix},\\;\end{matrix}\right\}\right]$
4. PRINT

Abbreviation: P.
See Also: LPRINT, PAUSE, USING, WAIT, LOCATE

PURPOSE:
Displays information.

REMARKS:
PRINT displays prompt information, results of calculations, etc.

If the start position is specified by the LOCATE statement, the data will be displayed from the specified location.
If a comma (,) or semicolon (;) is at the end of the statement, the contents will be displayed continuously.

Format 1 displays as follows:
If the expression is numeric, the value is shown from the right margin of the display. If it is a string, it is shown from the left margin of the display.

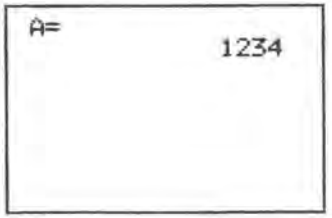
```
10: PRINT 1234
20: PRINT "ABCD"
```

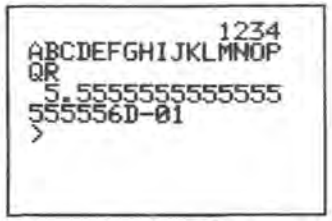
```
RUN
                    1234
ABCD
>
```

If a numerical value fits on one line of the display, it will be displayed from the right margin. If it does not fit on one line, it will be displayed from the left margin and continued on the following line(s). Character strings will be displayed from the left margin and continued on the following line(s).

```
10: A = 1234: B# = 5#/9: C$ = "ABCDEFGHIJKLMNOPQR":WAIT 200
20: CLS: PRINT "A=",A
30: CLS: PRINT A,C$,B#
```

RUN [ENTER]





Format 2 displays the data continuously from the left margin of the display.

Format 3 displays the data by following the specified format. Refer to the USING command for USING format. Commas (,) and semicolons (;) will be treated as usual. A USING statement can be used only once in one PRINT statement.

Example: PRINT USING "&&&&&&&&";"ANSWER=";:PRINT USING "####.##";5/9

Format 4 displays the previously displayed value as is. (Usually, it is used together with the WAIT command to retain the current display.)

```
10: CLS
20: FOR A=0 TO 127
30: PRINT CHR$ (A+32);
40: NEXT A
50: WAIT: PRINT
```

The characters displayed between lines 20 and 40 will remain on the display at line 50. (An infinite interval is set.)

PRINT → LPRINT setting

The Card can switch all PRINT commands to function as LPRINT commands. Connect the printer before executing the following statement:

Setting: PRINT=LPRINT
Resetting: PRINT=PRINT

Resetting can also be performed by:

- executing the RUN command,
- pressing the [SHIFT] [C•CE] keys, or
- turning the power off and then on.

Since the RUN command resets the setting, run the program using the GOTO command.

PRINT#

P
D

FORMAT: PRINT# file number, {expression / string} [{, / ;} {expression / string}] [{, / ;}]

Abbreviation: P.#
See Also: OPEN, INPUT#

PURPOSE:
Writes values of specified variables into a specified file.

REMARKS:
PRINT# is valid only for a file opened for OUTPUT or APPEND with the OPEN command. The file number is the number given to the file when opened.
The mode does not need to be specified when the device name is COM.

When an array variable (one or two dimensions) has been specified in the form of "array name(*)", the entire array is written to the file. Its elements are written in the order of, for example, C$(0,0), C$(0,1), C$(0,2)....C$(1,0).....C$(5,5).
It is recommended that the FOR...NEXT statement be used for writing array variable data.

When the respective elements of the array are specified, they must be specified in the form of "B(7)", "C$(5,6)", etc.

When a character or string element is used, it must not be specified using a comma (,) or semicolon (;):

```
PRINT#2,"ABC"
PRINT#2,A$
```

If PRINT#2,"ABC",A$ is executed, no data delimiter is written and "ABC" and A$ cannot be distinguished.

A numeric value is recorded in such a form that the sign (space when it is positive), numeric character string, and space appear in that order. The recording format is shown below:

(1) When a comma or semicolon does not follow the data, CR(&H0D) and LF(&H0A) are provided.

Example:

PRINT #2, −1.2

−	1	.	2		CR	LF

PRINT #2, "ABC"

A	B	C	CR	LF

(2) When a comma follows the data, 20 bytes are occupied. A numeric value is right justified and a character string is left justified.

Example: PRINT #2, − 1.2,3

		~	−	1	.	2			3		CR	LF

20 bytes

PRINT #2, "ABC", "DEF"

A	B	C	~	D	E	F	CR	LF

20 bytes

When the character string exceeds 20 bytes, the excess part is written to the next 20-byte area. The maximum size is 254 bytes.

(3) When a semicolon follows the data, it is stored without spaces.

Example:

PRINT #2, − 1.2;3

−	1	.	2			3		CR	LF

PRINT #2, "ABC";"DEF"

A	B	C	D	E	F	CR	LF

In this case, character strings "ABC" and "DEF" will not be read as separate strings during an INPUT# command.

When character strings are recorded with commas or semicolons, they must be read with the INPUT$ command by specifying the exact format in which they were recorded.

