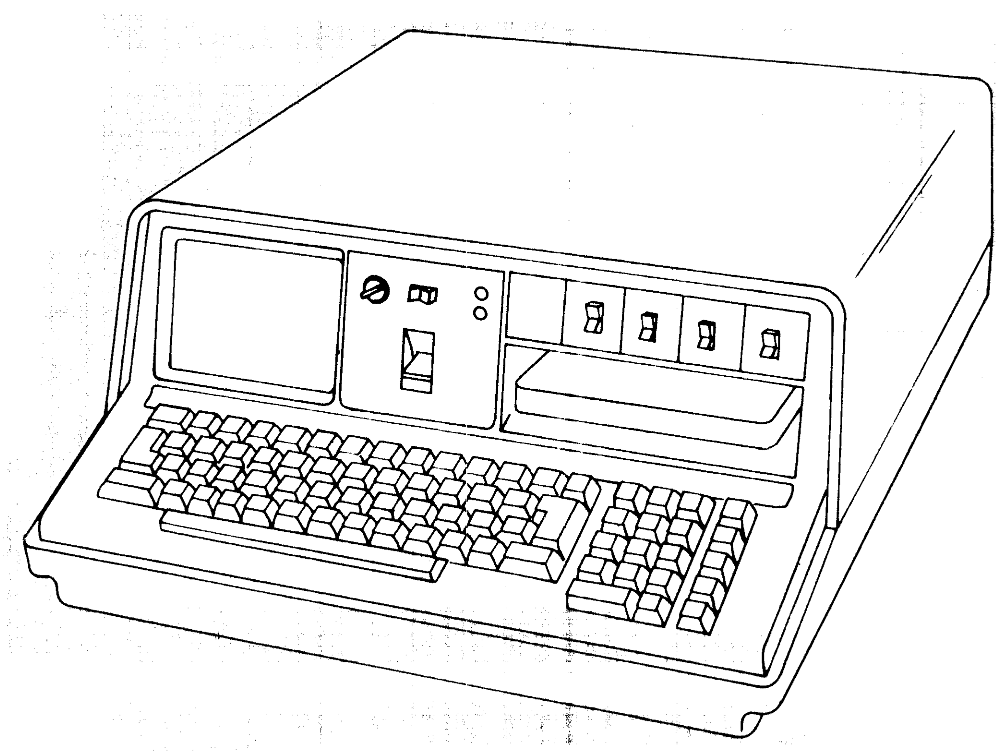
IBM

IBM 5110
APL Reference Manual

5110



IBM 5110
APL Reference Manual

SA21-9303-0

Preface

This publication is a reference manual that provides specific information about the use of the IBM 5110 Computer and the APL language. It also provides information about forms insertion and ribbon replacement for the 5103 Printer. This publication is intended for users of the 5110 and the APL language.

Prerequisite Publication

IBM 5110 APL Introduction, SA21-9301

Related Publications

- *IBM 5110 APL Reference Card,* GX21-9304
- *APL Language,* GC26-3847
- *IBM 5110 APL User's Guide,* SA21-9302
- *IBM 5110 Customer Support Functions Reference Manual,* SA21-9311

First Edition (December 1977)

Changes are continually made to the specifications herein; any such changes will be reported in subsequent revisions or technical newsletters.

Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A Reader's Comment Form is at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

IBM

PUBLICATION DRAWING

Part number	4836395
Date	**Change number**
1 Nov 77	832459
10 Jan 78	835462
10 APR 78	835514
22 JUN 78	835515
13 JUL 78	836558

Base form number: SA21-9303

Title: IBM 5110
APL Reference Manual

NOTE: All items following the last row of asterisks are to be shipped.

Change number	Form/TNL	Description/Comments
******************	******************	**
832459 835462	SA21-9303-0	First Release
	SN21-0282	TNL to add information about 5110 compatibility with the 5100
835514	SN21-0284	TNL to make miscellaneous corrections and additions.
836558	SN21-0289	TNL to make miscellaneous corrections.

A

IBM

PUBLICATION DRAWING

Part number	4836790
Date	Change number
1 Nov 77	832459
22 JUN 78	836536

Base form number: SA21-9302

Title: IBM 5110
APL User's Guide

NOTE: All items following the last row of asterisks are to be shipped.

Change number	Form/TNL	Description/Comments
***************	***************	**
832459	SA21-9302-0	First Release
***************	***************	**
836536	SA21-9302-1	Major revision to clarify several operations

A

IBM

PUBLICATION DRAWING

Part number	4836394
Date	**Change number**
1 Nov 77	832459

Base form number: SA21-9301

Title: IBM 5110
APL Introduction

NOTE: All items following the last row of asterisks are to be shipped.

Change number	Form/TNL	Description/Comments
***************	**************	**
832459	SA21-9301-0	First Release

A

IBM 4 U

IBM Technical Newsletter

This Newsletter No. SN21-0289
Date 15 June 1978

Base Publication No. SA21-9303-0
File No. None

Previous Newsletters SN21-0279
SN21-0282
SN21-0284

IBM 5110
APL Reference Manual

This technical newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

41, 42
83 through 88
88.1, 88.2 (added to accommodate moved text)
129, 130

Changes to text and illustrations are indicated by a vertical line at the left of the change.

Summary of Amendments

This technical newsletter contains additional information about the (.) inner product operator and the following scalar functions:

- > greater than
- = equal to
- < less than
- ≥ greater than or equal to
- ≤ less than or equal to
- ≠ not equal to

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

 Printed in U.S.A.

IBM Technical Newsletter

This Newsletter No. SN21-0284
Date 28 April 1978

Base Publication No. SA21-9303-0
File No. None

Previous Newsletters SN21-0279
SN21-0282

IBM 5110
APL Reference Manual

This technical newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

43 through 46	225, 226
177 through 182	237 through 240
185, 186	240.1, 240.2 (text rearranged)
203 through 206	241, 242

Changes to text are indicated by a vertical line at the left of the change.

Summary of Amendments

Miscellaneous corrections and additions have been made to clarify several operations.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Technical Newsletter

This Newsletter No.	SN21-0282
Date	10 January 1978
Base Publication No.	SA21-9303-0
File No.	None
Previous Newsletters	SN21-0279

IBM 5110
APL Reference Manual

This technical newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

227, 228

Changes to text are indicated by a vertical line at the left of the change.

Summary of Amendments

Additional information about 5110 compatibility with the 5100 is included in this technical newsletter.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

IBM Technical Newsletter

This Newsletter No.	SN21-0279
Date	30 December 1977
Base Publication No.	SA21-9303-0
File No.	None
Previous Newsletters	None

IBM 5110
APL Reference Manual

This technical newsletter identifies operational characteristics of your 5110 that vary from those described in the *IBM 5110 APL Reference Manual*. Please refer to this list as you use the manual.

Page	If You Are Using	Note This Consideration
17	APL PROC file	Procedure files for APL must be generated by an APL program or a BASIC program. The BASIC SAVE command in DATA mode cannot be used.
43	Diskette Recovery Customer Support Function on Model A1	This is not available on a 16K machine.
54	)SORT	Before executing)SORT be sure that the printer is switched off. An alternative is to execute: 1) OUTSEL OFF 2) OUTSEL OUT Failure to do so will cause a machine error.
181	Serial I/O Feature or write exchange (type 2) tape file	If you are writing to Serial I/O or writing an exchange (type 2) tape file, the last record may not be written correctly. You should write an extra record at the end.
182	OUTF TYPE=A	If you are writing OUTF to a TYPE=A file, use file sizes of 2K or greater.
182, 185, 188	ADD to a mixed record file	Do not attempt to ADD to a type 15 (mixed record file) on diskette.
187	Shared Variables with diskette	Do not open diskette drive door during shared variable operations.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

Contents

Chapter 1. Operation

IBM 5110 COMPUTER OVERVIEW

The IBM 5110 Model 1 (Figure 1) is a computer. The IBM 5110 Model 2 differs from the Model 1 only in that the Model 2 does not have a built-in tape unit. The 5110 has a display screen, keyboard, switches, indicator lights, and an adapter for black and white TV monitors. The display screen and indicator lights communicate information to the user. The keyboard and switches allow the user to control the operations the system will perform.

Features available for the 5110 are an auxiliary tape unit (Model 1 only), printer, up to four diskette drives, a communication adapter, and a serial I/O adapter.

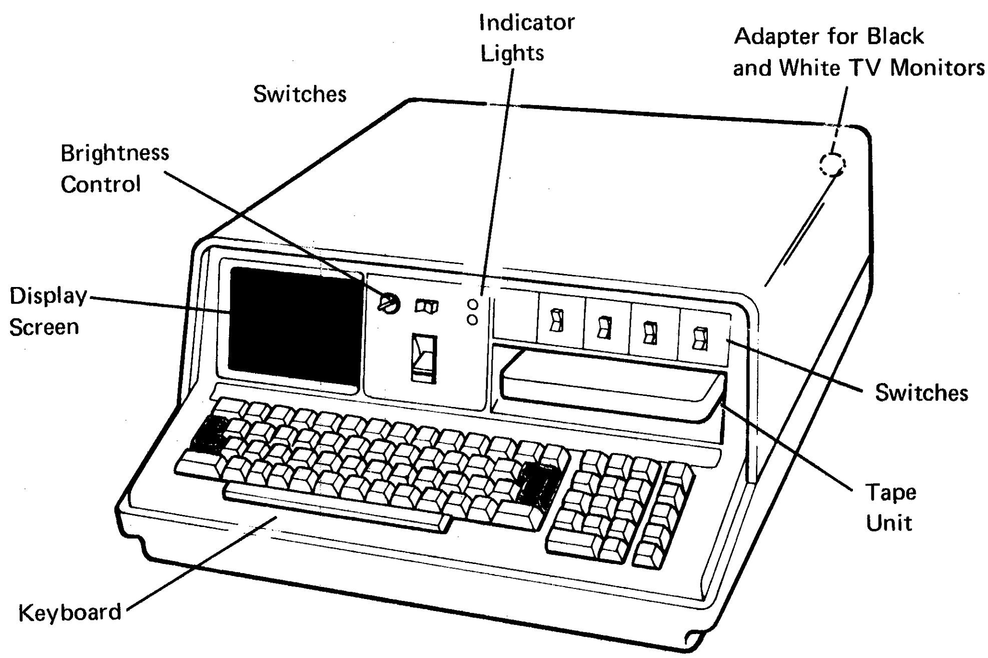


Figure 1. IBM 5110 Computer

DISPLAY SCREEN

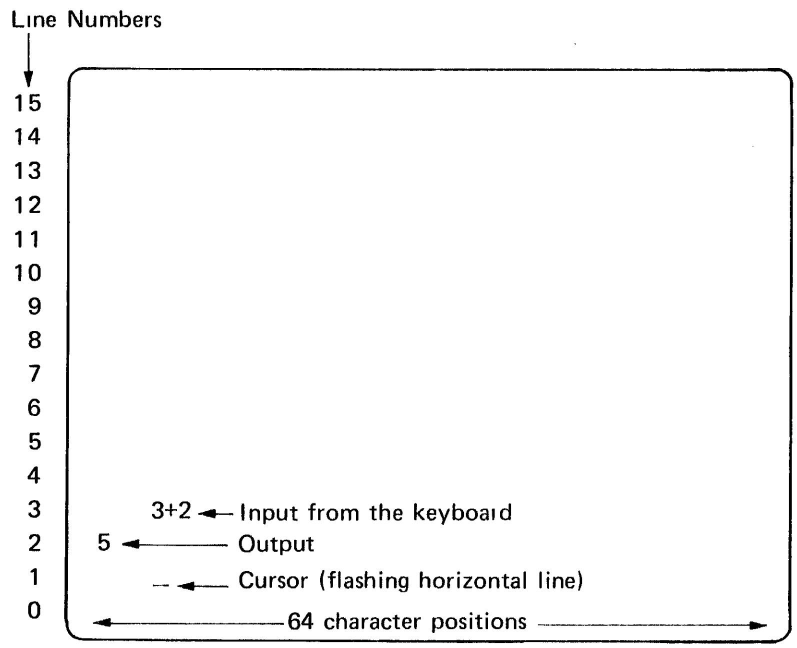
The display screen (Figure 2) can display 16 lines of information at a time, with up to 64 characters in each line. Input (information supplied by the user) as well as output (processed information) is displayed. The bottom two lines (lines 1 and 0) of the display contain information entered from the keyboard. The cursor (flashing horizontal line) indicates where the next input from the keyboard will be displayed. If the cursor is moved to a position that already contains a character, the flashing line is replaced by the flashing character. As the 5110 processes input, all lines of the display are moved up so that information can be entered on the two bottom lines again. The top lines of the display are lost as the lines are moved off the display screen.

SWITCHES

The switches on the 5110 (Figure 1) are used for turning power on, restarting the system, and controlling how information is displayed.

Power On or Restart Procedures

The following switches are used for turning power on to the system or restarting the system operation.



Normally, to distinguish input from output, input from the keyboard is indented and output is displayed starting at the left edge of the display screen.

Figure 2. The 5110 Display Screen

BASIC/APL

Only dual-language machines have this switch. The switch setting determines which language will be in operation when power is turned on or after RESTART is pressed. If the switch setting is changed after the power is turned on or after RESTART is pressed, the language in operation will not be changed.

Power ON/OFF

When this switch is in the ON position, power is supplied to the system. The system performs internal checks and becomes ready in 15 to 20 seconds. When the switch is put in the OFF position, no power is supplied to the system.

Note: The message CLEAR WS is displayed when the system becomes ready. If this message is not displayed after 20 seconds, restart the system operation, (the RESTART switch is discussed next).

RESTART

This switch restarts the system operation. When it is pressed, the system performs internal checks and becomes ready in 15 to 20 seconds. The message CLEAR WS is displayed when the system is ready. If the system does not display the message after 20 seconds, press RESTART again. If the system does not become ready after several attempts, call your service representative.

The primary uses of this switch are to restart the system operation after a system malfunction has occurred and to change the language in operation on dual-language machines.

Note: Any information you had stored in the active workspace will be lost when RESTART is pressed.

Display Screen Control

The following switches are used to control how the information on the display screen is displayed.

L32 64 R32

The three-position switch (positions 64, L32, and R32) operates as follows:

- 64–Characters are displayed in adjacent positions, and up to 64 characters can be shown on each line.
- L32–Characters are displayed in alternate positions (blanks between); only the left 32 characters of the 64-character lines are shown.
- R32–Characters are displayed in alternate positions (blanks between); only the right 32 characters of the 64-character lines are shown.

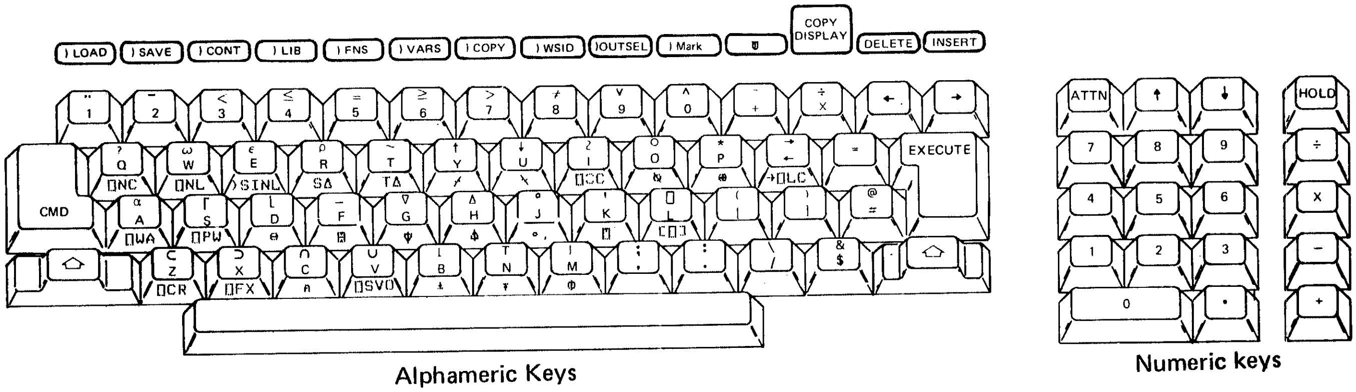


Figure 3. The 5110 Keyboard

REVERSE DISPLAY

This switch determines whether the display screen will display light characters on a dark background or dark characters on a light background. The brightness control may have to be adjusted when the switch setting is changed.

DISPLAY REGISTERS

This switch is for the service representative's use when servicing your 5110.

Note: When you use your 5110, this switch must be in the NORMAL position.

KEYBOARD

The 5110 keyboard (Figure 3) has alphameric and numeric keys. The alphameric keys are grouped together and are similar to those on a typewriter keyboard. When the keys are pressed, the characters entered appear in the input line (one of the bottom two lines) on the display screen. The top row of alphameric keys can be used to enter numbers; however, numbers can be conveniently entered via the numeric keys on the right side of the keyboard. The arithmetic symbols (+ - ÷ ×) located on the top row of the alphameric keyboard can also be entered via keys to the right of the numeric keys.

When the system power is first turned on, the system is in standard APL character mode. In standard APL character mode, you can enter uppercase alphabetic characters (without using the shift key), the APL symbols (using the shift key), and the APL system command keyword (above the top row of numeric keys) and special character combinations (engraved on the front of the keys) using the CMD key.

Note: For combined APL/BASIC systems, there is also a BASIC statement keyword engraved on the alphabetic keys. These BASIC statement keywords are above the special APL character combinations on the keys.

You can also place the system in lowercase alphabetic character mode by:

1. Pressing the HOLD key

2. Holding down the shift (⇧) key and pressing the scroll down (↓) key

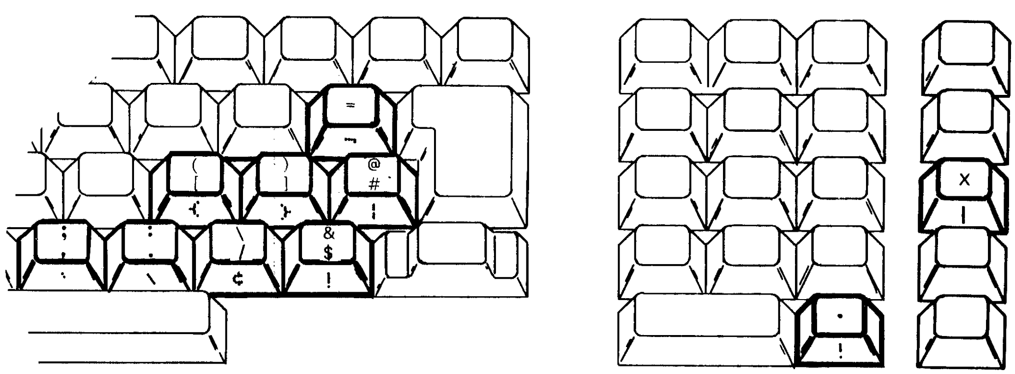
In lowercase alphabetic character mode, you can enter lowercase alphabetic characters (without using the shift key), uppercase alphabetic characters (using the the shift key), and the APL symbols using the CMD key. The lowercase alphabetic character mode applies only to the alphabetic keys. In lowercase alphabetic character mode, you cannot enter the special character combinations using the CMD key, because the CMD key is used to enter the APL symbols.