EXAMPLE:
Transmitting data using device name "PACOM"

Sender

```
10: DIM C (2,3)
20: OPEN "PACOM: DATA"
    FOR OUTPUT AS #2
30: FOR I=0 TO 2
40: FOR J=0 TO 3
50: C(I,J)=I*10+J
60: PRINT #2, C(I,J)
70: NEXT J: NEXT I
80: CLOSE
90: END
```

Receiver

```
10: DIM C (2,3)
20: OPEN "PACOM: DATA"
    FOR INPUT AS #3
30: FOR I=0 TO 2
40: FOR J=0 TO 3
50: INPUT#3, C(I,J)
60: NEXT J: NEXT I
70: CLOSE
80: END
```

After entering RUN [ENTER] on the receiver, enter RUN [ENTER] on the sender.

PSET

P
D

FORMAT: 1. PSET (expression 1, expression 2)
2. PSET (expression 1, expression 2) ,X

Abbreviation: PS.
See Also: PRESET, GCURSOR, POINT

PURPOSE:
Sets or clears a dot at the specified coordinates on the display.

REMARKS:
Format 1 sets the dot at the coordinates (expression 1, expression 2). Format 2 clears the specified dot if it is set, and sets it if it is cleared. The values of expressions 1 and 2 may be within the range of –32768 to 32767. A dot on the display is addressed only if the value of expression 1 is 0 to 95 and that of expression 2 is 0 to 63.

EXAMPLE:

```
10: CLS :DEGREE
20: FOR A=0 TO 600 STEP 3
30: B= –1 * SIN A
40: Y=INT (B *32)+32
50: X=INT (A/4)
60: PSET (X,Y)
70: NEXT A
80: WAIT:GPRINT
```


RADIAN

P
D

FORMAT: RADIAN

Abbreviation: RAD.
See Also: DEGREE, GRAD

PURPOSE:
Changes the form of angular values to radians.

REMARKS:
There are three forms for representing angular values — degrees, radians, and gradient. These forms are used in specifying the arguments to the SIN, COS and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The RADIAN function changes the form of all angular values to radian form until DEGREE or GRAD is used. Radian form represents angles in terms of the length of the arc with respect to the radius, i.e., 360° is 2π radians, since the circumference of a circle is 2π times the radius.

RANDOMIZE

P
D

FORMAT: RANDOMIZE

Abbreviation: RA.
See Also: RND

PURPOSE:
Resets the seed for random number generation.

REMARKS:
When random numbers are generated using the RND function, the Card begins with a predetermined "seed" or starting number. RANDOMIZE resets this seed to a new randomly determined value.

The starting seed will be the same each time the power is turned on, so the sequence of random numbers generated with RND is the same each time, unless the seed is changed. This is very convenient during the development of a program because it means that the behavior of the program should be the same each time it is run, even though it includes a RND function. When you want the numbers to be truly random, the RANDOMIZE statement can be used to make the seed itself random.

READ

P

FORMAT: READ variable, variable, ..., variable

Abbreviation: REA.
See Also: DATA, RESTORE

PURPOSE:
Reads values from a DATA statement and assigns them to variables.

REMARKS:
When assigning initial values to an array, it is convenient to list the values in a DATA statement and use a READ statement in a FOR...NEXT loop to load the values into the array. When the first READ is executed, the first value in the first DATA statement is returned. Succeeding READs use succeeding values in the order in which they appear in the program, regardless of how many values are listed in each DATA statement or how many DATA statements are used.

If desired, the values in a DATA statement can be read a second time using the RESTORE statement.

Note:
The type of data must match the type of variables (numerical or string) to which it is to be assigned.

REM(')

P

FORMAT: 1. REM remark
2. ' remark

Abbreviation:
See Also:

PURPOSE:
Includes comments in a program.

REMARKS:
It is often useful to include explanatory comments in a program. These can provide titles, names of authors, dates of last modification, usage notes, reminders about algorithms, etc. These comments are included using the REM (or apostrophe (')) statement.

The REM (') statement has no effect on program execution and can be included anywhere in the program. Everything following REM (') in that line is treated as a comment.

RENUM

D

FORMAT: RENUM [new line number] [, [old line number] [,increment]]
ENTER

Abbreviation: REN.
See Also: DELETE, LIST

PURPOSE:
Renumbers the lines of a program.

REMARKS:
Valid only as direct input in the PRO mode.

The line numbers are changed from old line numbers to new line numbers with the specified increment. If the new line number is not specified, the lines are renumbered starting with line 10. If the increment is not specified, the lines are renumbered with an increment of 10. RENUM updates referenced line numbers in GOTO, ON...GOTO, GOSUB, ON...GOSUB, RESTORE, and (IF)...THEN statements.

An error occurs if line numbers are given as a variable (GOTO A) or numerical expression (GOTO 2 X 50). If an error occurs, renumbering is not carried out. If a line number is given by a variable or expression, temporarily make it a remark (REM), and correct it after executing the RENUM command. It is recommended that you replace such commands with ON...GOTO commands, etc.

If a line number exceeds 65279, an error is generated. If a specified old line number does not exist, an error is generated. Changing the execution order generates an error. If a password has been used, an error occurs.

If the display shows "*", pressing the ON key will interrupt renumbering. A display of "**" indicates that renumbering cannot be interrupted. Error generation or use of the ON key leaves the program unchanged.

EXAMPLE:

```
10: INPUT "CONTINUE";A$
20: IF A$ = "YES" THEN 10
30: IF A$ = "NO" THEN 60
40: PRINT "ENTER YES OR NO PLEASE!"
50: GOTO 10
60: END

RENUM 100, 10, 5 [ENTER]

100: INPUT "CONTINUE";A$
105: IF A$ = "YES" THEN 100
110: IF A$ = "NO" THEN 125
115: PRINT "ENTER YES OR NO PLEASE!"
120: GOTO 100
125: END
```


RESTORE

FORMAT: 1. RESTORE {line number / *label}

2. RESTORE

Abbreviation: RES.
See Also: DATA, READ

PURPOSE:
Rereads values in a DATA statement or changes the order in which these values are read.

REMARKS:
In the regular use of READ, reading begins with the first value in a DATA statement and proceeds sequentially through the remaining values. Format 1 resets the pointer to the first value of the DATA statement whose line number is equal to the specified line number or *label. Format 2 resets the pointer to the first value of the first DATA statement, so that it can be read again.

EXAMPLE:

```
10: DIM B(10)
20: WAIT 32
30: FOR I = 1 TO 10
40: RESTORE
50: READ B(I)
60: PRINT B(I)*I;
70: NEXT I
80: DATA 20
90: END
```

[10] Sets up an array.
[50] Assigns the value 20 to each of the elements of B( ).

RESUME

P

FORMAT: 1. RESUME
2. RESUME NEXT
3. RESUME {line number / *label}

Abbreviation: RESU.
See Also: ON ERROR GOTO

PURPOSE:
Resumes program execution at the end of an error handling routine.

REMARKS:
RESUME resumes program execution after completing an error handling routine to which control was passed by the ON ERROR GOTO command. This command validifies the ON ERROR GOTO command again. If control is returned to the main program by any other command (GOTO, etc.), execution will be aborted if an error subsequently occurs.

The error handling routine lets you take the necessary action to prevent recurrence of the same error.

Control is returned depending on the format:
(1) Format 1 returns control to the statement which caused the error. If an error occurs again in the same statement, the error handling routine is executed again.
(2) Format 2 returns control to the statement following the error statement.
(3) Format 3 returns control to the specified line.

EXAMPLE:

```
 10: ON ERROR GOTO 100
 20: INPUT A, B
 30: PRINT A/B
   :
   :
100: RESUME 20
   :
```

If zero is assigned to variable B or an overflow occurs from A/B, control returns to the input routine on line 20 and prompts for correct data entry.

RIGHT$

P
D

FORMAT: RIGHT$(string,N)

Abbreviation: RI.
See Also: LEFT$, MID$

PURPOSE:
Returns N characters from the right end of a string.

REMARKS:
Fractions will be truncated. If N is less than 1, a null string is returned. If N is greater than the number of characters in the string, the whole string is returned.

RND

P
D

FORMAT: RND numeric expression

Abbreviation: RN.
See Also: RANDOMIZE

PURPOSE:
Generates a random number.

REMARKS:
If the value of the expression is less than 1 but greater than or equal to zero, the random number is less than 1 and greater than zero. If the expression is an integer greater than or equal to 1, the result is a random number greater than or equal to 1 and less than or equal to the expression. If the expression is greater than or equal to 1 and not an integer, the result is a random number greater than or equal to 1 and less than or equal to the smallest integer that is larger than the expression. (In this case, the generation of the random number changes depending on the value of the decimal portion of the argument.) If the expression is negative, the previously set numeric expression is used to generate the random number.

	Result	
Argument	Lower Bound	Upper Bound
.5	0<	<1
2	1	2
2.5	1	3

The same sequence of random numbers is normally generated because the same "seed" is used each time the power is turned on. To randomize the seed, see the RANDOMIZE command.

RUN

D

FORMAT: 1. RUN ENTER

2. RUN {line number / *label} ENTER

Abbreviation: R.
See Also: GOTO, ARUN

PURPOSE:
Executes a program in memory.

REMARKS:
Format 1 executes a program beginning with the lowest numbered statement in memory.
Format 2 executes a program beginning with the specified line number.

An error occurs if the specified line number or *label was not found.

If two or more identical labels exist in a program, the one with a smaller line number is executed.

SAVE

P
D

FORMAT: SAVE ["d:filename" [,A]]
d: E, PACOM, CAS, COM

Abbreviation: SA.
See Also: LOAD, DSKF, MERGE, FILES

PURPOSE:
Saves the basic program to the specified device.

REMARKS:
The SAVE statement names a BASIC program in memory and then writes it to the specified file.

A filename is the name given to a program or a set of data. The desired file can be readily retrieved by the Card if it is given a filename. A filename may consist of up to eight alphanumeric characters or symbols.

If all options are omitted, COM is assumed for the device name.

If the A option is specified, the file is saved in ASCII format, otherwise it is saved in intermediate code format. When the device name is PACOM, CAS, or COM, the file is saved in ASCII format even if the A option is not specified.