To return from lowercase alphabetic character mode to standard APL character mode:

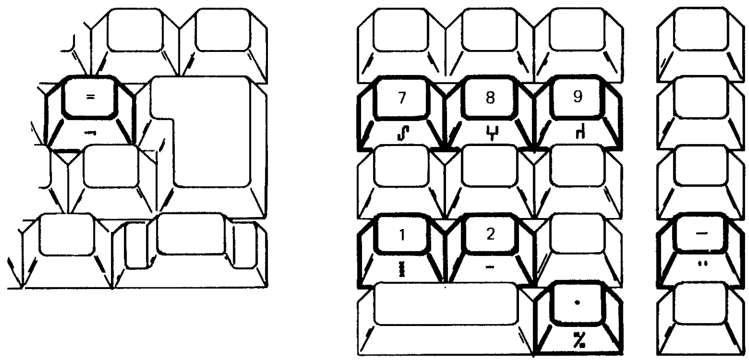
1. Press the HOLD key

2. Hold down the shift key while you press the scroll up (↑) key

The characters ⍨ ⊣{ }⊢ ¦ ` \ ¢ ! | ! and ⌐ ⍸ ⍽ ⍦ ⌶ ¨ '' % are not shown on the keyboard; however, you can enter the first group of these characters by using the CMD key and pressing the appropriate key as follows:



You can enter the second group of characters using the SHIFT key and pressing the appropriate key as follows:



Note: Some of these characters change when a unique national character set, other than EBCDIC, is specified. See Appendix C for more information on the unique national graphics.

The keyboard contains some keys that perform operations other than those performed by a typewriter. These keys are discussed in the following text. Uses of the APL language symbols on the keyboard are discussed in Chapter 4 of this manual.

Attention ATTN

Pressing ATTN (attention) when entering information from the keyboard erases everything from the cursor to the end of line 0.

Pressing ATTN when entering information using shared variables to the display screen erases everything from the cursor position to the end of the specified input area.

When ATTN is pressed (weak interrupt) during execution of any expression or user-defined function, the system stops operation at the end of the statement currently being processed. To restart the execution of a user-defined function, enter →⎕LC.

Output that was being generated before the system operation stopped may not be displayed because there is a delay between the execution of the statement that causes the output and the actual display of the output.

When the shift key is held and ATTN is pressed (strong interrupt) during the execution of a statement (either inside or outside a user-defined function), the execution of that statement stops as soon as possible. Also, the message INTERRUPT, the statement, and a caret (∧) that indicates where the statement was interrupted are displayed. In this case, the intermediate results might be lost, and restarting the function using →⎕LC might give incorrect results.

Hold

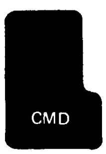
When pressed once, HOLD causes all processing to stop; when pressed again, it allows processing to resume. The primary purpose of HOLD is to permit reading rapidly changing display information during an output operation or to change from one character mode (standard APL or lowercase alphabetic) to the other character mode. When the HOLD is in effect (HOLD pressed once), the COPY DISPLAY, shift/scroll up, and shift/scroll down keys are active, and the characters HOLD are displayed in the lower left corner of the display screen.

Notes:
1. Holding down the CMD key and pressing HOLD is restricted to use by the service personnel.
2. When the hold is in effect (HOLD pressed once), holding down the CMD key and using the arithmetic keys (+ - ÷ ×) on the right side of the keyboard are restricted to use by service personnel.

Execute

When this key is pressed, the input line(s) of information on the display screen is processed by the system. This key must be pressed for any input to be processed.

Command



In standard APL character mode, when this key is pressed and held, pressing an alphameric key in the top row causes the APL command keyword or character above that key to be entered in the input line. Also, pressing the other key causes the special character combination engraved on the front of the key to be entered in the input line.

In lowercase alphabetic character mode, when this key is pressed and held, pressing an alphabetic key causes the APL symbol on the key to be entered in the input line.

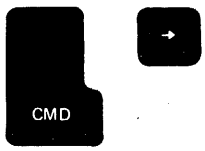
Note: Holding down the CMD key and pressing HOLD is restricted to use by the service personnel.

Positioning the Cursor and Information on the Display Screen

The following keys are used to position the cursor and information on the display screen.

Forward Space 

When this key is pressed once, the cursor moves one position to the right. When this key is held down, the cursor continues to move to the right. When the cursor reaches the last position on one input line (line 1 or 0), it wraps around to the first position on the other input line.

Insert 

When the CMD key is held down and the forward space key is pressed once, the characters at and to the right of the cursor position (flashing character) are moved to the right one position, and a blank character is inserted at the cursor position. The cursor does not move. For example:

Flashing Character

Before the insert operation: 123567

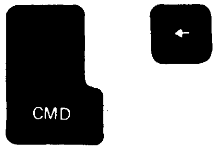
After the insert operation: 123_567

When these keys are both held down, the characters continue to move to the right and blank characters continue to be inserted.

Note: If there is a character in position 64 of line 0, the insert operation will not work.

Backspace 

When this key is pressed once, the cursor moves one position to the left. When it is held down, the cursor continues to move to the left. When the cursor reaches position 1 on one input line (line 1 or 0), it wraps around to the last position on the other input line.

Delete 

When the CMD key is held down and the backspace key is pressed once, the character at the cursor position (flashing character) is deleted and all characters to the right are moved over one position to the left to close up the space. The cursor is not moved. For example:

Before the delete operation: 1234456
Flashing Character
After the delete operation: 123456

When these keys are both held down, the characters at the cursor position continue to be deleted and all the characters to the right are moved to the left.

Scroll Up 

When this key (located above the numeric keys) is pressed once, each displayed line is moved up to the next line. As the lines are moved up, the top line is lost as it is moved off the display screen. When this key is held down, the lines continue to move up.

When you are doing input/output operations to the display screen using an APL shared variable, the cursor is moved up one line when this key is pressed. When this key is held down, the cursor continues to move up until the top line is reached.

When the system is in the hold state (the HOLD key has been pressed once), holding down the shift key and pressing this key causes the system to be in standard APL character mode.

Scroll Down 

When this key (located above the numeric keys) is pressed once, each displayed line is moved to the next lower line. As the lines are moved down, the bottom line is lost as it is moved off the display screen. When this key is held down, the lines continue to move down.

When you are doing input/output operations to the display screen using an APL shared variable, the cursor is moved down one line when this key is pressed. When this key is held down, the cursor continues to move down until the bottom line is reached.

When the system is in a hold state (the HOLD key has been pressed once), holding down the shift and pressing this key causes the system to be in lowercase alphabetic character mode.

Copy Display ÷ X

If there is a 5103 Printer, when the CMD key is held down and this key is pressed once, all the information presently on the display screen is printed. COPY DISPLAY is operational even when the system is in the hold state (the HOLD key has been pressed once).

Note: The L32 64 R32 switch has no effect on what will be printed.

INDICATOR LIGHTS

The 5110 (Figure 1) has two indicator lights.

Process Check

When on, the PROCESS CHECK light indicates that a system malfunction has occurred. In this case, press the RESTART switch to restart the system operation. If the system operation cannot be successfully restarted after several attempts, call your service representative. After the system operation is restarted, you must reload any user-defined functions or data that was previously in the active workspace.

In Process

The IN PROCESS light is on when the system is processing data and the display screen is turned off during the execution of a user-defined function (see the ⎕CC system function in Chapter 5). The light goes off when:

- The display screen is turned on using the ⎕CC system function
- A statement in the user-defined function generates results that are to be displayed
- Execution is completed and the cursor returns

Note: If the display screen is blank (no data or cursor is displayed) and the IN PROCESS light is off, check the brightness control before calling your service representative.

DATA REPRESENTATION

Numbers

The decimal digits 0 through 9 and the decimal point are used in the usual way. The character ¯, called the negative sign, is used to denote negative numbers. It appears as the leftmost character in the representation of any number whose value is less than zero:

```
          0-4
 ¯4
          ¯3-¯2
 ¯1
```

The negative sign, ¯, is distinct from - (the symbol used to denote subtraction) and can be used only as part of the numeric constant.

Scaled Representation (Scientific Notation)

You can represent numbers by stating a value in some convenient range, then multiplying it by the appropriate power of 10. This type of notation is called scaled representation in APL. The form of a scaled number is a number (multiplier) followed by E and then an integer (the scale) representing the appropriate power of 10. For example:

```
Number          Scaled Form

                ┌──────────Multiplier
                ▼
66700           6.67E4
                    ▲
                    └──────Scale
.00284          2.84E¯3
```

The E (E can be read *times ten to the*) in the middle indicates that this is scaled form; the digits to the right of the E indicate the number of places that the decimal point must be shifted. There can be no spaces between the E and the numbers on either side of it.

Numeric Value Range

Numeric values in the 5110 can range from ¯7.237005577332262E75 to 7.237005577332262E75. The smallest numeric value the 5110 can use is ±5.397604346934028E¯79.

Numeric Value Precision

Numbers in the 5110 are carried internally with a precision of 16 significant digits.

Note: As with all computers, some values cannot be expressed exactly, and certain computations are not fully accurate in the least significant (16th) digit.

Character Constants

Zero or more characters enclosed in single quotes, including overstruck characters (see Appendix B) and blank characters (spaces), is a character constant. The quotes indicate that the characters keyed do not represent numbers, variable names, or functions, but represent only themselves. When character constants are displayed, the enclosing quotes are not shown:

```
      'ABCDEFG'
ABCDEFG
      '123ABC'
123ABC
      M←'THE ANSWER IS:'
      M
THE ANSWER IS:
```

If the closing quote is not entered, the 5110 automatically supplies the closing quote when EXECUTE is pressed.

When a quote is required within the character constant, a pair of quotes must be entered to produce the single quote in the character constant. For example:

```
      'DON''T GIVE THE ANSWER AWAY'
DON'T GIVE THE ANSWER AWAY
```

Logical Data

Logical (Boolean) data consists of only ones and zeros. The relational functions (> ≥ = < ≤ ≠) generate logical data as their result; the result is 1 if the condition was true and 0 if the condition was false. The output can then be used as arguments to the logical functions (∧⍲∨⍱~) to check for certain conditions being true or false. Logical data can also be used with the arithmetic functions, in which case it is treated as numeric 1's and 0's.

STORAGE REQUIREMENTS

The following list shows how many bytes of storage are required for each data type:

Data Type	Number of Bytes Required
Character constant	1 byte per character
Whole numbers that are equal to or less than $2^{31}-1$	4 bytes
Whole numbers that are greater than $2^{31}-1$	8 bytes
Decimal numbers	8 bytes
Logical data	1/8 Byte (1 byte can contain 8 ones or zeros)
Variable Name	**Number of Bytes Required**
3 characters or less	12 bytes+(4×rank (ρ ρVARIABLE)) bytes
4 characters or more	1 byte per character+21 bytes+(4×rank (ρ ρVARIABLE)) bytes

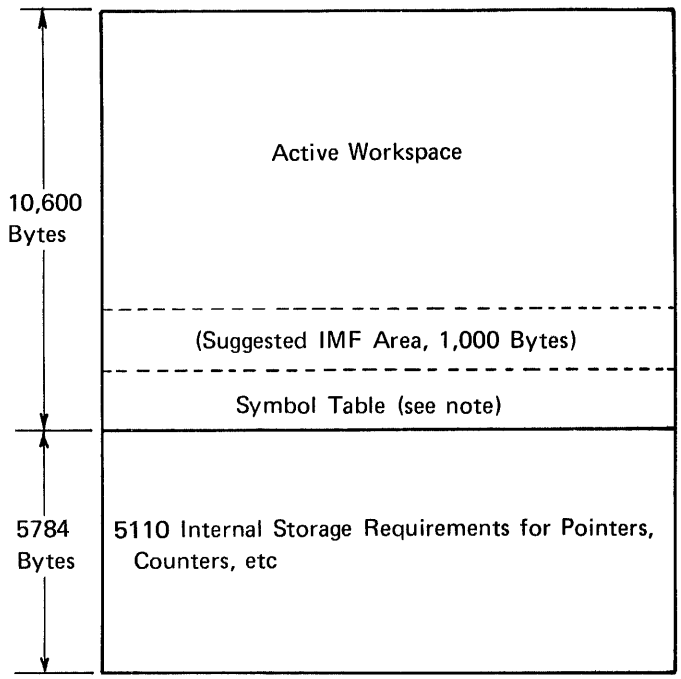
Note: All storage is allocated in 4 byte increments.

5110 STORAGE CAPACITY AND MODELS

The 5110 Models 1 and 2 can have a storage capacity of 16K (K = 1024 bytes), 32K, 48K, or 64K. The first digit in the model number indicates the model, and the second digit indicates the storage capacity as follows:

Storage Capacity	Model Number with Tape	Model Number without Tape
16K	A11	A21
32K	A12	A22
48K	A13	A23
64K	A14	A24

Figure 4 shows how the storage is allocated for the various requirements on a base 5110 (Model A11 or A21).



Note: The symbol table requires 8 bytes of storage for each symbol allowed in the active workspace (see)SYMBOLS in Chapter 3).

Figure 4. Storage Allocation for a Model A11 5110

Notice that the workspace available to the user (active workspace) is 10,600 bytes, while the remaining bytes are used for internal purposes. In the other models, all additional storage is allocated to the active workspace. For example, on the Model A14, the active workspace is approximately 60,000 bytes.

PERFORMANCE CONSIDERATIONS

Turning the Display Screen Off

The internal processing capability of the 5110 is increased by approximately 18% when the display screen is turned off during the execution of a user-defined function. The ⎕CC (console control) system function is used to turn the display screen on and off. For example:

X ← 1 ⎕CC 0	This statement turns the display screen off.
X ← 1 ⎕CC 1	This statement turns the display screen on again.

(See *The ⎕CC Function: Console Control* in Chapter 5 for a complete description of the ⎕CC function.)

Overlapped Printing

When data is sent to the printer, the 5110 continues (overlaps) internal processing as the last print line is being printed. For example:

```
      ∇FUNCTION ←——— When FUNCTION is executed, the
[1]  10 10⍴(⍳100)    5110 starts processing statement 2 as
[2]  (⍳50)*2∇        the last line of the matrix generated in
                     statement 1 is printed.
```

Therefore, to get maximum performance when data is sent to the printer, send the data to the printer in smaller segments. The following examples show how the data might be sent to the printer in smaller segments:

Example 1

```
      ∇ PRINT1;A;B;C;D
[1]      A←' COLUMN1 COLUMN2 COLUMN3 COLUMN4 COLUMN5'
[2]      B← 10 5 ⍴(⍳50)
[3]      C← 8 0 ⍕B
[4]      D← 8 0 ⍕(+/[1] B)
[5]      A,[1] C,[1] D
[6]   ∇
```

In this example, all the data is sent to the printer by statement 5. The overlapped printing capability is not used.

Example 2

```
      ∇ PRINT2;B;C;D
[1]      ' COLUMN1 COLUMN2 COLUMN3 COLUMN4 COLUMN5'
[2]      C← 8 0 ⍕B← 10 5 ⍴(⍳50)
[3]      D← 8 0 ⍕(+/[1] B)
[4]      C,[1] D
[5]   ∇
```

In this example, statement 2 is processed as the data from statement 1 is printed.

Example 3

```
      ∇ PRINT3;B;C
[1]      ' COLUMN1 COLUMN2 COLUMN3 COLUMN4 COLUMN5'
[2]      8 0 ⍕B← 10 5 ⍴(⍳50)
[3]      8 0 ⍕+/[1] B
[4]   ∇
```

In this example, statement 2 is processed as the data from statement 1 is printed, and statement 3 is processed as the last line of data from statement 2 is printed.

VARIABLE AND FILE NAMES

A variable name can be up to 77 characters in length with no blanks; the first character must be uppercase alphabetic or underscored uppercase alphabetic, and the remaining characters can be any combination of uppercase alphabetic, underscored uppercase alphabetic, and numeric characters. Variable names longer than 77 characters can be used, but only the first 77 characters are significant to the 5110.

Note: Do not use S∆ or T∆ as the first 2 characters in the variable name (see *Trace and Stop Controls* in Chapter 6).

The data file names that are allowed by the 5110 depends upon the type of file and storage media used:

- *Workspace file on tape.* The file name can be any combination of up to 11 alphabetic and numeric characters (with no blanks); the first character must be alphabetic. For example:

    FILE2

  If more than 11 characters are entered, only the first 11 are used.

- *Data file on tape.* The file name can be any combination of up to 17 characters (including blanks). For example:

    2 DATAFILE

  If more that 17 characters (including blanks) are entered, an INVALID PARAMETER error message is displayed.

- *Workspace file on diskette.* The file name can be any *simple* name. A simple name is any combination of up to 8 alphabetic and numeric characters (with no blanks); the first character must be alphabetic. For example:

    FILEONE

  If more than 8 characters are entered as the file name, an INVALID DISK FILE NAME message is displayed.

- *Data file on diskette.* The file name can be a simple or *complex* name. A complex name is two or more simple names, with each name separated by a period(.). A complex name cannot exceed 17 characters including the period. For example:

```
DATA.FILE
    ↖————————————A period must separate two simple names.
FILENAME ←———————For one simple name, a period is not required.
FILE.NAME←———————Complex names.
DISKETTE.FILENAME
```

Note: The names SYSAREA and ERRORSET are used by the system. You should not use these names as diskette file names.

NAME PROCESSING

Following is a description of how the system uses the device/file number and file ID parameters specified for system commands (see Chapter 3) or input/output operations (see Chapter 7).

For operations to tape files:

- The device/file number parameter must always be specified.
- The file ID parameter is optional, except for the)DROP command. When this parameter is specified, the specified file ID is checked against the tape file ID, and the operation fails if the file IDs do not match.

For operations to diskette files:

- If only the device/file number parameter is specified, the operation is performed to the specified diskette file.
- If only the file ID parameter is specified, the diskette files on each diskette drive are checked for the specified file ID. The specified file ID must be unique on the diskette(s), and the operation is performed to the appropriate diskette file when a match is found.
- If both the device/file number and file ID parameter are specified, the specified file ID is checked against the specified diskette file ID, and the operation fails if the file IDs do not match.

  Note: If the specified file number is 000 (for example, 11000 is diskette drive 1, file number 000), the diskette files on the specified diskette drive are checked for the specified file ID, which must be unique on the diskette. The operation is performed to the appropriate diskette file when a match is found.