If no extension is specified, .BAS is assumed. The extension can consist of up to three characters.

An existing file will be erased if the same filename is specified, but an error occurs if the existing file has the "P" (write-protect) attribute set (see SET command).
An error occurs if the program in memory is made secret.

EXAMPLE:
SAVE "E:PRO1", A

Saves the program as an ASCII file with the name "PRO1" on RAM disk E.

SET

D

FORMAT SET "E:filename", {"P" / "␣"}

Abbreviation: SE.
See Also: SAVE

PURPOSE:
Assigns or removes file protection.

REMARKS:
The contents of a file cannot be inadvertently deleted or rewritten if "P" is specified. To clear the protection, specify a space ("␣"). Once protection has been removed, the file can be deleted or written to freely.

Wildcards (* or ?) can be used for filename specification, but the extension must not be omitted. For example, the .BAS extension must be specified.

An error occurs if the SET command is used with a file that is open.

EXAMPLE:
SET "E:PAYRUN.BAS","P"

Protects the program "PAYRUN.BAS" on the RAM disk E from being written to, erased, or renamed.

STOP

P

FORMAT: STOP

Abbreviation: S.
See Also: CONT

PURPOSE:
Halts execution of a program for diagnostic purposes.

REMARKS:
When STOP is encountered in program execution, execution halts and a message such as "Break in 200" is displayed where 200 is the number of the line containing the STOP. STOP is used during the development of a program to check the flow of the program or to examine the state of variables. Execution may be restarted with the CONT command or [SHIFT] [V] keys. Pressing the [V] key executes the program statement-by-statement.

STR$

P
D

FORMAT: STR$ expression

Abbreviation: STR.
See Also: VAL

PURPOSE:
Converts numeric data into string data.

REMARKS:
The STR$ function changes numeric data to a string. The string will be composed of the same digits as the original number. The STR$ function has the opposite effect of the VAL function.

If the numeric data is negative, the string will be preceded by a minus (–) sign.

When a numerical value is converted into a character string using the STR$ command, the first character is the sign (space for +) e.g. B$="␣12.3456".

TEXT

D

FORMAT: TEXT [ENTER]

Abbreviation: TE.
See Also: BASIC

PURPOSE:
Sets the Text mode.

REMARKS:
Valid only as direct input in the PRO mode.

The text function is used when entering a program written for a higher-level personal computer. The program entered using the Card can be sent to the host through the serial I/O interface.

Executing the TEXT command sets the Text mode. In the Text mode, a number corresponding to the line number, and then information corresponding to program commands or data is entered. Press the [ENTER] key to write the entries to the program/data area.
The written contents are not converted to commands (internal codes), as they are in the BASIC mode. The text is stored as it is (as characters and/or numbers) in character codes. The text is arranged in the order of the numbers corresponding to the line number at the beginning of each line. (Line number editing function.)

The text written in the Text mode is stored as it is. Therefore, command abbreviations in BASIC (such as I. for INPUT) are displayed and stored as such.

If a program is stored in the internal code of the Card with the text mode set, it is converted to character code.

During program conversion, "**" is displayed at the right end of the display unit.
The prompt symbol is "<" in the Text mode. (It is usually ">".)
If a password has been set, an error occurs when the TEXT command is executed.

TIME$

P
D

FORMAT: TIME$

Abbreviation: TI.
See Also: DATE$

PURPOSE:
Recalls the time currently set in the Organizer.

REMARKS:
TIME$ command recalls the time currently set in the Organizer, in the form of 5 character string data. The time is displayed in the order of hour and minute on a 24-hour basis. A colon (:) is used to separate the hour digits from the minute digits.
For details on setting the time, see the Organizer manual.

EXAMPLE:
```
10: PRINT TIME$
20: A$ = TIME$
30: PRINT A$
```

TROFF

P
D

FORMAT: TROFF

Abbreviation: TROF.
See Also: TRON

PURPOSE:
Cancels trace (TRON) mode.

REMARKS:
Execution of TROFF restores normal execution of the program.

TRON

P
D

FORMAT: TRON

Abbreviation: TR.
See Also: TROFF

PURPOSE:
Starts the trace mode.

REMARKS:
The trace mode provides assistance in debugging programs. When the trace mode is on, the line number of each statement is displayed after each statement is executed. To stop trace execution, press the [ON] key or execute the STOP command. After trace execution is stopped, the Card waits for the [▽] key to be pressed before moving on to the next statement. The trace mode continues until TROFF is executed, or the [SHIFT] [C•CE] keys are pressed.

USING

P
D

FORMAT: 1. USING format string
2. USING

Abbreviation: U.
See Also: LPRINT, PAUSE, PRINT

PURPOSE:
Controls the format of displayed or printed output.

REMARKS:
USING can be used by itself or as a clause within a PRINT, LPRINT, or PAUSE statement. When the USING command is used in a PRINT, LPRINT, or PAUSE statement, it is valid only for the values or strings output by that statement. If it is used independently (on an independent line), it is valid for all the subsequent PRINT or LPRINT commands. USING establishes a specified format for output that is used for all output that follows until changed by another USING.

#: Right justified numeric field character.
Length of integer field: 2 to 21 (including sign)
If a value is shorter than the specified numeric field, the extra portion of the field is filled with spaces.
If a numeric field with a length of 22 or more digits is specified, it is regarded to be 21 digits long.
Length of decimal field: 0 to 20 (0 to 19 for exponential numbers)
If a value is shorter than the specified field, zeros appear in the extra portion of the field. If the former is longer than the latter, the extra digits are truncated.

.: Decimal point (delimiter for integer and decimal parts)

,: Used as a 3-digit separator in numeric fields.
To separate every 3 digits of integer field with commas (,), place a comma in or at the end of the integer field.

^: Used to indicate that numbers should be displayed in scientific notation.
With this notation, the length of the mantissa field is always 2 (1 digit and the sign), without regard to the specified length of the integer field. If the given length of the decimal field is 19 or more digits, the length of the decimal field of the mantissa is also 19 digits.

&: Left justified alphanumeric field
If a string is shorter than the specified field, spaces appear in the extra portion of the field. If the former is longer than the latter, the extra characters are dropped.

(1) USING"###"
Prints the sign and 2 integer digits.

(2) USING"###."
Prints the sign, 2 integer digits, and a decimal point.

(3) USING"###.##"
Prints the sign, 2 integer digits, a decimal point, and 2 decimal places.

(4) USING"###,###."
Prints the sign, 4 integer digits, a 3-digit separator (,) and a decimal point.
For numerical data, 3-digit separator (,) is counted as a digit. So if you want to print a number "–1,234,567.", you have to use ten field characters (#), such as USING"#######,###.".

(5) USING"##.##^"
Prints numerical data in exponential form with up to 2 decimal places.
Spaces for 1 integer digit and the sign are automatically reserved for the mantissa, and for 2 integer digits, the capital E or D, and the sign for the exponent.
Note: ^ and comma (,) may not be used concurrently.

(6) USING"&&&&&&"
Prints a string of 6 characters.

(7) USING"###&&&&"
Prints a string adjacent to a numeric value.

(8) USING
Format 2 clears formatting.

Formatting is also cleared by executing the RUN command, pressing SHIFT C•CE, or turning the power off and then on.

EXAMPLE:

```
10: B=–10:C=10.7703
20: PRINT USING "&&&###" ; "B=" ; B ; "␣C=" ;: PRINT USING "###.###"; C
```

Note:
The USING command is not valid for manual calculations. It must always be used in the PRINT or LPRINT statement.

Supplement:
A program which simultaneously outputs numerical and string characters written for other computers should be modified as follows:

```
PRINT USING "####.##" ; H ; "(m)"
          ↓
PRINT USING "####.##" ; H ;: PRINT "(m)"
```

VAL

P
D

FORMAT: VAL string

Abbreviation: V.
See Also: STR$

PURPOSE:
Converts a string of numeric characters into a decimal value.

REMARKS:
The VAL function converts a character string, which may include the hex number designator (&H), numbers (0–9), a sign (+, –), and exponential symbols (E or D), into a numeric value.

If the string is in decimal notation, it must be composed of the characters 0 to 9, with an optional decimal point and sign. In this form, VAL is the opposite of the STR$ function.

If illegal characters are included, conversion is performed up to the first occurrence of an illegal character.

Control codes (&H00 to &H1F) cannot be used.

EXAMPLE:

```
A=VAL"–120"     Assigns –120 to variable A.
B=VAL"3.2*4="   Assigns 3.2 to variable B.
C=VAL"&H64"     Assigns 100 to variable C.
```


WAIT

P
D

FORMAT: 1. WAIT expression
2. WAIT

Abbreviation: W.
See Also: PRINT, GPRINT

PURPOSE:
Controls the length of time that displayed information is shown before program execution continues.

REMARKS:
Format 1 specifies the time in which execution of the PRINT command halts. The program temporarily halts for the specified time interval, then automatically restarts.

The value of the expression may be set to any value from 0 to 65535. A value of 1 as the expression corresponds to an interval of approx. 1/59 sec. The power-on default for the value of the expression is zero. The WAIT command is valid for all the PRINT or GPRINT commands used in the program. To set an infinite interval, use format 2.

Note:
The WAIT command is not available on personal computers in general. On PCs, the FOR...NEXT statement is used for wait time control as follows:

```
50: FOR J=1 TO 500:NEXT J
```

WIDTH

P
D

FORMAT: WIDTH {,4 / ,8}

Abbreviation: WI.
See Also: LOCATE

PURPOSE:
Switches between 4-line and 8-line display modes.

REMARKS:
WIDTH,4 selects 12 column by 4 line mode; WIDTH,8 selects 16 column by 8 line mode. Once the display mode is selected with this command, it remains valid until the next WIDTH command. The displayable character size depends on the selected display mode.

Execution of the WIDTH command returns the cursor to the home position.

If the WIDTH command is executed as direct input, the display is cleared. However, if it is executed in a program, the display is not cleared. Combining the WIDTH command with the LOCATE command allows for mixed display of large and small sized characters.

The 8-line mode is selected just after the power is turned off and on or the STAT or AER mode is entered.

Pressing [SHIFT] [4⇔8 LINES] will also switch between 4-line and 8-line display modes.