PROCEDURE FILE

A procedure file provides an alternative to entering input from the keyboard. A procedure file can contain any executable expression or statement that can be entered from the keyboard; for example, system commands, function definition, and APL expressions. When the)PROC command is issued (see Chapter 3), these expressions or statements from the procedure file replace the input from the keyboard.

Creating a Procedure File

A procedure file can be a sequential access type I (file type 2) or direct access type I or U (file type 9) data file. Each record in the procedure file represents one statement or expression to be executed by the 5110. These records *cannot* be greater than 128 characters. See Chapter 7 for more information on creating data files.

Using a Procedure File

The)PROC command is used to initiate input from the procedure file. Once the)PROC command is issued, the records (expressions or statements) in the specified procedure file are sequentially read, displayed, and executed. The results of the executed statements or expressions are also displayed. When records are executed from a procedure file, except for direct access input from the display screen (see *Entering Data from the Keyboard* later in this chapter), all keyboard equivalent input comes from the procedure file. This includes the response for ⎕ input ⍞ input, and the ALREADY MARKED message. For example:

```
        ADD
⎕:
3+4
7
```

ADD ← This user-defined function is executed from a procedure file.

The function requests ⎕ input.

The response for the ⎕ input must be the next record in the procedure file.

If an error occurs on a statement or expression executed from the procedure file, an error message is displayed, and the next record on the procedure file is read and executed. The 5110 continues to read and execute input from the procedure file until one of the following occurs:

- The last record is read from the procedure file.
- A strong interrupt is entered (hold down the shift key and press the ATTN key).
- Another)PROC command is issued. In this case, the input then comes from the new procedure file, and control is *not* returned to the previous procedure file.
- An I/O error occurs when the 5110 is reading the procedure file.
- The)LINK command is issued from the procedure file.

Except when another)PROC command is issued, the 5110 returns to using keyboard input when one of the previous conditions occurs.

Doing Input/Output Operations

When using a procedure file, you can use shared variable to do input/output operations such as:

- Read records from and write records to a data file.

  Note: If the procedure file is a sequential access tape data file, the tape is repositioned if input/output operations are performed on the same tape that the procedure file is on. This causes an error when the next physical read of data from the procedure file occurs. However, since each physical read from the tape procedure file consists of 512 characters, a tape procedure file with less than 512 characters can also be used to perform input/output operations. In this case, the last command in the 512 characters should be)PROC to return the 5110 to keyboard entry.

- Read records from and write records to the display screen. This allows you to enter data from the keyboard when using a procedure file.

- Read records from and write records to the procedure file. This allows you to update the procedure file as the records are read from the procedure file and executed.

- Send data to the printer.

See Chapter 7 for more information on these input/output operations. Entering data from the keyboard and updating the procedure file is discussed next.

Entering Data from the Keyboard

Even though the records from the procedure file replace keyboard input, you can enter data from the keyboard using a pair of shared variables and direct access to the display screen. For example, assume you are using a procedure file and the following user-defined function has already been copied into the active workspace:

```
      ∇EXAMPLEENTER[⎕]∇
    ∇ EXAMPLEENTER;CTL;DAT;A
[1]   ⍝**************************************************
[2]   ⍝OFFER VARIABLE NAMES TO BE SHARED
[3]    A←1 ⎕SVO 2 3 ⍴'CTLDAT'
[4]    →(∨/(2≠A))/ERROR1
[5]   ⍝**************************************************
[6]   ⍝SPECIFY INPUT/OUTPUT OPERATIONS TO THE DISPLAY
[7]    CTL←'DISPLAY'
[8]    →(0≠1↑CTL)/ERROR2
[9]    'ENTER NEW DATA'
[10]   CTL← 0 903 120
[11]   →(0≠1↑CTL)/ERROR2
[12]  ⍝**************************************************
[13]  ⍝DISPLAY THE INFORMATION FROM THE KEYBOARD
[14]   DAT
[15]   →0
[16]  ⍝**************************************************
[17]  ⍝ERROR MESSAGES
[18]  ERROR1:'THE SHARED VARIABLE OFFER FAILED'
[19]   →0
[20]  ERROR2:'ENTERING DATA FROM THE KEYBOARD FAILED'
    ∇
```

When this statement is executed, the cursor appears at character position 903 on the display screen. Now, you can enter 120 characters from the keyboard.

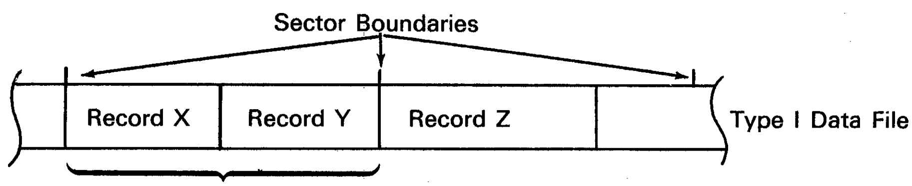
After the data is entered from the keyboard and EXECUTE is pressed, the 120 characters starting from position 903 are assigned to DAT.

Now when the record EXAMPLEENTER is read from the procedure file and executed, you will be able to enter data from the keyboard.

Updating a Procedure File on Diskette

When using a procedure file that is on a diskette, you can also do direct access input/output operations to the procedure file. This allows you to update records (statements or expressions) before they are read and executed.

If a procedure file is to be updated, it is recommended that the procedure file be a type U data file, because a type U data file has only one record per sector and the 5110 reads one sector at a time when executing records from a procedure file. Type I data files might have more than one record per sector, and the records can cross sector boundaries. Therefore, when a sector is read, the data read might include more than one record. For example:



When this sector is read, both record X and record Y are read into the system.

In the previous example, if record X executes a user-defined function that updates record Y (see the following example), record Y has already been read into the system and the updated record is not executed from the procedure file. Instead, the original record Y is executed.

Following is an example of a user-defined function that allows you to enter information from the keyboard and to update the procedure file with the information you entered.

In this example, assume that you are using a procedure file, and that the user-defined function has already been copied into the active workspace.

```
      ∇EXAMPLEWRITE[⎕]∇
    ∇ EXAMPLEWRITE;CTLX;CTLY;DATX;DATY;A;B
[1]   ⍝************************************************************
[2]   ⍝OFFER THE VARIABLE NAMES TO BE SHARED
[3]    A←1 ⎕SVO 4 4 ⍴'CTLXDATXCTLYDATY'
[4]    →(∨/(2≠A))/ERROR1
[5]   ⍝************************************************************
[6]   ⍝SPECIFY THE INPUT/OUTPUT OPERATIONS TO THE DISPLAY AND
[7]   ⍝THE PROCEDURE FILE
[8]    CTLX←'DISPLAY'
[9]    →(0≠1↑CTLX)/ERROR2
[10]   CTLY←'IOR 11001'
[11]   →(0≠1↑CTLY)/ERROR2
[12]  ⍝************************************************************
[13]  ⍝ENTER DATA FROM THE KEYBOARD
[14]   'ENTER THE RECORD NUMBER TO BE UPDATED'
[15]   CTLX← 0 903 3
[16]   →(0≠1↑CTLX)/ERROR2
[17]   B←DATX
[18]   'ENTER THE NEW RECORD'
[19]   CTLX← 0 896 127
[20]   →(0≠1↑CTLX)/ERROR2
[21]   CTLX←⍳0
[22]   DATY←128↑DATX
[23]  ⍝************************************************************
[24]  ⍝UPDATE THE PROCEDURE FILE
[25]   CTLY←1,(⍕B),1
[26]   →(0≠1↑CTLY)/ERROR3
[27]   CTLY←⍳0
[28]   'PROCEDURE FILE UPDATED'
[29]   →0
[30]  ⍝************************************************************
[31]  ⍝ERROR MESSAGES
[32]  ERROR1:'THE SHARED VARIABLE OFFER FAILED'
[33]   →0
[34]  ERROR2:'ENTERING DATA FROM THE KEYBOARD FAILED'
[35]   →0
[36]  ERROR3:'UPDATING THE PROCEDURE FILE FAILED'
    ∇
```

In this example, the procedure file is on diskette drive 1, file 1.

The new record number is assigned to B for later use in this user-defined function.

DATY must contain the information to be written to the procedure file. Note, in this example, that the procedure file contains 128-byte character records.

TAPE DATA CARTRIDGE HANDLING AND CARE

- Protect the tape data cartridge from dust and dirt. Cartridges that are not needed for immediate use should be stored in their protective plastic envelopes.

- Keep data cartridges away from magnetic fields and from ferromagnetic materials that might be magnetized. Information on any cartridge exposed to a magnetic field could be lost.

- Do not expose data cartridges to excessive heat (more than 130°F [54°C]) or sunlight.

- Do not touch or clean the tape surface.

- If a data cartridge has been exposed to a temperature drop exceeding 30°F (-1°C) since the last usage, move the tape to its limits before using the tape. The procedure for moving the tape to its limits is:

1. Use the)LIB command to move the tape to the last marked file.

2. Use the)MARK command to mark from the last marked file to the end of the tape. For example:

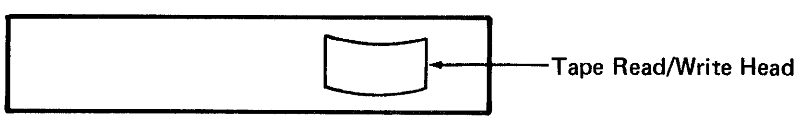
)MARK 20 10 n

   where n is the number of the last marked file, plus one.

3. When ERROR 012 (end of tape) is displayed, use the)REWIND command to rewind the tape.

TAPE HEAD CLEANING PROCEDURE

Occasional cleaning of the tape read/write head provides more reliable operation of the tape drive. Use a soft lint-free cloth or paper towel dampened with isopropyl alcohol to clean tape oxide from the tape head. Then wipe the tape head dry.



Tape Drive Opening

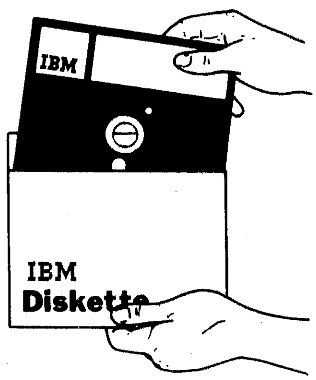
DISKETTE HANDLING AND CARE

Diskette Insertion

CAUTION
If a diskette has been exposed to temperatures outside the range 50°F to 125°F (10°C to 51°C), keep the diskette at room temperature for about five minutes before inserting it in the diskette drive.

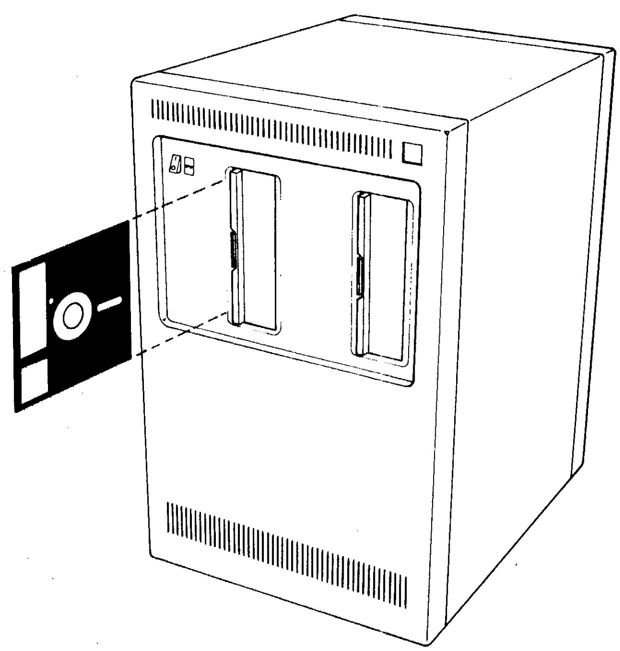
1. Open the diskette drive cover.

2. Remove the diskette from its envelope by grasping its upper edge and and listing.



3. Insert the diskette into the diskette drive.

Note: The permanent diskette label must be in the lower right corner as the diskette is inserted in the drive.



4. Close the diskette drive cover only after the diskette is fully inserted.

Diskette Removal

1. Open the diskette drive cover (only when the cursor is flashing).

2. Remove the diskette by grasping its upper edge and pulling it straight out.

3. Return the diskette to its envelope.

Handling Defective Cylinders

With use, the diskette can develop areas on which readable records cannot be written. A diskette with a defective area should be reinitialized to use one of the alternate cylinders available. See the *IBM 5110 Customer Support Functions Reference Manual*, SA21-9311, for information on the initialization function.

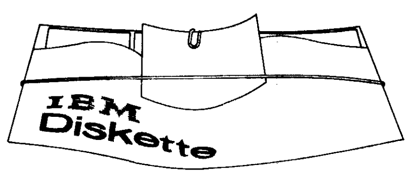
Note: Before reinitializing a diskette, copy any useful data from the diskette onto another diskette.

After the diskette has been reinitialized, record the number of the defective cylinder on the permanent label.

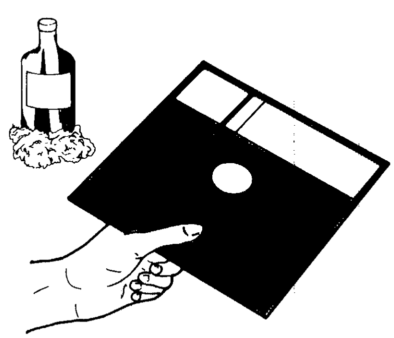
Handling Precautions

Do not bend or fold the diskette.

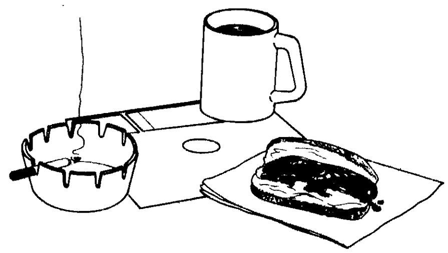
Do not use rubber bands or paper clips on the diskette.



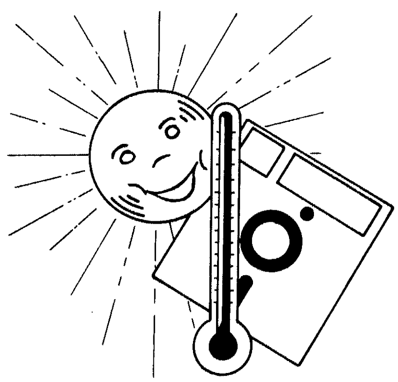
Do not touch or clean the exposed diskette surface.



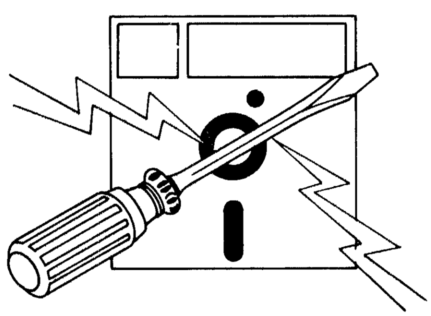
Do not eat, smoke, or drink while handling the diskette.



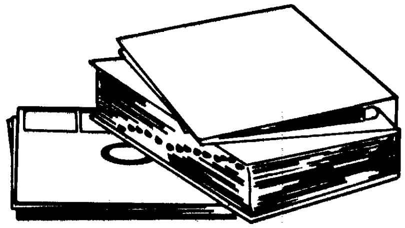
Do not expose the diskette to excessive heat or sunlight.



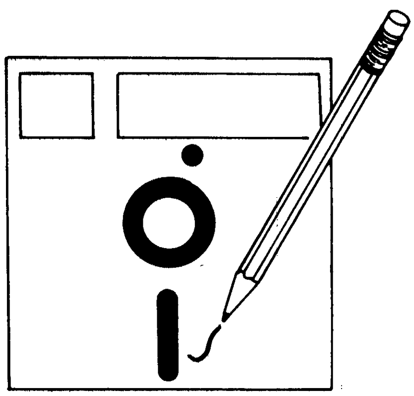
Do not use magnets or magnetic objects near the diskette. Data can be lost from a diskette that is exposed to a magnetic field.



Do not place heavy objects on the diskette.



Do not erase labels attached to the diskette, or make any erasures on or near the diskette. Erasure residue could get in the diskette, and this should be avoided. To discourage erasures, it is recommended that you use a fiber-tip or ballpoint pen when marking on the diskette labels. Mark temporary labels before attaching them to the diskette. Alter the temporary labels with the diskette in the envelope.

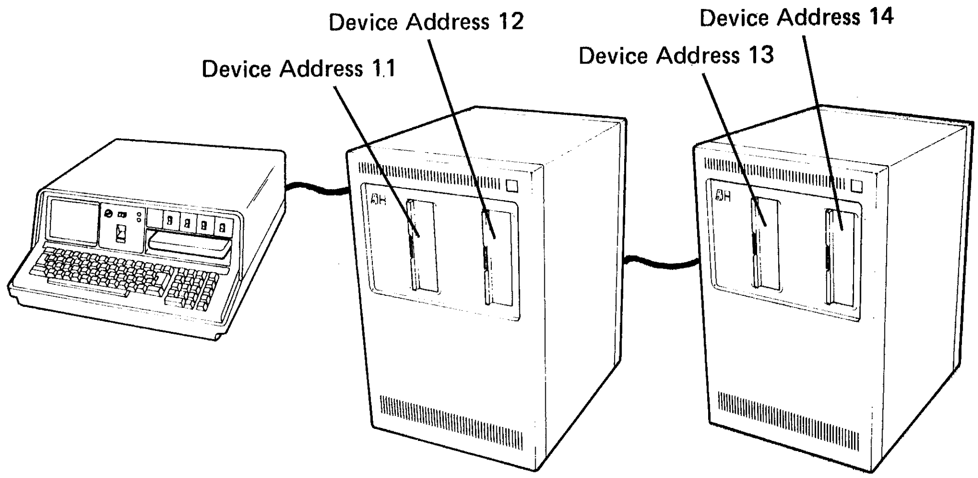


DEVICE ADDRESSING

The device address for the 5110 tape units and diskette drives are as follows:

Device	Device Address
Built-in tape unit	1
Auxiliary tape unit	2
Diskette drive 1	11
Diskette drive 2	12
Diskette drive 3	13
Diskette drive 4	14

The following illustration shows the device address for diskette drives attached to the 5110:



DEFAULT DEVICE

The default device is the tape unit or diskette drive used by the system when only the file number is specified. For example:

)LOAD 3 DATA

Fewer than four digits are specified; therefore, the value specified represents only the file number, and the default device is used.

For the Model 1, the default device is the built-in tape unit (1). For the Model 2, the default device is the first diskette unit (11).