EXAMPLE:
```
10: CLS :WAIT 100
20: WIDTH ,4:PRINT "SHARP"
30: WIDTH ,8:LOCATE 8,1
40: PRINT "BASIC":LOCATE 9,2:PRINT "CARD"
50: WIDTH ,4:LOCATE 1,2
60: PRINT "Organizer"
70: WIDTH ,8:LOCATE 0,7
80: END
```


PART 4

APPENDICES

APPENDIX A

ERROR MESSAGES

When an error occurs, one of the error messages listed below will be displayed. For errors which occur during program execution, the error message is followed by the line number in which the error occurred. The error number and the line number are stored into the variables ERN and ERL, respectively. Errors which occur during direct input operation do not change the values of the variables ERN and ERL.

Error message	Error No.	Meaning
Syntax error	10	Invalid expressions or statements have been used.
Direct command error	11	An attempt was made to execute a command which is illegal in direct input operation. An attempt was made to execute a command which is illegal in program execution.
Mode error	12	The mode for PRO or RUN was selected incorrectly. The mode selection in the OPEN statement was incorrect.
Can't continue	13	The CONT statement was executed illegally.
Program not exist	14	An attempt was made to designate a password to a program which does not exist.
Overflow	20	The calculated result exceeds the calculation range.
Division by Zero	21	An attempt was made to divide by zero.
Illegal function call	22	An illegal operation was attempted.
Duplicate Definition	30	An attempt was made to declare an array variable name which is already declared.
Array specified without DIM	31	The array variable name was specified without the DIM statement.
Subscript out of range	32	Array was addressed illegally (array subscript exceeds the size of the array specified in the DIM statement)

Error message	Error No.	Meaning
Data out of range	33	The specified value exceeds the allowable range.
Undefined line	40	The specified line number or label does not exist.
Illegal line number	41	The line number was specified illegally.
Bad line number	44	The ending line number was specified with a number less than the starting line number in a statement such as LLIST or DELETE.
GOSUB or FOR nesting exceeded	50	The levels of nesting in the GOSUB or FOR statement exceeds the allowable range.
RETURN without GOSUB	51	An attempt was made to execute the RETURN statement without calling the subroutine.
NEXT without FOR	52	The FOR statement is missing for the NEXT statement.
Out of data	53	The DATA statement is missing for the READ statement.
Buffer space exceeded	54	The size of the BASIC interpreter exceeds the available work area.
String too long	55	The length of the entered string exceeds 254 bytes.
Line buffer overflow	56	The line exceeds 254 bytes.
RESUME without error	57	An attempt was made to execute the RESUME statement during non-error processing.
Out of memory	60	The size of program or variable exceeds the memory capacity.
Can't print in specified format	70	Characters cannot be printed nor displayed in the format specified in the USING statement.
USING format error	71	The format specified in the USING statement is illegal.
I/O error	72	I/O device error.
Too many files open	73	The number of files to be opened exceeds the limit.

Error message	Error No.	Meaning
NAME error	74	The file name specified in the NAME statement is illegal.
Bad drive name	75	The specified drive name is illegal.
File write protected	76	The file is write protected.
Disk full	77	No further memory storage available on the disk, or the number of files exceeds the limit.
Tape read error	80	An error occurred while reading data from the cassette tape recorder.
Verify error	82	Error in data verification.
Printer error	84	Printer error (The printer was turned off while printing, etc.)
File not open	85	The file has not been opened.
File already open	86	The file has already been opened.
Input past end	87	An attempt was made to read data past the end of file.
Type mismatch	90	The type of the specified data does not match.
Password mismatch	92	An invalid password was entered.
Invisible program	93	An attempt was made to write to a protected program.
File not found	94	The specified file does not exist.
Bad file name	95	The specified file name is illegal.

APPENDIX B

CHARACTER CODE CHART

The character code chart shows the characters and their character codes used by the CHR$ and ASC commands. Each character code consists of 2 hex characters (or 8 binary bits). The most significant hex character (4 bits) is shown along the top of the chart and the least significant hex character (4 bits) is shown down the left side of the chart. If no character is shown, it is an illegal character on the Card.

For example, the character "A" is hex 41 or decimal 65 or binary 01000001. The character "P" is decimal 80 or hex 50 or binary 01010000.

The character codes are represented as follows:

Examples:

Code for *

Hexadecimal	&H2A	
Decimal	42	(32 + 10)