CUSTOMER SUPPORT FUNCTIONS

The following customer support programs are provided with the 5110:

- Tape-to-tape copy
- Diskette-to-diskette copy
- Diskette-to-tape copy
- Diskette initialization
- Generalized loader
- Diskette recovery
- Tape recovery
- Diskette Compress
- Label display

See the *IBM 5110 Customer Support Functions Reference Manual,* SA21-9311, for a complete description of each program.

AUDIBLE ALARM

The 5110 has an audible alarm feature that can be used to signal the operator. For example, the audible alarm can signal the completion of a user-defined function or a procedure file (see *Procedure File* in this chapter). The ⎕CC system function sounds the audible alarm (see Chapter 5). For example:

To sound the audible alarm, the left argument must be a 2.

```
2 ⎕CC 2 2 2
```

The audible alarm sounds for approximately 1/4 second for each 2 in the right argument.

Chapter 3. System Commands

OVERVIEW

The 5110 contains an active workspace, which is the part of internal storage where the user's data and user-defined functions (programs) are stored. When the power is turned off or the RESTART switch is pressed on the 5110, all the data in the active workspace is lost. However, the contents of the active workspace can be saved on tape or diskette (stored workspace) and then read back into the active workspace for use at a later time (see *System Command Descriptions* in this chapter). The contents of the active workspace then exist in both the active workspace and on tape or diskette.

The tape and/or diskette is your library; that is, it is a place where you can store data for later use. Before a tape or diskette can be used, it must be formatted. A formatted tape or diskette contains one or more files where data can be stored. Each file has a file header, which contains information about the file. See the)LIB system command in this chapter for a description of the file header.

The system commands, which are used to control and provide information about the system, are discussed next.

SYSTEM COMMAND DESCRIPTIONS

The following list shows how system commands are used to control and provide information about the various parts of the system. Each system command is described in detail later in this chapter.

Commands That Control the Active Workspace

Command	Meaning
)CLEAR	Clear the active workspace.
)COPY	Copy stored objects (see note 1) into the active workspace.
)ERASE	Erase global objects (see note 1) from the active workspace.
)LOAD	Replace the active workspace with a stored workspace.
)PCOPY	Copy stored objects (see note 1) into the active workspace, and protect objects in the active workspace from being destroyed.
)SYMBOLS	Change the number of symbols allowed in the active workspace.
)WSID	Change the active workspace ID.
)RESUME	Replace the active workspace with a workspace stored using the)CONTINUE command, and reestablish the system environment as it was when the workspace was stored.

Commands That Control the Library (Tape and/or Diskette)

Command	Meaning
)CONTINUE	Write the contents of the active workspace on tape or diskette. The active workspace can contain suspended functions and open shared variables.
)DROP	Drop a tape or diskette file.
)FILEID	Change the file name on a diskette volume.
)MARK	Format the tape or diskette.
)FREE	Make a diskette file available for reallocation.
)SAVE	Write the contents of the active workspace on tape or diskette. The active workspace cannot contain suspended functions or open shared variables.
)PROTECT	Invoke or remove the write-protect feature on a diskette file.
)VOLID	Change the volume ID or access protection indicator of a diskette device.

Commands That Provide Information about the System

Command	Meaning
)FILEID	Display the file name on a diskette volume.
)FNS	Display the names of the user-defined functions.
)LIB	Display file headers.
)SI	Display the state indicator.
)SINL	Display the state indicator and local names.
)SYMBOL	Display the number of symbols allowed and used in the active workspace.
)VARS	Display the names of the global variables.
)VOLID	Display the volume ID of a diskette device.
)WSID	Display the active workspace ID.

Other Commands That Control the System

Command	Meaning
)LINK	Load microcoded programs, such as functions or asynchronous communications.
)OUTSEL	Select printer output.
)PROC	Open a procedure file.
)REWIND	Rewind the tape.
)SORT	Transfer system control to the sort function.

Notes:
1. Objects refers to both user-defined functions and variables.
2. The system commands)CONTINUE,)COPY,)DROP,)LOAD,)MARK,)REWIND, and)SAVE will blank the top 8 to 12 lines on the display screen when they are used.

All system commands (and only system commands) have as their first character a right parenthesis. Each system command must begin on a new line. Parameters (required or optional information) for the system commands must be separated by blanks.

Some system commands can be entered two ways:

- The system command can be entered one character at a time from the keyboard.

- The system commands)LOAD,)SAVE,)CONTINUE,)LIB,)FNS,)VARS,)COPY,)WSID,)OUTSEL, and)MARK can be entered in one operation if you hold down the CMD key while pressing the top-row key just below the label of the command you want.

System commands can be executed two ways:

- Press EXECUTE after the parameters, if required, are entered.

- Read the system command and required parameters from a procedure file (see *The)PROC Command* in this chapter).

System commands cannot be used within APL expressions and cannot be used as part of a user-defined function (see Chapter 6).

Following is an explanation of some terms and symbols used as parameters for system commands:

- *Device/file number* specifies the device and file to be used. The built-in tape unit is device 1, the auxiliary tape unit is device 2, diskette drive 1 is device 11, diskette drive 2 is device 12, and so on. If the value specified is less than four digits, the default device is assumed and the value specified represents only the file number (see *Default Device* in Chapter 2). If the value specified is greater than three digits, the rightmost three digits specify the file number and remaining digits specify the device. For example:

Device/File Number	Meaning
1	For a 5110 Model 1–Tape unit 1, file 1
	For a 5110 Model 2–Diskette drive 1, file 1
2002	Tape unit 2, file 2
12002	Diskette drive 2, file 2

- *Workspace ID* specifies the name of the stored workspace on the media. See *Variable and File Names* in Chapter 2 for a description of the valid file names.

- *Password* is any combination of up to 8 alphabetic or numeric characters (with no blanks). If more that 8 characters are entered, only the first eight are used.

- *Object* is a user-defined function or variable name.

- Parameters enclosed in brackets can be optional in certain cases.

Note: See *Name Processing* in Chapter 2 for information on when the device/file number and file ID parameters are optional for tape and diskette operations.

The)CLEAR Command

The)CLEAR command clears the active workspace. A cleared workspace has no valid name and contains no user-defined variables or functions and no data. The workspace attributes are set to:

Index origin	–	1
Workspace identification	–	CLEAR WS
Comparison tolerance	–	1E 13
Printing width	–	64
Printing precision	–	5
Random number seed	–	16807
Data printed	–	ALL
Symbols	–	125

When the command is successfully completed, CLEAR WS is displayed.

Syntax

)CLEAR

There are no parameters.

The)CONTINUE Command

The)CONTINUE command, using the specified workspace ID, stores the contents of the active workspace onto tape or diskette (the media) without changing the active workspace. Primarily, this command stores active status, such as suspended functions, so that an operation can be resumed later on the same or a similar machine. When the command is successfully completed, CONTINUED *device/file number workspace ID* is displayed.

The)CONTINUE command on the 5110 is similar in function and format to the)SAVE command (except as noted below).

Notes:
1. A clear workspace cannot be written on the media.
2. A workspace with suspended functions can only be written on the media with the)CONTINUE command (it cannot be written to the media by the)SAVE command).
3.)COPY and)PCOPY commands cannot specify stored workspaces that were written on the media by the)CONTINUE command.
4. A stored workspace written to the media by the)CONTINUE command cannot be loaded into a 5110 active workspace that is smaller than the original active workspace.
5. If a stored workspace that was written to the media by the)CONTINUE command is loaded into another 5110 with a larger active workspace, the workspace available (see the ⎕WA system variable in Chapter 5) is the same as when the workspace was written to the media.
6. If ATTN is pressed during a)CONTINUE operation, the system operation is interrupted and the file is set to unused.
7. Shared variable execution status can be stored by the)CONTINUE command. A subsequent)RESUME allows the user to resume execution if the media is restored to the same condition as when the workspace was stored by the)CONTINUE command (see *The)RESUME Command* in this chapter).
8. The ⎕LX system function is executed if the stored workspace is loaded by the)LOAD command. However, the ⎕LX system function is *not* executed if the workspace is loaded by the)RESUME command.
9. Workspaces are stored and loaded into the active workspace faster by the)CONTINUE command than by the)SAVE command.
10. IMFs (internal machine fixes) are not stored by)CONTINUE. If an IMF is required for operation of the stored workspace, it should be reapplied by the)LINK command (if the IMF is not already in the system) before the workspace is reloaded.
11. Workspaces stored on tape by the)CONTINUE command *cannot* be interchanged between the IBM 5110 and the IBM 5100.

Syntax

)CONTINUE [device/file number] [workspace ID] [:password]

Where:

device/file number (optional) is the number of the device and file on the device where the contents of the active workspace are to be written. If no device/file number is specified, the device/file number from which the active workspace was loaded or specified by a previous)WSID command is used.

workspace ID (optional) is the name of the workspace to be stored. This name must match the workspace ID of both the active workspace and the file to be used on the media unless the file is marked unused. If the file is marked unused, the active workspace ID and file workspace ID are changed to this workspace ID. If no name is specified in the command, the name of the active workspace is used.

:password (optional) is any combination of up to 8 alphabetic or numeric characters (without blanks), preceded by a colon. This sequence of characters must be matched when the stored workspace is to be read back into the active workspace. If no workspace ID or password is entered, the password associated with the active workspace (if any) is assigned to the workspace being stored. If just the workspace ID and no password is entered, any password associated with the active workspace is not used.

The)COPY Command

The)COPY command copies all or specified global objects from a stored workspace to the active workspace. Only objects in stored workspaces that were written on the media with the)SAVE command can be copied. When the command is successfully completed, COPIED *device/file number workspace ID* is displayed.

Notes:
1. If the active workspace contains suspended functions, objects cannot be copied into it.
2. If the ATTN key is pressed during a)COPY operation, the system operation is interrupted and the amount of information copied into the active workspace is unpredictable.
3. If the active workspace contains shared variables, objects having the same name as a shared variable cannot be copied into the active workspace.

Syntax

)COPY [device/file number] [workspace ID] :password [object name(s)]

Where:

device/file number (optional for diskette only) is the number of the device, and workspace file the objects are copied from.

workspace ID (optional) is the name of the stored workspace on the media. This parameter is optional only if the device/file number is specified.

:password is the security password assigned by a previous)WSID or)SAVE command. If no password was assigned previously, a password cannot be specified by this command.

object name(s) (optional) is the name of the global objects(s) to be copied from the designated stored workspace. If this parameter is omitted, all global objects in the designated stored workspace are copied.

The)DROP Command

The)DROP command marks a specified file unused. After the file has been marked unused, the data in the file can no longer be read from the media. Once the file is marked unused, the file is available to be used in the same manner as a newly formatted file (see *The)MARK Command*). When the command is successfully completed, DROPPED *device/file number file ID* is displayed.

Note: For diskette files, the)FREE command must be used if the diskette file space is to be reallocated to another file (see *The)FREE Command* in this chapter).

Syntax

)DROP [device/file number] [file ID]

Where:

device/file number (optional for diskette only) is the number of the device and the file on the media.

file ID is the name of the file to be marked unused. This parameter is required for all APL stored workspace files and is optional for data files if the device/file number is specified. The name specified by this parameter is compared to the media file ID. If the names do not match, the error message FILE NOT FOUND is displayed.

Note: When a diskette file is dropped, the diskette file name is *not* changed.

The)ERASE Command

The)ERASE command erases the named global objects from the active workspace. No message is displayed after the successful completion of the command.

Notes:
1. When a pendent function is erased, the response SI DAMAGE is issued.
2. If the object being erased is a shared variable (see Chapter 7), the shared variable will be retracted.
3. Even after the object is erased, the name remains in the symbol table (the part of the active workspace that contains all the symbols used).

Syntax

)ERASE object name(s)

Where:

object name(s) are global names separated by blanks.

The)FILEID Command

The)FILEID command is used to change or display the file name on a diskette volume. When the)FILEID command is issued without specifying a new file name, *device file number file name* is displayed. When the)FILEID command is issued specifying a new file name, WAS *device/file number file name* is displayed and the file name is changed to the new file name.

Syntax

)FILEID [device/file number] [current file name] [new file name]

Where:

device/file number (optional) is an integer that specifies the device/file number.

current file name (optional) is the current name of the file. This parameter is optional only if the device/file number is specified and a new name is not specified.

new file name (optional) will be the new name for the file.

The)FNS Command

The)FNS command displays the names of all global user-defined functions in the active workspace. The functions are listed alphabetically. If the character parameter is specified, the names are displayed beginning with the specified character or character sequence.

Note: You can interrupt the)FNS command by pressing the ATTN key.

Syntax

)FNS [character(s)]

Where:

character(s) (optional) is any sequence of alphabetic and numeric characters that starts with an alphabetic character and contains no blanks. This sequence of characters determines the starting point for an alphabetic listing.

The)FREE Command

The)FREE command makes diskette files available for reallocation. After the file is made available for reallocation, the space can be used for other numbered files by the)MARK command.

When the command is successfully completed, FREED *device/file number file ID* is displayed.

Syntax

)FREE [device/file number] file ID

Where:

device/file number (optional) is an integer that specifies the diskette device/file number where the diskette file is stored.

file ID is the name of the stored file. This parameter must be specified for APL workspace files. For data files, this parameter is optional if the device/file number is specified.

The)LIB Command

The)LIB command displays the file headers of the files on tape or diskette. The file header contains the following information:

- File number.
- File ID. The file ID can be from 1 to 17 characters. If the file contains a stored workspace, the file ID is the same as the stored workspace ID.
- File type. The file type is a two-digit code; the following chart gives the meaning of each code:

File Type	Description
00	Unused file
01	Sequential access–exchange data file
02	Sequential access–general exchange data file
03	BASIC source file
04	BASIC workspace file
05	BASIC keys file
06	APL continued file–5100 only
07	APL saved file
08	Sequential access–APL internal format data file
09	Direct access–exchange, general exchange, and unblocked/unspanned data file
B9	Direct access–unblocked/unspanned data file that can be used for exchanging data with other products
10	Direct access–APL internal format data file
11	BASIC source file–5110 only
12	BASIC keys file–5110 only
15	Direct access–APL mixed record data file
16	Patch, tape recovery, and tape copy file–5100 only
17	Diagnostic file–5100 only
18	Communications file–5100 only
19	IMF file–5100 only
21	Patch, tape recovery, and tape copy file–5110 only
22	Feature file–5110 only
23	IMF file–5110 only
24	Diagnostic file–5110 only
26	APL continued file–5110 only
72	Storage dump–tape only

- Size of the file. The files are formatted in increments of 1024-byte blocks of storage.
- Number of unused contiguous 1024-byte blocks of storage in the file.

- (Tape only) Number of defective records (512-byte blocks) in the file; an asterisk (*) is displayed if there are more than nine defective records.

  Note: This value can indicate when you should relocate a file to avoid loss of data due to defective areas on the tape.

- (Diskette only) Write-protect indicator; a P indicates that the file is write-protected. That is, you cannot write data to the file. Otherwise, this position is blank.

- (Diskette only) Starting location of the first record in the file. This hex value is in the format CCHRR, where CC is the cylinder number, H is the head number, and RR is the sector number.

Following is an example of a tape file header:

```
006 FILE6                     07 010,001 0
```

File Type

Size of the File

Available Storage

Number of Defective Records

File ID

File Number

Following is an example of a diskette file header:

```
0023 FILE23                   26 0020,0007 P 19104
```

File Type

Size of the File

Available Storage

Write Protect Indicator

Starting Location (CCHRR)

File ID

File Number

You can interrupt the)LIB command operation by pressing the ATTN key.

Syntax

)LIB [device/file number]

Where:

device/file number (optional) is the number of the device and starting file number. All file headers from that file to the end of the media are displayed. If this parameter is not specified, the default device is assumed and:

- For the 5110 Model 1, the display begins with the first file following the file you are currently positioned at on tape unit 1
- For the 5110 Model 2, the display begins with the first file on diskette drive 1

For the tape units, the device number followed by 000 displays the file headers beginning with the first file you are currently positioned at on the tape unit. For example, 2000 specifies tape unit 2.

For diskette drives, the device number followed by 000 displays the file headers beginning with the first file on the diskette drive.

Note: For diskette files, only the file headers of the formatted files (see *The)MARK Command*) are displayed.

The)LINK Command

The)LINK command is used to load special programs (file type 21 only), such as asynchronous communications or customer support functions, into the 5110.

Syntax

)LINK [device/file number] [file ID]

Where:

device/file number (optional for diskette only) is the number of the device and the number of the file that contains the microcoded programs.

file ID (optional) is the name of the file that contains the microcode programs. This parameter is optional on diskette only if the device/file number is specified.

Note: If an I/O error or workspace full condition occurs when you are using the)LINK command, the program is not loaded and the CLEAR WS message is displayed. In this case, you should press RESTART to clear any IMFs (internal machine fixes), storage allocated for procedure files, or other special programs from the active workspace. Then try the)LINK command again. If the program loads successfully, you must also reload any required IMFs. If the program still does not load, call your service representative.

The)LOAD Command

The)LOAD command loads the contents of a stored workspace from the tape or diskette into the active workspace, completely replacing the contents that were in the active workspace. When the command is successfully completed, LOADED *device/file number* *workspace ID* is displayed.

Notes:

1. If the ATTN key is pressed during a load operation, the system operation is interrupted and the active workspace is cleared.
2. Any shared variables that were established when the stored workspace was written to the media are not automatically reestablished during a load operation. (See *The)RESUME Command* for information on reestablishing shared variables when a stored workspace was written to the media by the)CONTINUE command.)
3. The system *does not* successfully terminate current input/output operations (see Chapter 7) when a)LOAD command is issued. Therefore, if a new data file is being created or if records are being added to an existing data file and a)LOAD command is issued without terminating the operation, the file header is not updated and the file contents are unpredictable.
4. If the stored workspace contains a latent expression (see the ⎕LX system variable in Chapter 5), the latent expression is always executed when the)LOAD command is issued.

Syntax

)LOAD [device/file number] [workspace ID] :password

Where:

device/file number (optional for diskette only) is the number of the device and the number of the file on the media.

workspace ID (optional) is the name of the stored workspace. This parameter is optional only if the device/file number is specified.

:password is the security password assigned to the stored workspace by a previous)WSID,)CONTINUE, or)SAVE command. If no password was previously assigned, a password cannot be specified. If a password was assigned to the stored workspace but is not specified, or if it is incorrectly specified for this command, the error message WS LOCKED is displayed.

The)MARK Command

The)MARK command formats the media so that the active workspace or data can be saved on it. Each)MARK command formats a specified number of files to a specified size. Additional files of different sizes can be formatted by additional)MARK commands.

When the operation is successfully completed, MARKED *number of the last file marked size of the last file marked* is displayed.