Code for P

Hexadecimal	&H50
Decimal	80

Most Significant 4 Bits

Least Significant 4 Bits

H / L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	◀	*1/*2	space	0	@	P	‘	p	Ç	É	á	Ä	Ä	0	α	≡
1	*1/11	*2/1	!	1	A	Q	a	q	ü	æ	í	È	P	1	β	±
2	*1/13	*2/2	"	2	B	R	b	r	é	Æ	ó	Ö	▲	2	Γ	≥
3	*1/15	*2/3	#	3	C	S	c	s	â	ô	ú	Å	☃	3	π	≤
4	*1/17	*2/4	$	4	D	T	d	t	ä	ö	ñ	Í	Y	4	Σ	⌠
5	*1/19	*2/5	%	5	E	U	e	u	à	ò	Ñ	Ú	✓	5	σ	⌡
6	*1/21	*2/6	&	6	F	V	f	v	ª	û	º	Ó	┼	6	μ	÷
7	▶	*2/7	’	7	G	W	g	w	ç	ù	¿	Á	☠	7	τ	≈
8	*1/23	*2/8	(	8	H	X	h	x	ê	ÿ	⌐	Ê	ij	8	Φ	°
9	↲	*2/9	)	9	I	Y	i	y	ë	Ö	¬	Ô	¨	9	Θ	∙
A	↖	*1/10	*	:	J	Z	j	z	è	Ü	Ã	õ	^	×	Ω	·
B	↗	*1/11	+	;	K	[	k	{	ï	¢	½	Õ	~	§	δ	√
C	▲	*1/12	,	<	L	\	l	¦	î	£	¼	¡	‾	‼	∞	n
D	▼	*1/13	-	=	M	]	m	}	ì	¥	¡	Ø	~	→	ø	2
E	_	▷	.	>	N	^	n	~	Ä	₧	«	Ł	▪	←	∈	■
F	●	■	/	?	O	_	o	Δ	Å	ƒ	»	ł	_	◇	∩	

*1/*2 *1 For 4-line display
*2 For 8-line display
Blank refers to a null.

Note:
Character codes &H00 to &H1F can be used only with the LPRINT command.

APPENDIX C

BATTERY REPLACEMENT

Battery Life
After a new battery has been installed in the Card, it should maintain the data stored in the Card for a period of approximately 2 years, at room temperatures (approx. 20°C [68°F]).

Be sure to replace the battery every 2 years.

Batteries Used:

Type:	Lithium
Model:	CR2016
Quantity:	1 Battery

Changing the Battery
Refer to Installing the Battery for details on changing the battery.

The life of the battery can be shortened by environmental factors, for example in locations where temperatures are unusually high or low. In such cases, you may have to change the battery before two years have passed.

Notes:
- Change the battery with the Card inserted into the Organizer. If the battery is changed without the Card inserted, all data stored in the Card will be lost.
- Before replacing the battery, important information should be backed up by writing it down on paper or storing it on cassette tape.
- As a reminder for the next battery replacement, write the date of battery replacement on the label on the back of the Card.

Battery Precautions
- **Keep the battery out of the reach of children.**
- When the battery becomes weak, remove it from the Card immediately. If a depleted battery is left in the Card for any length of time, it may leak and cause corrosion inside the Card.
- Do not dispose of the battery in a fire, as it may explode.

APPENDIX D

TROUBLESHOOTING

This appendix provides you with some hints on what to do when your Card does not do what you expect it to. You should try each of the following suggestions, one at a time, until you have corrected the problem. **(Refer to the Organizer Operation Manual.)**

1. If the display is too light or too dark,
   - adjust the contrast control.
2. If the power does not come on (nothing is displayed),
   - the batteries may be exhausted. Replace the Organizer operating batteries.
   - the Card lock switch might not be set to the LOCK position. Check that the lock is set to the LOCK position.
3. If the power does not turn off,
   - the Card is running a program using a command which takes a long time, such as BEEP or SAVE "CAS:". Press the [ON] key to interrupt program execution and then the [OFF] key.
     If the power is still not turned off, perform the following operation 4.
4. If the Card does not operate properly,
   - a peripheral device may have been connected or disconnected while the power was on, or there may have been an error during program execution, or the Organizer may have been subjected to strong electrical noise or shocks during use.
     Perform one of the following operations.

   (1) **Reset** (retaining the memory contents)
   Press the RESET switch with a ball-point pen or any other appropriate device and then the [OFF] [ON] keys. If the Organizer still operates improperly after clearing the error condition with this operation, there may be an error in programs or data entered. Follow (2) below, pressing the [Y] key to clear the Card memory contents.

   (2) **All Reset** (clearing the Card memory contents)
   Press the RESET switch while holding the [ON] key.
   Release the RESET switch before releasing the [ON] key.

When the RESET switch is released, the display shown at right will appear.

```
MAIN DATA
ALL CLEAR
OK (Y/N) ?
```

(1) Press the [N] key.
The display shown at right will appear.

```
S1:CARD DATA
ALL  CLEAR
OK  (Y/N) ?
```

Caution: Do not press the [Y] key. Pressing the [Y] key will clear all data in the Organizer.

(2) Press the [N] key.
   - To clear the Card memory, press the [Y] key.

(3) Reset the clock.
   - The clock data will be cleared as a result of the steps above.