Notes:
1. The ATTN key is not operative during the)MARK command operation.
2. If the message ALREADY MARKED is displayed after a)MARK command has been issued, the specified file already exists on the media. To re-mark the specified file, enter GO. If the file is not to be re-marked, press EXECUTE to continue.

   CAUTION
   If an existing file on tape is re-marked, the original information in the re-marked file and the existing files following the re-marked file cannot be used again.

3. (Diskette only) The files within the range specified by the)MARK command are the *only* files on the diskette that are affected.
4. (Diskette only) The user can control the location of a file on the diskette only by using a totally unformatted diskette and issuing)MARK commands in the same order as the files that are to be formatted on the diskette.
5. (Diskette only) If a size of zero is specified, the files within the range specified by the)MARK command are made available for reallocation. After the file is made available for reallocation, the diskette space can be used for other numbered files by the)MARK command.
6. (Diskette only) The maximum number of files allowed on a diskette is 19 for a diskette type 1; 45 for a diskette type 2; 71 for a diskette type 2D (unless the diskette initialization function was used to get additional file headers; see the *IBM 5110 Customer Support Functions Reference Manual,* SA21-9311).

Syntax

)MARK size number of files to mark starting file number [device]

Where:

size is an integer specifying the size of each file in 1024-byte (1K) blocks of storage.

The following formulas can be used to determine the maximum size the file must be marked. The formula for a workspace file (the contents of the active workspace written to tape with a)SAVE or)CONTINUE command) is:

MAXSIZE=3+⌈(CLEAR-ACTIVE)÷1024

where:

- MAXSIZE is the maximum amount of media storage (number of 1024-byte blocks) that might be required to write the contents of the active workspace to the media.
- CLEAR is the value of ⎕WA (see Chapter 5) in a clear workspace.
- ACTIVE is the value of ⎕WA just before the contents of the active workspace are written to the media.

The amount of storage actually required can vary considerably between different workspaces because of their content, if they are on tape or diskette, or if they were written with a)SAVE or)CONTINUE command.

The formula for a data file (data written to the media with an APL shared variable—see Chapter 7) when all of the data is contained in the active workspace is:

MAXSIZE=⌈(without-with)÷1024

where:

- MAXSIZE is the maximum amount of media storage (number of 1024-byte blocks) required to write the data to the media.
- WITH is the value of ⎕WA (see Chapter 5) with the data in the active workspace.
- WITHOUT is the value of ⎕WA before any data to be written to the media was stored in the active workspace.

There is no formula for determining what size to mark a data file when the data is written to the media as it is entered from the keyboard. The amount of media storage required depends upon how much data is entered from the keyboard and what type of data is used. For information on how many bytes of storage are required by the various types of data, see *Storage Considerations* in Chapter 2.

Note: The file header for each marked tape file requires 0.5K bytes of storage. Therefore, the number of bytes of tape storage required for each file is the specified size of the file plus 0.5K.

number of files to mark is an integer specifying the number of files of the specified size to format.

starting file number is an integer specifying the file number where formatting is to start.

device (optional) specifies the device that contains the media to be formatted. An entry of 1 specifies tape unit 1, 2 specifies tape unit 2, 11 specifies diskette drive 1, and 12 specifies diskette drive 2. If no entry is made, the default device is used.

To format a tape for four 12K files, two 16K files, and three 10K files, the following commands are required:

```
)MARK   12   4   1
)MARK   16   2   5   Starting File Number
)MARK   10   3   7
```

To format a diskette on diskette drive 1 for four 12K files, two 20K files, and one 10K file, the following commands are required:

```
)MARK   12   4   1   11
)MARK   20   2   5   11   Device 11 (diskette drive 1)
)MARK   10   1   7   11
```

The)OUTSEL Command

The)OUTSEL command specifies which data on the display will go to the printer.

Syntax

)OUTSEL [option]

Where:

option is one of the following:

- When ALL is specified, all subsequent information that is displayed will be printed.
- When OUT is specified, only the output is sent to the printer; input is displayed, but it does not go to the printer.
- When OFF is specified, none of the information displayed is printed, unless it is assigned to an APL shared variable used by the printer (see Chapter 7).

If no parameter is specified, ALL is assumed. After a)LOAD or)CLEAR command or when the machine is first turned on, the ALL option is active.

The)PCOPY Command

The)PCOPY command copies all or specified global objects from a stored workspace into the active workspace. It is the same as the)COPY command, except that if the object name already exists in the active workspace, it is not copied from a stored workspace. Therefore, the object in the active workspace is protected from being overlaid and destroyed. Only objects in stored workspaces that were written on the media with the)SAVE command can be copied.

When the command is successfully completed, COPIED *device/file number workspace ID* is displayed.

Notes:
1. If the active workspace contains suspended functions, objects cannot be copied into it.
2. If the ATTN key is pressed during a)PCOPY operation, the system operation is interrupted and the amount of information copied into the active workspace is unpredictable.
3. If the specified object name already exists in the active workspace, the message NOT COPIED: *object name* is also displayed.

Syntax

)PCOPY [device/file number] [workspace ID] :password [object name(s)]

Where:

device/file number (optional for diskette only) is the number of the device and the stored workspace file.

workspace ID (optional) is the name of the stored workspace on the media. This parameter is optional only if the device/file number is specified.

:password is the security password assigned by the previous)WSID or)SAVE command. If no password was assigned, a password cannot be specified by this command.

object name(s) (optional) is the name of the global object(s) to be copied from the designated stored workspace. If omitted, all global objects in the designated stored workspace are copied, except those already in the active workspace (if any).

The)PROC Command

The)PROC command opens an external procedure file that is used by the 5110 as an alternative to keyboard input (see Chapter 2). A procedure file contains character records that represent any input that is possible to enter from the keyboard, such as system commands, function definitions, or APL statements.

When a)PROC command is issued, all input comes from the procedure file until the end-of-data is reached. Then the 5110 goes back to using regular keyboard input.

Notes:
1. The)PROC command requires 768 bytes from the active workspace. This space can be allocated only in a CLEAR WS; therefore, the active workspace is always cleared the first time a)PROC command is issued. The active workspace is *not* cleared for any subsequent)PROC commands because the 768 bytes are already allocated.
2. You can establish the 768 bytes of active workspace by issuing the)PROC command without any parameters. This allows you to clear the active workspace, establish the 768-byte area required by the)PROC command, use the active workspace for any APL operations, and issue subsequent)PROC commands when you are ready to execute statements from a procedure file.

Syntax

)PROC [device/file number] [file ID]

Where:

device/file number (optional for diskette only) is an integer that specifies the device/file number where the procedure file is stored.

file ID (optional) is the name of the procedure file. This parameter is optional only if the device/file number is specified.

The)PROTECT Command

The)PROTECT command invokes or removes the write-protect indicator on a diskette file. When the write-protect indicator is on for a specified diskette file, you cannot write to that diskette file. This will prevent someone from writing data to a diskette file accidently.

Syntax

)PROTECT [device/file number] [file ID] [OFF/ON]

Where:

device/file number (optional) is an integer that specifies the device/file number where the write-protect indicator is invoked or removed.

file ID (optional) is the name of the diskette file. This parameter is optional only if the device/file number is specified.

OFF/ON (optional) specifies whether the write-protect indicator is to be removed (OFF) or invoked (ON). This parameter can be used only when the file ID is specified. If this parameter is not specified, the write-protect indicator is invoked.

The)RESUME Command

The)RESUME command loads the contents of a workspace that was stored by the)CONTINUE command into the active workspace and attempts to reestablish the system's environment as it was when the workspace was stored.

This includes reestablishing shared variables and reestablishing suspended functions. For shared variables, an attempt is made to reestablish the shared variables as follows:

- Diskette data files are checked to make sure they are on the proper diskette drive. If the data files are not on the proper diskette drive, a WRONG VOLUME error message is displayed. However, the stored workspace is still loaded into the active workspace.

  If the WRONG VOLUME error message is displayed, place the diskette containing the correct data files in the proper diskette drive before doing any input/output.

  CAUTION
  If the correct diskette is not placed in the proper diskette drive, data on the diskette presently in the diskette drive can be overwritten and lost.

- Tape files are *not* checked. It is assumed that all tapes remain in the correct position and are mounted on the correct devices. If the wrong tape is placed in the tape drive or if the tape has been repositioned, it is possible to overwrite in the wrong location.

- Any shared variables to the printer are reestablished.

- Any shared variables to attached serial I/O devices are reestablished; however, you are responsible for establishing the correct serial I/O environment (such as making sure the serial I/O devices are attached) before issuing the)RESUME command.

When the)RESUME command is used, the latent expression (⎕LX) is not executed and)OUTSEL ALL is assumed unless a shared variable to the printer was specified. When the command is successfully completed, RESUMED *device/file number workspace ID* is displayed.

Syntax

)RESUME [device/file number] [workspace ID] :password

Where:

device/file number (optional) is the number of the device and file number of the stored workspace.

workspace ID (optional) is the name of the stored workspace. This parameter is optional only if the device/file number is specified.

:password is the security password assigned to the stored workspace by a previous)WSID or)CONTINUE command. If no password was previously assigned, a password cannot be specified. If a password was assigned to the stored workspace but is not specified, or if it is specified incorrectly for this command, the error message WS LOCKED is displayed.

The)REWIND Command

The)REWIND command rewinds the specified tape. No message is displayed at the successful completion of this command.

Syntax

)REWIND [device number]

Where:

device number (optional) is the tape (on drive 1 or 2) to be rewound. If this parameter is omitted, tape 1 is rewound.

The)SAVE Command

The)SAVE command stores the contents of the active workspace onto the media without changing the contents of the active workspace. Individual global objects can be copied from the stored workspace to the active workspace. When this command is successfully completed, SAVE *device/file number workspace ID* is displayed. Do not remove the tape or diskette until this message is displayed.

Notes:
1. A clear workspace or a workspace with suspended function cannot be written to the media by the)SAVE command; however, a workspace with suspended functions can be written to the media by the)CONTINUE command.
2. The)COPY and)PCOPY commands can specify stored workspaces that were written on the media only if the)SAVE command was used.
3. Depending on the amount of data in the stored workspace, a stored workspace that was written to the media by the)SAVE command can be loaded into another 5110 with a smaller active workspace.
4. If ATTN is pressed during a)SAVE operation, the system operation is interrupted and file is set to unused.
5. Shared variables cannot automatically be reestablished in the active workspace when the workspace is stored on the media by the)SAVE command. However, the shared variable can be automatically reestablished in the active workspace when the workspace is stored by the)CONTINUE command.
6. IMFs (internal machine fixes) are not stored by the)SAVE operation. If an IMF is required, it is necessary to use the)LINK command to reload the IMF (if the IMF is not already in the system) before the stored workspace is reloaded.
7. Only workspaces stored on the media by the)SAVE command can be interchanged between the 5110 and the 5100.

Syntax

)SAVE [device/file number] [workspace ID] [:password]

Where:

device/file number (optional) is the number of the device and the file on the tape or diskette where the contents of the active workspace are to be written. If no device/file number is specified, the device/file number from which the active workspace was loaded or which was specified by a previous)WSID command is used.

workspace ID (optional) is the name of the workspace to be stored. This name must match the workspace ID of both the active workspace and the file to be used on the media unless the file is mark unused. If the file is marked unused, the active workspace and file workspace ID will be changed to this workspace ID. If no name is specified in the command, the name of the active workspace is used.

:password (optional) is any combination of up to 8 alphabetic or numeric characters (without blanks), preceded by a colon. This sequence of characters must be matched when the stored workspace is to be read back into the active workspace. If no workspace ID or password is entered, the password associated with the active workspace (if any) is assigned to the workspace being stored. If just the workspace ID and no password is entered, any password associated with the active workspace is not used.

The)SI Command

The)SI command displays the names of the suspended and pendent user-defined functions. The suspended functions are indicated by an *, with the most recently suspended function listed first, followed by the next most recently suspended function, and so on.

Syntax

)SI

There are no parameters.

The)SINL Command

The)SINL command displays the names of the suspended and pendent user-defined functions and the names local to each function. The suspended functions are indicated by an *, with the most recently suspended function listed first, followed by the next most recently suspended function, and so on.

Syntax

)SINL

There are no parameters.

The)SORT Command

The)SORT command transfers system control to the sort feature (see the *5110 Customer Support Functions Reference Manual*, SA21-9311, for a complete description of the sort feature).

Note: If the)SORT command is issued from a procedure file (see *The)PROC Command*), the statement immediately following the)SORT command statement is used as the initial input to the sort feature. When the sort feature is complete, the 5110 continues to take input from the procedure file.

Syntax

)SORT

There are no parameters.

The)SYMBOLS Command

The)SYMBOLS command is used to change or display the number of symbols (variable names, function names, and labels) allowed in the active workspace. The number of symbols allowed can only be changed immediately after a)CLEAR command has been issued. In a clear workspace, the number of symbols allowed is initially set to 125 by the 5110. When the command is used to display the number of symbols allowed, IS *the number of symbols allowed*, *number of symbols used* IN USE is displayed. When the command is used to change the number of symbols allowed, WAS *the former number of symbols allowed* is displayed.

Note: When a stored workspace is loaded into the active workspace, the number of symbols allowed in the active workspace will be the same as when the stored workspace was written to the media.

Syntax

)SYMBOLS [n]

Where:

n (optional) is an integer equal to or greater than 26 that specifies the number of symbols allowed in the active workspace. Each symbol allowed requires 8 bytes of storage in the active workspace.

Notes:
1. The number of symbols allowed is assigned in blocks of 21; therefore, the actual number allowed can be larger than the number specified.
2. When a symbol is used in the active workspace, it remains in use even though the object is erased or, in the case of VALUE ERROR never existed. When the active workspace is written to tape with the)SAVE command and subsequently reloaded, these unused names are removed from the symbol table, and the number of symbols in use will be the same as the number of objects in the workspace.
3. The total number of allowed symbols remains the same after you write the workspace to the media with a)SAVE or)CONTINUE command and then reload the workspace to the active workspace. The number of symbols in the active workspace can be changed as follows:
   a. Save the active workspace with the)SAVE command.
   b. Clear the active workspace with the)CLEAR command.
   c. Set the new number of symbols with the)SYMBOLS command.
   d. Copy the stored workspace to the active workspace with the)COPY command.
   e. The workspace ID and ⎕LX must be respecified.

The)VARS Command

The)VARS command displays the names of all global variables in the active workspace. The variables are displayed alphabetically. If the character parameter is included, the names are displayed beginning with the specified character sequence.

Syntax

)VARS [character(s)]

Where:

character(s) (optional) is any sequence of alphabetic and numeric characters that starts with an alphabetic character and contains no blanks. This entry can be used to define the starting point for an alphabetic listing.

The)VOLID Command

The)VOLID (volume ID) command is used to change or display the volume ID or change the access-protect indicator of a diskette device. When the)VOLID command is issued without any parameters, *volume ID owner ID record size* is displayed. When the)VOLID command is issued with parameters, WAS *volume ID owner ID record size* is displayed.

Note: The record size cannot be changed by the)VOLID command.

Syntax

)VOLID [device number] [volume ID] [owner ID] [access-protect indicator]

Where:

device number (optional) is the number of the diskette device.

volume ID (optional) will be the new identifier for the diskette volume. This parameter can be up to 6 alphabetic and numeric characters.

owner ID (optional) is the new identifier for the owner of the diskette volume. This parameter can be up to 14 alphabetic and numeric characters. If this parameter is not specified, the owner ID is not changed.

access-protect indicator (optional) is specified as either ON or OFF. This parameter can be specified only when the *volume ID* and *owner ID* parameters are also specified. The access-protect indicator prevents unauthorized access to the diskette volume; any attempt to access the diskette generates an ERROR 043 message.

Following are examples of using the)VOLID command:

)VOLID	Displays the volume ID, owner ID, and record size of the diskette on diskette drive 1.
)VOLID 12	Displays the volume ID, owner ID, and record size of the diskette on diskette drive 2.
)VOLID 11 NEW	Changes the volume ID of the diskette on diskette drive 1 to NEW; the owner ID remains the same.
)VOLID 11 NEW JOHN	Changes the owner ID of the diskette on diskette drive 1 to JOHN; the volume ID is also changed if the volume ID was not already NEW.
)VOLID 11 NEW JOHN ON	Sets the access-protect indicator on for the diskette on diskette drive 1.

The)WSID Command

The)WSID (workspace ID) command is used to change or display the device/file number and workspace ID of the active workspace. The)WSID command is also used to change or assign the security password. If a)SAVE or)CONTINUE command is specified without any parameters, the contents of the active workspace are written to the device and file number specified by the)WSID command. When the)WSID command is issued without any parameters, *device/file number workspace ID* is displayed. When the)WSID command is issued with parameters, WAS *device/file number workspace ID* is displayed.

Note: The)WSID command affects only the active workspace; it cannot be used to change any information on tape or diskette.

Syntax

)WSID [device/file number] [workspace ID] [:password]

Where:

device/file number (optional) is an integer that specifies the device/file number where the active workspace will be stored when either the)SAVE or)CONTINUE command is issued.

Note: If this parameter is omitted, the device/file number is cleared; a)SAVE or)CONTINUE command will not work unless a device/file number is specified in that)SAVE or)CONTINUE command.

workspace ID (optional) will be the new name for the active workspace. This parameter must be entered if any other parameter is used.

:password (optional) is any combination of up to 8 alphabetic or numeric characters (without blanks), preceded by a colon. These characters will become the security password for the tape file when the active workspace is written on tape.

Chapter 4. Primitive (Built-In) Functions

APL functions are of two types: user-defined and built-in. User-defined functions are discussed in Chapter 6. Built-in functions, called primitive functions, are denoted by a symbol and operate on the data you supply to them.

The value or values you supply are called arguments. Primitive functions that use two arguments, such as A÷B, are said to be dyadic; functions that use one argument are said to be monadic, such as ÷B, which yields the reciprocal of B. Arguments can be single data items (scalars), strings of data (vectors), tables of data (matrices), or multiple tables of data (N-rank arrays). Arguments can also be expressions or user-defined functions that result in a scalar, vector, matrix, or N-rank array.

There are two types of primitive functions: scalar functions and mixed functions. There are also operators that operate on the primitive functions. Examples of the functions and operators are provided throughout this chapter for easy reference and are set up as they would appear on the display.