APPENDIX E
SPECIFICATIONS

Model:	OZ-707 Scientific Computer Card [BASIC]
Processor:	8-bit CMOS CPU
Programming language:	BASIC
System ROM:	128 K bytes
Memory capacity:	System internal 5K bytes approx. Fixed variable area 312 bytes Program/data area 27597 bytes
Stack:	Total: 145 bytes (Subroutine: 4 bytes/stack FOR-NEXT: 21 bytes/stack)
Operators:	Addition, subtraction, multiplication, division, trigonometric and inverse trigonometric functions, logarithmic and exponential functions, angle conversion, square and square root, power, sign, absolute, integer, coordinate conversion, pi, etc.
Numeric precision:	10 digits (mantissa) + 2 digits (exponent) single-precision mode 20 digits (mantissa) + 2 digits (exponent) double-precision mode
Editing features:	Cursor left and right, line up and down, character insert, character delete
Memory protection:	Battery backup
Power supply:	3V ⎓ (DC) Lithium battery (CR2016) × 1
Operating time:	Approximately 2 years (at constant temperature of 20°C [68°F] — varies depending on the type of battery and use conditions)

Serial input/output features:

Standards:	Start-stop transmission (asynchronous) system Half/full duplex
Baud rates:	300, 600, 1200, 2400, 4800, 9600 baud (bps)
Parity bits:	Even, odd, or no parity
Data bits:	7 or 8 bits
Stop bit:	1 or 2 bits
Connectors used:	15-pin connector (for external equipment)
Output signal level:	CMOS level (4 to 6 volts)
Interfacing signals:	Inputs: RD, CS, CD Outputs: SD, RS, RR, ER Others: SG, FG, VC

Operating temperature:	0° – 40°C (32° – 104°F)
Dimensions:	54(W) × 85.5(D) × 2(H) mm 2-1/8(W) × 3-3/8"(D) × 3/32"(H)
Weight:	18 g (0.04 lb.) (with battery)
Accessories:	Soft case, one lithium battery, and Operation Manual.

APPENDIX F

CARE OF THE OZ-707

- Do not carry the Card in the back pocket of slacks or trousers. This may subject the Card to bending and damage it.
- Do not bend or twist the Card. Such mistreatment may make it impossible to insert the card into the Organizer, or it may cause the card to malfunction.
- Never touch the terminals of the Card — this may damage the card with static electricity or cause other problems. Also, never allow liquids or materials to touch the Card as they may cause it to malfunction.

Note:
Make sure to turn the power off by pressing the [OFF] key before installing or removing the optional IC card. If the power is not OFF during installation or removal, no key other than the RESET switch will function and data stored in memory may be lost.

COMMAND INDEX

INDEX

SHARP SERVICE CENTER ADDRESS

Sharp Electronics Corporation
1300 Naperville Drive
Romeoville, IL 60441
(708) 759-8555

To order Supplies or Accessories, contact your local SHARP Dealer/Retailer or in the U.S.A., only, contact THE SHARP ACCESSORIES AND SUPPLY CENTER at 1 (800) 642-2122.

Regional Sales Offices and Distribution Centers

Eastern: Sharp Plaza, Mahwah, New Jersey, 07430-2135
Phone: (201) 529-8200

Midwest: 1300 Naperville Drive, Romeoville, IL 60441
Phone: (708) 759-8555

Western: Sharp Plaza, 20600 South Alameda St., Carson, California 90810
Phone: (213) 637-9488

Electronic Organizer Limited Warranty

Sharp Electronics Corporation warrants to the first consumer purchaser, for a period of 1 year from the date of purchase, that this IC Card ("the Product") will be free from defective workmanship and materials, and agrees that it will, at its option, either repair the defect or replace the defective Product or part thereof at no charge to the purchaser for parts or for labor.

This warranty does not apply to any appearance items of the Product, any consumable items such as paper, ink ribbon, or batteries supplied with the Product, or to any equipment or any hardware, software, firmware, or peripheral other than the Product. This warranty does not apply to any Product the exterior of which has been damaged or defected, which has been subjected to misuse, abnormal service or handling, or which has been altered or modified in design, construction or interfacing.

In order to enforce the rights under this limited warranty, the purchaser should mail, ship, or carry the Product, together with proof of purchase, to a Sharp Service Center. To find out the location of the nearest Sharp Service Center, see the last page of this book.

The limited warranty described above is in addition to whatever implied warranties may be granted to purchasers by law. To the extent permitted by applicable law, ALL IMPLIED WARRANTIES INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR USE ARE LIMITED TO A PERIOD OF 1 YEAR FROM THE DATE OF PURCHASE. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you.

Neither the sales personnel of the seller nor any other person is authorized to make any warranties other than those described above, or to extend the duration of any warranties beyond the time period described above on behalf of Sharp Electronics Corporation.

The warranties described above shall be the sole and exclusive remedy available to the purchaser. Correction of defects, in the manner and for the period of time described above, shall constitute complete fulfillment of all liabilities and responsibilities of Sharp Electronics Corporation to the purchaser with respect to the Product, and shall constitute full satisfaction of all claims, whether based on contract, negligence, strict liability or otherwise. In no event shall Sharp Electronics Corporation be liable, or in any way responsible, for any damages or defects in the Product which were caused by repairs or attempted repairs performed by anyone other than a Sharp Service Center technician. Nor shall Sharp Electronics Corporation be liable or in any way responsible for any incidental or consequential economic or property damage. Some states do not allow exclusion of incidental or consequential damages, so the above exclusion may not apply to you.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

SHARP®

SHARP ELECTRONICS CORPORATION

Sharp Plaza, Mahwah, New Jersey, 07430-2135

PRINTED IN JAPAN

0A5.5KS(TINSE5319ECZZ)①