PRIMITIVE SCALAR FUNCTIONS

Scalar functions operate on scalar arguments and arrays. They are extended to arrays element by element. The shape and rank of the result depend on the shape and rank of the arguments. For dyadic scalar functions, the relation between the types of arguments and the shape of the result is shown in the following table. Each scalar function is described following the table:

Argument A	**Argument B**	**Result**
Scalar	Scalar	Scalar
Array	Array with the same shape as A	Array with the same shape as the arguments
Scalar or one-element array	Array of any shape	Array with the same shape as argument B
Array of any shape	Scalar or one-element array	Array with the same shape as argument A
One-element array	One-element array with the rank different from the rank of A	One-element array with the shape of the array with the greater rank

The + Function: Conjugate, Plus

Monadic (One-Argument) Form: Conjugate $+B$

The *conjugate* function does not change the argument. The argument can be a numeric scalar, vector, or other array, and the shape of the result is the same as that of the argument:

```
        +5
5
        A←¯5
        +A
¯5
```

If B is an array, the function is extended to each of the elements of B. The shape of the result is the shape of B:

```
        B←2 3⍴¯3 ¯2 ¯1 0 1 2
        B
¯3 ¯2 ¯1
 0  1  2
        +B
¯3 ¯2 ¯1
 0  1  2
```

Dyadic (Two-Argument) Form: Plus $A+B$

The *plus* function results in the sum of the two arguments. The arguments can be numeric scalars, vectors, or other arrays. Arguments must be the same shape, unless one of the arguments is a scalar or single-element array. If the arguments have the same shape, the result has the same shape as the arguments:

```
        3+3
6
        3+2.73
5.73
        2.6+¯3.8
¯1.2
        5.1 1 ¯1 ¯3+5.1 2 0 4
10.2 3 ¯1 1
```

If one argument is a scalar or single-element array, the shape of the result is the same as that of the other input argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3+B
4 5 6
7 8 9
      B+3
4 5 6
7 8 9
```

The - Function: Negation, Minus

Monadic (One-Argument) Form: Negation -B

The *negation* function changes the sign of the argument. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      A←¯1 ¯3
      A
¯1 ¯3
      -A
1 3
```

If the argument is an array, the function is extended to each element of the array.

```
      B←2 3⍴¯3 ¯2 ¯1 0 1 2
      B
¯3 ¯2 ¯1
 0  1  2
      -B
 3  2  1
 0 ¯1 ¯2
```

Dyadic (Two-Argument) Form: Minus A-B

The *minus* function subtracts argument B from argument A. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape unless one of the arguments is a scalar or any single-element array. If the arguments are the same shape, the result has the same shape as the arguments:

```
      3-2
1
      4-5
¯1
      4-¯5
9
      5.1 1 ¯1 ¯3-5.1 2 0 ¯4
0 ¯1 ¯1 1
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other input argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3-B
 2  1  0
¯1 ¯2 ¯3
      B-3
¯2 ¯1  0
 1  2  3
```

The × Function: Signum, Times ÷ ×

Monadic (One-Argument) Form: Signum ×B

The *signum* function indicates the sign of the argument. If the argument is negative, ¯1 is the result; if the argument is zero, then 0 is the result; if the argument is positive, 1 is the result. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      ×¯25 0 33
¯1 0 1
```

If the argument is an array, the function is extended to each of the elements:

```
      B←2 3⍴¯2 ¯1 0 1 2 3
      B
¯2 ¯1  0
 1  2  3
      ×B
¯1 ¯1  0
 1  1  1
```

Dyadic (Two-Argument) Form: Times $A \times B$

The *times* function result is the product of argument A times argument B. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      2×2.1
4.2
      3×¯6
¯18
      0 2 4×3 6.1 ¯4
0 12.2 ¯16
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other input argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3×B
 3  6  9
12 15 18
```

The ÷ Function: Reciprocal, Divide

Monadic (One-Argument) Form: Reciprocal ÷*B*

The *reciprocal* function result is the reciprocal of the argument. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      ÷4
0.25
      ÷2
0.5
```

If the argument is an array, the function is extended to each of the elements:

```
   B←2 2⍴2 .5
   B
2             0.5
2             0.5
   ÷B
0.5           2
0.5           2
```

Dyadic (Two-Argument) Form: Divide A÷*B*

The *divide* function result is the quotient when argument A is divided by argument B. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape unless one of the arguments is a scalar or a single-element array. Arguments of the same shape have the same shape result:

```
      6÷3
2
      ¯3÷¯2
1.5
      10 9 4÷5 3 4
2 3 1
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other input argument. The single element is applied to every element of the multielement array:

```
      B←2 2ρ1 10 20 100
      B
 1  10
20 100
      3÷B
 3              0.3
 0.15           0.03
```

Note: There are two additional rules that apply to the divide function:

1. When zero is divided by zero, the result is 1:

```
      0÷0
1
```

2. Any value other than zero cannot be divided by zero:

```
      3÷0
DOMAIN ERROR
      3÷0
       ^
```

The ⌈ Function: Ceiling, Maximum [⌈ S]

Monadic (One-Argument) Form: Ceiling ⌈B

The *ceiling* function result is the next integer larger than the argument (the argument is rounded up), unless the argument already is an integer. In this case, the result is the same as the argument. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      ⌈3.5 ¯3.5
4 ¯3
      ⌈4
4
```

If the argument is an array, the function is extended to each of the elements:

```
      B←2 2⍴1 1.3 1.5 2
      B
    1           1.3
    1.5         2
      ⌈B
1 2
2 2
```

Note: The result of the ceiling function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).

Dyadic (Two-Argument) Form: Maximum A⌈B

The *maximum* function result is the larger of the arguments. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      4⌈6
6
      3⌈2
3
      ¯6⌈¯10
¯6
      5.1 1 ¯1 ¯3⌈5.1 2 0 ¯4
5.1 2 0 ¯3
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3⌈B
3 3 3
4 5 6
```

The ⌊ Function: Floor, Minimum

Monadic (One-Argument) Form: Floor ⌊*B*

The *floor* function result is the next integer smaller than the argument (the argument is rounded down) unless the argument is already an integer. In this case, the result is the same as the argument. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      ⌊3.9 ¯2.3
3 ¯3
      ⌊4
4
```

If the argument is an array, the function is extended to each of the elements:

```
      B←2 2⍴1 1.5 1.6 2
      B
     1              1.5
     1.6            2
      ⌊B
1 1
1 2
```

Note: The result of the floor function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).

Dyadic (Two-Argument) Form: Minimum A⌊*B*

The *minimum* function result is the smaller of the arguments. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      4⌊6
4
      3⌊2
2
      ¯6⌊¯10
¯10
      5.1 1 ¯1 ¯3⌊5.1 2 0 ¯4
5.1 1 ¯1 ¯4
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3⌊B
1 2 3
3 3 3
```

The | Function: Magnitude, Residue

Monadic (One-Argument) Form: Magnitude |B

The *magnitude* function result is the absolute value of the argument. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      |7.9
7.9
      |¯3
3
```

If the argument is an array, the function is extended to each of the elements:

```
    B←2 2⍴¯5.1 ¯1 0 3.14
    B
¯5.1          ¯1
 0             3.14
    |B
 5.1           1
 0             3.14
```

Dyadic (Two-Argument) Form: Residue A | B

The *residue* function result (when both argument A and argument B are positive) is the remainder when argument B is divided by argument A. The following rules apply when using the residue function:

1. If argument A is equal to zero, then the result is equal to argument B:

```
      0|6
6
```

2. If argument A is not equal to zero, then the result is a value between argument A and zero (the result can be equal to zero, but not equal to argument A). The result is obtained as follows:

   a. When argument B is positive, the absolute value of argument A is subtracted from argument B until a value between argument A and zero is reached:

```
      3|5
2
```

   b. When argument B is negative, the absolute value of argument A is added to argument B until a value between argument A and zero is reached:

```
      3|¯5
1
```

The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      3|7
1
      3|6
0
      6|3
3
      0|7
7
      7|0
0
      ¯2|12.3
¯1.7
      ¯2|¯12.3
¯0.3
      2|¯12.3
1.7
      1|2.385
0.385
      1|¯2.385
0.615
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3|B
1 2 0
1 2 0
```

The * Function: Exponential, Power [*/P]

Monadic (One-Argument) Form: Exponential *B

The *exponential* function result is the Naperian base e (2.718281828459045) raised to the power indicated by the argument. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      *1
2.7183
      *3
20.086
```

If the argument is an array, the function is extended to each element of the array:

```
      B←2 2⍴0 1 2 3
      B
0 1
2 3
      *B
    1           2.7183
    7.3891     20.086
```

*Dyadic (Two-Argument) Form: Power A*B*

The *power* function result is argument A raised to the power indicated by argument B. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape unless one of the arguments is a scalar, or any single-element array. Arguments of the same shape have the same shape result:

```
      2*3
8
      ¯.5*2
0.25
      3*0
1
      9*.5
3
      2*¯3
0.125  ←——————————————  2*¯3 = 1/2³ = 1/8 = .125
```

To find the root of a number, you raise the number to the power indicated by the reciprocal of the root. For example, to find the square root:

```
      1 4 9 16*÷2
1 2 3 4
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 2ρ1 2 3 4
      B
1 2
3 4
      B*2
     1           4
     9          16
```

The ⍟ Function: Natural Log, Logarithm [○ O] [* P]

You form the ⍟ symbol by overstriking the ○ symbol and the * symbol.

Monadic (One-Argument) Form: Natural Log ⍟B

The *natural log* function result is the log of the argument B to the Naperian base e (2.718281828459045). The argument can be a nonnegative numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
      ⍟2.7183
1
      ⍟20.086
3
```

If the argument is an array, the function is extended to each element of the array:

```
      B←2 2⍴1 3 10 20
      B
 1  3
10 20
      ⍟B
      0               1.0986
      2.3026          2.9957
```

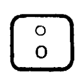
Dyadic (Two-Argument) Form: Logarithm A⍟B

The *logarithm* function result is the log of argument B to the base of argument A. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      2⍟8
3
      3.1⍟12.8
2.2534
      2 3 4⍟8 9 16
3 2 2
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 2⍴1 2 3 4
      B
1 2
3 4
      10⍟B
   0               0.30103
   0.47712         0.60206
```

The ○ Function: Pi Times, Circular 

Monadic (One-Argument) Form: Pi Times ○B

The *pi times* function result is the value of pi (3.141592653589793) times B. The argument can be a numeric scalar, vector, or other array. The shape of the result is the same as that of the argument.

```
      ○1
3.1416
      ○3
9.4248
```

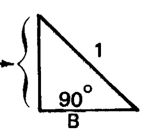
If the argument is an array, the function is extended to each element of the array:

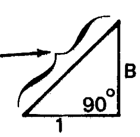
```
      B←2 2⍴1 2 3 4
      B
1 2
3 4
      ○B
    3.1416        6.2832
    9.4248       12.566
```

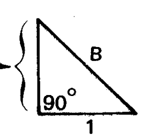
Dyadic (Two-Argument) Form: Circular A○B

The *circular* function result is the value of the specified trigonometric function (argument A) for the specified radians (argument B). The arguments can be numeric scalars, vectors, or other arrays. Arguments must be the same shape, unless one is a scalar or single-element array. Arguments of the same shape have the same shape result. The following is a list of the values for the A argument and the related functions performed. A negative argument A is the mathematical inverse of a positive argument A; any values for argument A other than the following will result in DOMAIN ERROR:

Value of A	**Operation Performed**
0○B	(1-B⋆2)⋆.5
1○B	Sine B
2○B	Cosine B
3○B	Tangent B
4○B	(1+B⋆2)⋆.5
5○B	Hyperbolic sine of B (sinh B)
6○B	Hyperbolic cosine of B (cosh B)
7○B	Hyperbolic tangent of B (tanh B)
¯1○B	Arcsin B
¯2○B	Arccos B
¯3○B	Arctan B
¯4○B	(¯1+B⋆2)⋆.5
¯5○B	Arcsinh B
¯6○B	Arccosh B
¯7○B	Arctanh B







If B is 45°, here is how to solve for the sine, cosine, and tangent of B (45° is equivalent to pi radians divided by 4):

```
      B←○÷4
      B                      The left argument specifies
0.7854                       the trigonometric function.
      1○B
0.70711                      Sine of B
      2○B
0.70711                      Cosine of B
      3○B
1                            Tangent of B
```

If B is the sine of an angle, then 0○B yields the cosine of the same angle, and conversely, if B is the cosine, 0○B yields the sine. Suppose you wanted the sine of 30°, which is equivalent to pi divided by 6:

```
      B←1○(○÷6)
      B
0.5                          Sine of 30°
      0○B
0.86603                      Cosine of 30°
      B←2○(○÷6)
      B
0.86603                      Cosine of 30°
      0○B
0.5                          Sine of 30°
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      A←2 2ρ1 2 3 4
      B←○÷4
      A
 1 2
 3 4
      B
0.7854
      A○B
    0.70711     0.70711
    1           1.2716
```

The ! Function: Factorial, Binomial

You form the ! symbol by overstriking the quotation mark (') and the period (.).

Monadic (One-Argument) Form: Factorial !B

The *factorial* function result is the product of all the positive integers from one to the number value of the argument. The argument can be a positive numeric scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
          !4
24
          1×2×3×4
24
          !1 2 3 4 5
1 2 6 24 120
```

The factorial function also works with decimal numbers and zero, but negative integers are not allowed. When used in this way, factorial can be defined by use of the mathematical gamma function–(! A) is equal to gamma (A+1):

```
          !3.14
7.1733
          !0
1
```

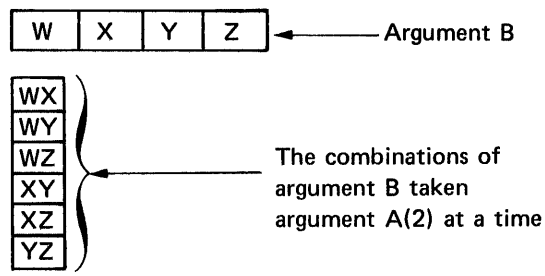
If the argument is an array, the function is extended to each of the elements:

```
          B←2 2⍴0 1 2 3
          B
0 1
2 3
          !B
      1           1
      2           6
```

Dyadic (Two-Argument) Form: Binomial A!B

The *binomial* function result is the number of different combinations of argument B that can be taken A at a time. The result of A!B is also the $(A+1)^{th}$ coefficient of the binomial expansion of the B^{th} power. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      2!4
6
      2!6
15
      3!0
0
      0!3
1
      2!3
3
```



If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      0 1 2 3!3
1 3 3 1
      0 1 2 3 4!4
1 4 6 4 1
      B←2 2⍴0 1 2 3
      B
 0 1
 2 3
      B!5
     1           5
    10          10
      2!¯4 5 4.5 4
10 10 7.875 6
```

If noninteger arguments are used, this function relates to the beta function as follows: Beta (P,Q) is equal to ÷Q×(P-1)!P+Q-1

The ? Function: Roll [? Q]

Monadic (One-Argument) Form: Roll ?B

The *roll* function result is a randomly selected integer from 0 through B-1 or 1 through B (depending on the index origin). Each integer in the range has an equal chance of being selected. The argument can be a positive integral scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
          ?300
202
          ?300
3
          ?5 7 9
2 1 4
          ?6 6
5 4
          ?6 6
6 6
```

If the argument is an array, the function is extended to each element of the array:

```
        B←2 3⍴11 22 33 44 55 66
        B
11 22 33
44 55 66
        ?B
 5 12 28
 2  3 35
```

Dyadic (Two-Argument) Form

See the *deal* function later in this chapter under *Primitive Mixed Functions*.

The ∧ Function: And $\boxed{\wedge / 0}$

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: And A∧B

The *and* function result is 1 when A and B are both 1; otherwise, the result is 0. The value of the arguments must be either 0 or 1. The arguments can be scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      0∧1
0
      1∧1
1
      1∧0
0
      0 0 1 1∧0 1 0 1
0 0 0 1
```

And Table

Operator

∧	0	1
0	0	0
1	0	1

Argument A

Argument B

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 2⍴0 1 1 0
      B
0 1
1 0
      1∧B
0 1
1 0
```

The ∨ Function: Or

Monadic (One-Argument) Form

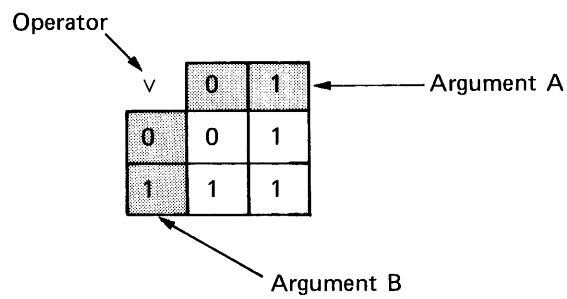
There is no monadic form.

Dyadic (Two-Argument) Form: Or A∨B

The *or* function result is a 1 when either or both arguments are 1; otherwise, the result is 0. The values of the arguments must be 1 or 0. The arguments can be scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

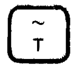
Or Table

```
      1∨0
1
      0∨0
0
      0 0 1 1∨0 1 0 1
0 1 1 1
```



If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 2⍴0 1 0 1
      B
0 1
0 1
      1∨B
1 1
1 1
```

The ~ Function: Not 

Monadic (One-Argument) Form: Not ~B

The *not* function result is 1 when B is 0 and 0 when B is 1. The values of the argument must be 1 or 0. The argument can be a scalar, vector, or other array. The shape of the result is the same as that of the argument:

```
        ~0
1
        ~1
0
```

If the argument is an array, the function is extended to each element of the array:

```
        B←2 3⍴0 1
        B
0 1 0
1 0 1
        ~B
1 0 1
0 1 0
```

Dyadic (Two-Argument) Form

There is no dyadic form.

The ⍲ Function: Nand

∧ 0 ~ T

You form the ⍲ symbol by overstriking the *and* (∧) and the *not* (~) symbols.

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Nand A⍲B

The *nand* function result is 0 when both A and B are 1; otherwise, the result is 1. The values of the arguments must be 1 or 0. The arguments can be scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      0⍲1
1
      1⍲1
0
      0 0 1 1⍲0 1 0 1
1 1 1 0
```

Nand Table

Operator → ⍲ ; Argument A (top row) ; Argument B (left column)

⍲	0	1
0	1	1
1	1	0

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 2⍴0 1
      B
0 1
0 1
      1⍲B
1 0
1 0
```

The ⍱ Function: Nor [∨ 9] [~ T]

You form the ⍱ symbol by overstriking the *or* (∨) and the *not* (~) symbols.

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Nor A⍱B

The *nor* function result is 1 when A and B are both 0; otherwise, the result is 0. The values of the arguments must be 1 or 0. The arguments can be scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      1⍱0
0
      0⍱0
1
      0 0 1 1⍱0 1 0 1
1 0 0 0
```

Nor Table

Operator → ⍱ ← Argument A

⍱	0	1
0	1	0
1	0	0

Argument B

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 2⍴0 1
      B
0 1
0 1
      0⍱B
1 0
1 0
```

The > Function: Greater Than

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Greater Than A>B

The *greater than* function result is 1 when argument A is greater than argument B; otherwise, the result is 0. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
        1.56>2
0
        ¯2>0
0
        ¯3>¯2
0
        0>¯4.4
1
        1.123>1.123
0
        5.1 1 ¯1 ¯3>5.1 2 0 ¯4
0 0 0 1
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
       B←2 3ρ1 2 3 4 5 6
       B
1 2 3
4 5 6
       3>B
1 1 0
0 0 0
```

Notes:

1. The result of the > function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).
2. When the > function is used with logical data (see Chapter 2), a system error display can occur if the number of elements in the right argument is a multiple of 32. The following are examples:

```
0>Y←32ρ1

A←B←70ρ1
(64↑A)>B[⍳64]
```

To prevent this error, separate the statement into two statements as follows:

```
Y←32⍴1
0>Y

A←B←70⍴1
Y←B[⍳64]
(64↑A)>Y
```

First, assign the right argument for the > function to a variable.

Then, use the variable as the right argument for the > function.

The = Function: Equal To $\boxed{\begin{matrix}=\\5\end{matrix}}$

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Equal To A=B

The *equal to* function result is 1 when the value of argument A equals the value of argument B; otherwise, the result is 0. The arguments (numeric or character) can be scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any other single-element array. Arguments of the same shape have the same shape result:

```
      0=5
0
      1.654321=1.654321
1
      1='A'
0
      'A'='B'
0
      '1'=1
0
      1 3 5 7=2 3 4 5
0 1 0 0
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      'A'='ABACADAEAFAG'
1 0 1 0 1 0 1 0 1 0 1 0
```

Notes:

1. If the arguments are numeric, the result of the = function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).
2. When the = function is used with logical data (see Chapter 2), a system error display can occur if the number of elements in the right argument is a multiple of 32. The following are examples:

```
0=Y←32⍴1


A←B←70⍴1
(64↑A)=B[⍳64]
```

To prevent this error, separate the statements as follows:

```
Y←32⍴1
0=Y

A←B←70⍴1
Y←B[⍳64]
(64↑A)=Y
```

First,, assign the right argument for the = function to a variable.

Then, use the variable as the right argument for the = function.

The < Function: Less Than [< 3]

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Less Than A<B

The *less than* function result is 1 when argument A is less than argument B; otherwise, the result is 0. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      1.65<2
1
      ¯2<0
1
      ¯3<¯2
1
      0<¯4.4
0
      1.123<1.123
0
      5.1 1 ¯1 ¯3<5.1 2 0 ¯4
0 1 1 0
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3<B
0 0 0
1 1 1
      B<3
1 1 0
0 0 0
```

Notes:

1. The result of the < function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).
2. When the < function is used with logical data (see Chapter 2), a system error display can occur if the number of elements in the right argument is a multiple of 32. The following are examples:

```
0<Y←32⍴1

A←B←70⍴1
(64↑A)<B[⍳64]
```

To prevent this error, separate the statement into two statements as follows:

```
Y←32⍴1
0<Y

A←B←70⍴1
Y←B[⍳64]
(64↑A)<Y
```

First, assign the right argument for the < function to a variable.

Then, use the variable as the right argument for the < function.

The ≥ Function: Greater Than or Equal To ≥ 6

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Greater Than or Equal To A≥B

The ***greater than or equal to*** function result is 1 when argument A is greater than or equal to argument B; otherwise, the result is 0. The arguments can be numeric scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      1.65≥2
0
      ¯2≥0
0
      2≥2
1
      5.1 1 ¯1 ¯3≥5.1 2 ¯3 3
1 0 1 0
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 3ρ1 2 3 4 5 6
      B
1 2 3
4 5 6
      3≥B
1 1 1
0 0 0
```

Notes:

1. The result of the ≥ function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).
2. When the ≥ function is used with logical data (see Chapter 2), a system error display can occur if the number of elements in the right argument is a multiple of 32. The following are examples:

```
0≥Y←32ρ1

A←B←70ρ1
(64↑A)≥B[⍳64]
```

To prevent this error, separate the statement into two statements as follows:

```
Y←32ρ1
0≥Y

A←B←70ρ1
Y←B[⍳64]
(64↑A)≥Y
```

First, assign the right argument for the ≥ function to a variable.

Then, use the variable as the right argument for the ≥ function.

The ≤ Function: Less Than or Equal To

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Less Than or Equal To A≤B

The *less than or equal to* function result is 1 when argument A is less than or equal to argument B; otherwise, the result is 0. The arguments can be numeric scalars, vectors or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      1.65≤2
1
      ¯2≤0
1
      ¯3≤¯2
1
      0≤¯2
0
      5.1 1 ¯1 ¯3≤5.1 3 ¯2 ¯5
1 1 0 0
```

If one argument is a scalar or a single-element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3≤B
0 0 1
1 1 1
```

Notes:

1. The result of the ≤ function depends on the ⎕CT system variable (see
2. When the ≤ function is used with logical data (see Chapter 2), a system error display can occur if the number of elements in the right argument is a multiple of 32. The following are examples:

```
0≤Y←32⍴1

A←B←70⍴1
(64↑A)≤B[⍳64]
```

To prevent this error, separate the statement into two statements as follows:

```
Y←32ρ1
0≤Y

A←B←70ρ1
Y←B[⍳64]
(64↑A)≤Y
```

First, assign the right argument for the ≤ function to a variable.

Then, use the variable as the right argument for the ≤ function.

The ≠ Function: Not Equal To [≠ 8]

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Not Equal To A≠B

The *not equal to* function result is 1 when argument A is not equal to argument B; otherwise, the result is 0. The arguments (numeric or character) can be scalars, vectors, or other arrays. The arguments must be the same shape, unless one of the arguments is a scalar or any single-element array. Arguments of the same shape have the same shape result:

```
      0≠5
1
      1.123≠1.123
0
      'A'≠'A'
0
      '1'≠1
1
      5.1 1 ¯1 ¯3≠5.1 2 0 ¯4
0 1 1 1
```

If one argument is a scalar or a single element array, the shape of the result is the same as that of the other argument. The single element is applied to every element of the multielement array:

```
      'A'≠'ABACADAEAFAG'
0 1 0 1 0 1 0 1 0 1 0 1
```

Notes:

1. If the arguments are numeric, the result of the ≠ function depends on the ⎕CT system variable (see Chapter 5 for information on the ⎕CT system variable).

   The *not equal to* function can also be used as an *exclusive or* function. When used in this manner, the value of the arguments must be either 0 or 1:

   Exclusive Or Table

   Operator → ≠ ; Argument A (top row) ; Argument B (left column)

≠	0	1
0	0	1
1	1	0

2. When the ≠ function is used with logical data (see Chapter 2), a system error display can occur if the number of elements in the right argument is a mulltiple of 32. The following are examples:

```
0≠Y←32⍴1

A←B←70⍴1
(64↑A)≠B[⍳64]
```

   To prevent this error, separate the statement into two statements as follows:

```
Y←32⍴1
0≠Y

A←B←70⍴1
Y←B[⍳64]
(64↑A)≠Y
```

   First, assign the right argument for the ≠ function to a variable.

   Then, use the variable as the right argument for the ≠ function.

PRIMITIVE MIXED FUNCTIONS

The mixed functions differ from scalar functions because the shape of their results depends on the particular mixed function rather than exclusively on the shape of the arguments. The following list gives a brief description of each of the mixed functions. Following the list, each function is discussed in detail:

Monadic Mixed Functions	**Name**	**Result**
⍴B	Shape	The length of each coordinate of the argument.
,B	Ravel	A vector containing the elements of B in the order they exist in the rows of B.
⍋B	Grade up	The index values that would select the elements of B in ascending order.
⍒B	Grade down	The index values that would select the elements of B in descending order.
⍳B	Index generator	B consecutive integers starting from the index origin.
⌽B or ⌽[1]B or ⊖B	Reverse	The elements of the argument are reversed.
⍉B	Transpose	The coordinates of the argument are reversed.
⌹B	Matrix inverse	The inverse of a square matrix or the pseudoinverse of a rectangular matrix.
⍎B	Execute	Argument B executed as an expression.
⍕B	Format	Argument B converted to a character array.

Dyadic Mixed Functions	Name	Result
A⍴B	Reshape (structure)	An array of a shape specified by A, using elements from B.
A,B or A,[I]B	Catenate	The two arguments joined along an existing coordinate ([I] is a positive integer).
A,[I]B	Laminate	The two arguments joined along a new coordinate ([I] is a fraction).
A/B or A/[I]B or A⌿B	Compress	The elements from B that correspond to the 1's in A.
A\B or A\[I]B or A⍀B	Expand	B is expanded to the format specified by A; 1 in A inserts an element from B; a 0 in A inserts a 0 or blank element.
A↑B	Take	The number of elements specified by A are taken from B.
A↓B	Drop	The number of elements specified by A are dropped from B.
A⍳B	Index of	The first occurrence in A of the elements in B.
A⌽B or A⌽[I]B or A⊖B	Rotate	The elements of B are rotated as specified by A. If A is positive, the elements of B are rotated to the left. If A is negative, the elements of B are rotated to the right.
A⍉B	Generalized transpose	The coordinates of B interchanged as specified by A.
A?B	Deal	The number of elements specified by A are randomly selected from B; the same number is never selected twice.

Dyadic Mixed Functions	**Name**	**Result**
A⊥B	Decode (base value)	The value of argument B expressed in the number system specified by argument A.
A⊤B	Encode (representation)	The representation of argument B in the number system specified by argument A.
A∊B	Membership	A 1 for each element of A that can be found in B and a 0 for each element not found.
A⌹B	Matrix divide	Solution to one or more sets of linear equations with coefficient matrix (matrices) B and right-hand sides A or the least squares solution to one or more sets of linear equations.
A⍕B	Format	Argument B converted to a character array in the format specified by argument A.

Note: The mixed functions *reverse*, *rotate*, *compress*, and *expand*, and the operators (see *APL Operators* later in this chapter) *reduction* and *scan* can be applied to a specific coordinate of an array. This requires the use of an index entry [I] that indicates the coordinate to which the mixed function or operator is applied. The value of the index entry can be from 1 to the number of coordinates in the array; the leftmost coordinate (first coordinate) has an index value of 1, the next coordinate has an index value of 2, and so on. A matrix, for example, has an index value of 1 for the row coordinate and an index value of 2 for the column coordinate. If an index entry is not specified, the last coordinate (columns) is assumed. If a - (minus) symbol is overstruck with the function symbol or operator symbol, the first coordinate is assumed (unless an index value was also used). When a function or operator is applied to a specific coordinate, the operation takes place between corresponding elements in the specified coordinate. For example, assume you have a three-rank array:

- When the first coordinate (planes) is specified, the operation takes place between corresponding elements in each plane.
- When the second coordinate (rows) is specified, the operation takes place between the corresponding elements in each row per plane.
- When the third coordinate (columns) is specified, the operation takes place between the corresponding elements in each column per plane.

The ρ Function: Shape, Reshape (Structure)

Monadic (One-Argument) Form: Shape ρB

The *shape* function result is the shape of the argument; it has one element for each coordinate of the argument, which indicates the length of that coordinate. The argument can be any variable or constant:

```
        ρ'ABCD'            A Vector with Four Elements
4
        ρ1 2
2
        ρ1 2 3
3
        A←2 3ρ1 2 3 4 5 6
        A
 1 2 3
 4 5 6
        ρA
2 3
```

The shape function applied to a scalar yields an empty vector, because a scalar has no coordinates. An empty vector is indicated by a blank result line:

```
ρ2
                          Blank Result Lines
ρ'T'

```

The instruction ρρB yields the rank (shape of the shape, or number of coordinates) of B:

```
        B←2 2 3ρ'CARBARFARARE'
        B
CAR
BAR

FAR
ARE
        ρB
2 2 3
        ρρB
3
```

Dyadic (Two-Argument) Form: Reshape (Structure) $A \rho B$

The *reshape* function forms an array of the shape specified by argument A using element(s) from argument B. The elements of argument B are placed into the array in row order. If there are not enough elements in argument B to fill the array, the elements are repeated. If there are more elements in argument B than are required to fill the array, only the required number of elements are used. Argument A must be a nonnegative integer or vector of nonnegative integers. The number of elements in argument A is equal to the number of coordinates, or the rank, of the result. Argument B can be any variable or constant. If all of the elements of argument A are nonzero, then B cannot be an empty array:

```
      2 3⍴1 2 3 4 5 6
 1 2 3
 4 5 6
      4 2⍴'ABCDEFGH'
AB
CD
EF
GH
      5⍴'MOUSETRAP'
MOUSE
      3 4⍴123
 123 123 123 123
 123 123 123 123
 123 123 123 123
      A←4 2⍴1 2 3 4 5 6 7 8
      A
 1 2
 3 4
 5 6
 7 8
      2 3⍴A
 1 2 3
 4 5 6
      6⍴'¨'
¨¨¨¨¨¨
```

The , Function: Ravel, Catenate, Laminate $\boxed{;\ ,}$

Monadic (One-Argument) Form: Ravel **,** *B*

The *ravel* function results in a vector containing the element of argument B. If argument B is an array, the elements in the vector are taken from argument B in row order. Argument B can be a scalar, vector, or other array. The resulting vector contains the same number of elements as argument B:

```
      A←2 2 2⍴1 2 3 4 5 6 7 8
      A
 1 2
 3 4

 5 6
 7 8
      ,A
1 2 3 4 5 6 7 8
      B←2 3⍴'ABCDEF'
      B
ABC
DEF
      ,B
ABCDEF
```

*Dyadic (Two-Argument) Form: Catenate or Laminate A***,***[I]B*

The function is *catenate* when the [I] entry (index entry) is an integer and *laminate* when the [I] entry is a fraction.

Catenate (The Index [I] Entry Is an Integer)

The *catenate* function joins two items along an existing coordinate. (See the *laminate* function following for a description of how to join two items along a new coordinate.) The index [I], if given, specifies which coordinate is expanded. The index entry must be a positive scalar or one-element array. If no index [I] is specified, the last coordinate is used. Matrices of unequal sizes can be joined, providing the lengths of the coordinate *not* specified are the same (see *Catenation* in Chapter 3):

```
      A←1 4
      B←7 9 5
      A,B
1 4 7 9 5
      A←2 3⍴1 2 3 4 5 6
      B←2 3⍴7 8 9 10 11 12
      A
1 2 3
4 5 6
      B
 7  8  9
10 11 12
      A,B
 1  2  3  7  8  9
 4  5  6 10 11 12
      A,[1]B
 1  2  3
 4  5  6
 7  8  9
10 11 12
      A,[2]B
 1  2  3  7  8  9
 4  5  6 10 11 12
      A,[2]10 20
 1  2  3 10
 4  5  6 20
      10 20 30,[1]A
10 20 30
 1  2  3
 4  5  6
```

Laminate (The Index [I] Entry Is a Fraction)

The *laminate* function joins two items by creating a new coordinate specified by the index entry [I], which must be a positive fraction. If the index entry is between 0 and 1, the new coordinate becomes the first coordinate; if the index entry is between 1 and 2, the new coordinate is placed between existing coordinates 1 and 2 (the new coordinate that is added always has a value [or length] of 2). The following chart shows the positions of a new coordinate in the shape vector (see the following examples) when two 3 by 3 matrices are laminated:

Index Value	Positions of New Coordinate in the Shape Vector
.1 - .9	2 3 3
1.1 - 1.9	3 2 3
2.1 - 2.9	3 3 2

Lamination requires either that arguments A and B are the same shape or that one of the arguments is a scalar:

```
      A←3 3⍴1 2 3 4 5 6 7 8 9
      B←3 3⍴11 22 33 44 55 66 77 88 99
      A
1 2 3
4 5 6
7 8 9
      B
11 22 33
44 55 66
77 88 99
      C←A,[.8]B
      C
 1  2  3
 4  5  6
 7  8  9

11 22 33
44 55 66
77 88 99
      ⍴C
2 3 3←————————————Shape Vector
```

```
        C←A,[1.5]B
        C
  1  2  3
 11 22 33

  4  5  6
 44 55 66

  7  8  9
 77 88 99
        ⍴C
3 2 3
```
← Shape Vector

```
        C←A,[2.1]B
        C
 1 11
 2 22
 3 33

 4 44
 5 55
 6 66

 7 77
 8 88
 9 99
        ⍴C
3 3 2
```
← Shape Vector

The following examples show the result when the two matrices in the preceding example are catenated instead of laminated:

```
       A,[1]B
  1  2  3
  4  5  6
  7  8  9
 11 22 33
 44 55 66
 77 88 99
       A,[2]B
  1  2  3 11 22 33
  4  5  6 44 55 66
  7  8  9 77 88 99
```

The / Function: Compress

Monadic (One-Argument) Form

See *Reduction Operator* later in this chapter under *APL Operators*.

Dyadic (Two-Argument) Form: Compress A/[I]B or A/B or A⌿B

The *compress* function selects elements from argument B corresponding in sequence to 1's in argument A. Argument A must be a logical scalar or vector having the values 0 or 1. Argument B can be any scalar, vector, or other array. Both arguments must have the same number of elements unless:

- One of the arguments is a scalar or single-element array.
- Argument B is a multidimensional array; then the number of elements in argument A must be the same as the length of the argument B coordinate being acted on.

When argument B is a multidimensional array, the [I] index entry is used to specify the coordinate that is acted on. If the index entry is omitted, the last coordinate (columns) is assumed. If the A⌿B form is used, the first coordinate is assumed. The rank of the result is the same as the rank of argument B:

```
      1 0 1 0 1/1 2 3 4 5
1 3 5
      1 1 1/1 2 3
1 2 3
      0 0 0/1 2 3
                                     Blank Display Line (empty array)
      B←3 4⍴1 2 3 4 5 6 7 8 9 10 11 12
      B
 1  2  3  4
 5  6  7  8
 9 10 11 12
      1 0 1/[1]B          The first coordinate (rows) is specified; the first
 1  2  3  4               and third rows, as specified by argument A, are
 9 10 11 12               selected.
      0 1 1 0/[2]B
 2  3                     The second coordinate (columns) is specified;
 6  7                     the second and third columns, as specified by
10 11                     argument A, are selected.
      1 0 1⌿B
 1  2  3  4
 9 10 11 12
      0 1 1 0/B
 2  3
 6  7
10 11
      1/B
 1  2  3  4
 5  6  7  8
 9 10 11 12
      0/B
                          Blank Display Line (empty array)
```

The \ Function: Expand [\]

Monadic (One-Argument) Form

See *Scan Operator* later in this chapter under *APL Operators*.

Dyadic (Two-Argument) Form: Expand A\[I]B or A\B or A⍀B

The result of the *expand* function is argument B expanded as indicated by argument A. Each 1 in argument A selects an element from argument B, and each 0 in argument A inserts a 0 (or blank for character data) in the result. Argument A must be a logical scalar or vector having the values 0 or 1. Argument B can be any scalar, vector, or other array. If argument B is a vector, argument A must have the same number of 1's as the number of elements in argument B. If argument B is a multidimensional array, argument A must have the same number of 1's as the length of the argument B coordinate being acted on.

When argument B is an array, the [I] index entry is used to specify the coordinate that is acted on. If the index entry is omitted, the last coordinate (columns) is assumed. If the A⍀B is used, then the first coordinate is assumed.

If argument B is a scalar or single-element array, it is extended to a length equal to the number of 1's in argument A. If argument B is not a scalar or single-element array, the rank of the result is the same as the rank of the B argument.

```
      1 0 1 1 0\1 2 3
1 0 2 3 0
      B←2 3⍴1 2 3 4 5 6
      B
 1 2 3
 4 5 6
      1 0 1\[1]B          The first coordinate (rows) is
 1 2 3                    expanded; a row is inserted between
 0 0 0                    the first and second row.
 4 5 6
      1 1 0 1\[2]B        The second coordinate (columns) is
 1 2 0 3                  expanded; a column is inserted between
 4 5 0 6                  the second and third columns.
      1 0 1⍀B
 1 2 3
 0 0 0
 4 5 6
```

The ⍋ Function: Grade Up [Δ H] [| M]

You form the ⍋ symbol by overstriking the Δ symbol and the | symbol.

Monadic (One-Argument) Form: Grade Up ⍋B

The *grade up* function result is the index values that would select the elements of argument B in ascending order. That is, the first element of the result is the index of the smallest element in argument B, the next element is the index of the next smallest element in argument B, and so on. Argument B must be a numeric vector. When two or more elements in the vector have the same numeric value, their position in the vector determines their order in the result (the index value of the first occurrence appears first in the output). The number of elements in the result is the same as the number of elements in the argument:

```
      ⍋3 1 5 2 4
2 4 1 5 3
      ⍋A←6 2 5 1 4 3
4 2 6 5 3 1
      ⍋3 6 3 1 5 2
4 6 1 3 5 2
```

The following example shows how the grade up function can be used to sort a vector into ascending order:

```
      A←14 12 16 18 15 11
      A [⍋A]
11 12 14 15 16 18
```

The result of the grade up function is not the reverse of the grade down function because of the way equal elements are handled; see *The ⍒ Function: Grade Down* for an example using the grade up and grade down functions with equal elements.

Note: The result of the ⍋ function depends on the ⎕IO system variable (see Chapter 5 for information on the ⎕IO system variable).

Dyadic (Two-Argument) Form

There is no dyadic form.

The ⍒ Function: Grade Down ∇ G | M

You form the ⍒ symbol by overstriking the ∇ symbol and the | symbol.

Monadic (One-Argument) Form: Grade Down ⍒B

The *grade down* function result is the index values that would select the elements of the numeric vector of argument B in descending order. That is, the first element of the result is the index of the largest element in argument B, the next element is the index of the next largest element in argument B, and so on. Argument B must be a numeric vector. When two or more elements in the vector have the same numeric value, their position in the vector determines their order in the result (the index value of the first occurrence appears first in the output). The number of elements in the result is the same as the number of elements in the argument:

```
      ⍒3 1 5 2 4
3 5 1 4 2
      ⍒A←6 2 5 1 4 3
1 3 5 6 2 4
      ⍒3 6 3 1 5 2
2 5 1 3 6 4
```

The following example shows how the grade down function can be used to sort a vector in descending order:

```
      A←14 12 16 18 15 11
      A[⍒A]
18 16 15 14 12 11
```

The following example shows how equal elements are handled when the grade up and grade down functions are used:

```
      A←5 2 8 7 3 4 10 1 2 3
      A
5 2 8 7 3 4 10 1 2 3
      ⍒A
7 3 4 1 6 5 10 2 9 8
      ⍋A
8 2 9 5 10 6 1 4 3 7
```

Positions 2 and 9 and 5 and 10 are equal.

Because the indices for the equal elements are in the same order (first occurrence first) for both the grade down and grade up functions, the grade down function is not the reverse of the grade up function:

```
      A[⍒A]
10 8 7 5 4 3 3 2 2 1
      A[⍋A]
1 2 2 3 3 4 5 7 8 10
```

Note: The result of the ⍒ function depends on the ⎕IO system variable (see Chapter 5 for information on the ⎕IO system variable).

Dyadic (Two-Argument) Form

There is no dyadic form.

The ↑ Function: Take

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Take A↑B

The *take* function result is the number of elements specified by argument A, taken from argument B. Argument B can be a scalar, vector, or other array. Argument A must be a scalar or vector of integers. If argument B is a vector, argument A must be a scalar. Argument A must be a vector with an element for each coordinate of argument B. When argument A is positive, the first elements of argument B are taken; when argument A is negative, the last elements are taken. If argument A specifies more elements than the number of elements in argument B, the result is padded with 0's (or blanks for character data). The shape of the result is the value of A:

```
        3↑1 2 3 4 5
1 2 3
        ¯3↑1 2 3 4 5
3 4 5
        7↑1 2 3 4 5
1 2 3 4 5 0 0
        ¯7↑1 2 3 4 5
0 0 1 2 3 4 5
        B←3 4⍴1 2 3 4 5 6 7 8 9 10 11 12
        B
  1  2  3  4
  5  6  7  8
  9 10 11 12
        2 3↑B
 1 2 3
 5 6 7

      B←2 2 3⍴1 2 3 4 5 6 7 8 9 10 11 12
      B
 1  2  3
 4  5  6

 7  8  9
10 11 12
      1 1 1↑B
1
      2 1 1↑B
1

7
      1 2 3↑B
1 2 3
4 5 6
      ¯1 2 3↑B
 7  8  9
10 11 12
```

The ↓ Function: Drop ↓ U

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Drop A↓B

The *drop* function result is the remaining elements of arguments B after the number of elements specified by argument A is dropped. Argument B can be a vector or other array. Argument A must be a scalar if argument B is a vector.

When argument B is an array, argument A must have one element for each coordinate of argument B. When argument A is positive, the first elements of argument B are dropped from the result; when argument A is negative, the last elements are dropped:

```
      3↓1 2 3 4 5
4 5
      ¯3↓1 2 3 4 5
1 2
      B←3 4⍴1 2 3 4 5 6 7 8 9 10 11 12
      B
 1  2  3  4
 5  6  7  8
 9 10 11 12
      1 2↓B
 7  8
11 12
      ¯1 ¯2↓B
1 2
5 6
```

The ⍳ Function: Index Generator, Index of

Monadic (One-Argument) Form: Index Generator ⍳B

The *index generator* function result is a vector containing the first B integers, starting with the index origin (see ⎕IO system variable in Chapter 5). The argument can be a nonnegative integer that is either a scalar or a single-element array:

```
      ⍳5
1 2 3 4 5
      A←⍳6
      A
1 2 3 4 5 6
      5+⍳5←——— Each of the generated integers is added to 5.
6 7 8 9 10
```

Dyadic (Two-Argument) Form: Index of A⍳B

The *index of* function result is the index of the first occurrence in argument A of the element(s) in argument B. Argument A must be a vector. Argument B can be a scalar, vector, or array. The result is the same shape as argument B. If the element in argument B cannot be found in argument A, the value of the index for that element is one greater than the largest index of A (⎕IO+⍴A):

```
          2 5 3 5 8⍳5
2                                    Second Element
          'ABCDEFG'⍳'C'
3
          A←11 22 33 44 55
          A⍳22
2
          A⍳23
6                                    not found
          A←9 2 8 8 2 6 4 8
          B←2 3⍴1 9 8 1 5 2
          B
 1 9 8
 1 5 2
          A⍳B
 9 1 3
 9 9 2
```

Note: The result of the ⍳ function depends on the ⎕IO system variable (see Chapter 5 for information on the ⎕IO system variable).

The ⌽ Function: Reverse, Rotate

You form the ⌽ symbol by overstriking the ○ symbol and the | symbol. A special form of the function symbol is ⊖, which you form by overstriking the ○ symbol and the - symbol.

Monadic (One-Argument) Form: Reverse ⌽[I]B or ⌽B or ⊖B

The *reverse* function reverses the elements of argument B. Argument B can be any expression.

When argument B is a multidimensional array, the index entry [I] can be used to specify the coordinate that is acted on. If the index entry is omitted, the last coordinate (columns) is acted on. If the ⊖B form is used, then the first coordinate is acted on:

```
        ⌽1 2 3 4
4 3 2 1
        ⌽'LIVE'
EVIL
        ⌽2 3⍴⍳6
 3 2 1
 6 5 4
        A←2 2 4⍴ 'SAVEMUCHMORETIME'
        A
SAVE
MUCH

MORE
TIME
        ⌽[1]A
MORE                   The first coordinate (plane) is specified;
TIME                   the planes are reversed.

SAVE
MUCH
        ⌽[2]A
MUCH                   The second coordinate (rows) is specified;
SAVE                   the rows in each plane are reversed.

TIME
MORE
        ⌽[3]A
EVAS                   The third coordinate (columns) is
HCUM                   specified; the columns in each plane are
                       reversed.
EROM
EMIT
        ⌽A
EVAS                   The last coordinate is acted on.
HCUM

EROM
EMIT
        ⊖A
MORE                   The first coordinate is acted on.
TIME

SAVE
MUCH
```

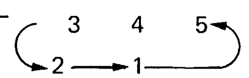
Dyadic (Two-Argument) Form: Rotate A⌽[I]B or A⌽B or A⊖B

The *rotate* function rotates the elements of argument B the number of positions specified by argument A. If argument A is positive, then the elements of argument B are rotated to the left (rows), or upward (columns). If A is negative, the elements are rotated to the right (rows), or downward (columns). Argument B can be any expression. The shape of the result is the same as that of argument B.

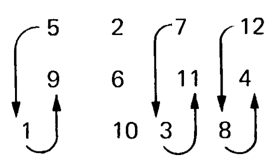
When argument B is a multidimensional array, the index entry [I] can be used to specify the coordinate that is acted on. If the index entry is omitted, the last coordinate (column) is acted on. If the A⊖B form is used, then the first coordinate is acted on.

If argument B is a vector, then argument A must be a scalar or single-element array. If argument B is a matrix, then argument A must be a scalar or vector. When argument A is a vector, the number of elements in argument A must be the same as the number of elements in the coordinate being rotated. For example, if B is a 3 by 4 matrix (each row has four elements) and the row coordinate is specified, A must have four elements:

```
      2⌽1 2 3 4 5
3 4 5 1 2
      ¯2⌽1 2 3 4 5
4 5 1 2 3
      7⌽1 2 3 4 5
3 4 5 1 2
      B←3 4⍴1 2 3 4 5 6 7 8 9 10 11 12
      B
 1  2  3  4
 5  6  7  8
 9 10 11 12
      1 0 1 2⌽[1]B
 5  2  7 12
 9  6 11  4
 1 10  3  8
      0 1 2⌽[2]B
 1  2  3  4
 6  7  8  5
11 12  9 10
      0 1 2⌽B
 1  2  3  4
 6  7  8  5
11 12  9 10
      A←-1 0 1 2
      A
¯1 0 ¯1 ¯2
      A⌽[1]B
 9  2 11  8
 1  6  3 12
 5 10  7  4
```



The first coordinate (rows) is specified; therefore, the rotation is between rows.



The second coordinate (columns) is specified; therefore, the rotation is between columns.

The last coordinate is acted on.

If argument B is an N-rank array, argument A must be a scalar or an array with a rank that is one less than the rank of argument B. The shape of argument A must be the same as argument B less the coordinate being acted on:

```
        B←3 3 3⍴⍳27
        B
 1  2  3
 4  5  6
 7  8  9

10 11 12
13 14 15
16 17 18

19 20 21
22 23 24
25 26 27
        ⍴B
3 3 3
        A←3 3⍴1 0 0 0 2 0 0 0 0
        A
1 0 0
0 2 0
0 0 0
        ⍴A
3 3
```

The shape of argument A must be the same as argument B less the coordinate being acted on.

```
        A⌽[1]B
10  2  3
 4 23  6
 7  8  9

19 11 12
13  5 15
16 17 18

 1 20 21
22 14 24
25 26 27
```

The first coordinate (planes) is specified; therefore, the rotation is between planes.

The first element in each plane is rotated one position between planes.

Argument A

1	0	0
0	2	0
0	0	0

The middle element in each plane is rotated two positions between planes.

```
        A⌽[2]B
 4  2  3
 7  5  6
 1  8  9

10 17 12
13 11 15
16 14 18

19 20 21
22 23 24
25 26 27
```

The second coordinate (rows) is specified; therefore, the rotation is between the rows.

Argument A

1	0	0	Rotation between rows of the first plane
0	2	0	Rotation between rows of the second plane
0	0	0	Rotation between rows of the third plane

The ⍉ Function: Transpose, Generalized Transpose

You form the ⍉ symbol by overstriking the ○ symbol and the \ symbol.

Monadic (One-Argument) Form: Transpose ⍉B

The *transpose* function reverses the coordinates of argument B. Argument B can be any expression. If argument B is a scalar or vector, the argument is unchanged by the function:

```
        ⍉7
7
        ⍉'ABCD'
ABCD
        B←2 3⍴1 2 3 4 5 6
        B
 1 2 3  ←─────────── 2-row 3-column matrix.
 4 5 6
        ⍉B
 1 4  ←───────────── 3-row 2-column matrix.
 2 5
 3 6
        B←2 3 4⍴⍳24
        B
  1  2  3  4
  5  6  7  8
  9 10 11 12

 13 14 15 16
 17 18 19 20
 21 22 23 24
        ⍉B
  1 13
  5 17
  9 21

  2 14
  6 18  ←─────────── The coordinates are reversed.
 10 22

  3 15
  7 19
 11 23

  4 16
  8 20
 12 24
```

Dyadic (Two-Argument) Form: Generalized Transpose A⍉B

The *generalized transpose* function interchanges the coordinates of argument B as specified by argument A. Argument B can be any expression. Argument A must be a vector or a scalar, and must have an element for each coordinate of argument B; also, argument A must contain all the integers between 0 or 1 (depending on the index origin) and the largest integer specified. For example, to transpose the rows and columns of a matrix, argument A would be 2 1:

```
      B←2 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      2 1⍉B
1 4
2 5
3 6
```

To transpose the rows and columns of a three-rank (three-coordinate) array, argument A would be 1 3 2:

```
      B←2 3 4⍴⍳24
      B
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
      1 3 2⍉B
 1  5  9
 2  6 10
 3  7 11
 4  8 12

13 17 21
14 18 22
15 19 23
16 20 24
```

An array with two planes, three rows, and four columns.

The second and third coordinates have been interchanged, forming an array with two planes, four rows, and three columns.

The ? Function: Deal

Monadic (One-Argument) Form

See the *roll* function earlier in this chapter under *Primitive Scalar Functions*.

Dyadic (Two-Argument) Form: Deal A?B

The *deal* function randomly selects numbers from 0 through B-1 or 1 through B (depending on the index origin), without selecting the same number twice. Both arguments must be single positive integers. Argument A must be less than or equal to argument B; argument A determines how many numbers are selected:

```
      1?7
4
      2?7
7 3
      7?7
5 2 1 7 3 6 4
      7?7
5 3 2 1 7 6 4
      7?7
2 7 6 4 5 1 3
```

The ⊥ Function: Decode (Base Value)

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Decode A⊥B

The *decode* function result is the value of argument B expressed in the number system specified by argument A. For example, to convert the vector 1 7 7 6 to its value in the decimal number system (base 10):

```
      10 10 10 10⊥1 7 7 6
1776
```

The following illustration shows how it was done:

Argument A (number system) specifies the following:

10 10 10 10

Ten units in each of these positions equal one unit of the next position to the left.

Argument B is a vector with these values:

1 7 7 6

The result is the same as doing the following:

```
           6 =    6
        7 x10 =   70
     7 x10x10 =  700
  1 x10x10x10 = 1000
                ----
                1776
```

The units position always represents itself.

The value in the next position is multiplied by the rightmost value in argument A.

The value in the next position is multiplied by the two rightmost values in argument A, and so on.

The arguments must be numeric. If one argument is a scalar or single-element array, the other argument can be a scalar, vector, or other array. The result will have the rank of the larger argument minus one.

If either argument A or B is not a scalar, they both must have the same length, or an error results.

Note: The value of the leftmost position of argument A can be zero, because even though there must be a value in that position, it is not used when the result is calculated. For example:

```
      0 10 10 10⊥1 7 7 6
1776
```

If either argument is a scalar, the value of that argument is repeated to match the length of the other:

```
      2 ⊥ 1 0 1
5
      2 2 2 ⊥ 1
7
```

```
        1 = 1
    0 × 2 = 0
1 × 2 × 2 = 4
            5
```

```
        1 = 1
    1 × 2 = 2
1 × 2 × 2 = 4
            7
```

If argument A is a vector and argument B is a matrix, argument A must have an element for each row of B:

```
      B←2 3⍴1 5 2 7 9 4
      B
1 5 2
7 9 4
      8 16 ⊥ B
23 89 36
```

If argument A is a matrix and argument B is a vector, each row of argument A is a separate conversion factor; argument B must be the same length as a row of argument A. The result will be a vector with one element for each row of argument A:

```
      A←2 3⍴10 10 10 1 3 12
      A
10 10 10
 1  3 12
      A⊥1 2 3
123 63
      A⊥2 3 4
234 112
```

This number system could represent yards, feet, and inches.

```
         3 =  3
    2 × 12 = 24
1 × 3 × 12 = 36
             63
```

If both arguments are matrices, each row of A (conversion factor) is applied to each column of B. The result is a matrix containing the converted values for each column of B:

```
      A←2 3⍴8 8 8 16 16 16
      B←3 2⍴1 2 2 12 3 3
      A
 8  8  8
16 16 16
      B
 1  2
 2 12
 3  3
      A⊥B
 83  227
291  707
```

```
          3 =  3
      2 × 8 = 16
  1 × 8 × 8 = 64
              83

            3 =   3
      12 × 16 = 192
 2 × 16 × 16 = 512
               707
```

The following examples convert hours, minutes, and seconds to all seconds:

```
      24 60 60⊥1 30 15
5415
```

The following illustration shows how it was done:

Argument A (number system)

24 60 60

60 units (seconds) equal one unit of the next position to the left.

60 units (minutes) equal one unit of the next position to the left.

24 units (hours) equal one unit of the next position.

Argument B

1 (hour) 30 (minutes) 15 (seconds)

The result was obtained as follows:

15	=	15 seconds
30 x 60	=	1800 seconds
1 x 60 x 60	=	3600 seconds
		5415 seconds

The ⊤ Function: Encode (Representation) [T N]

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Encode $A{\top}B$

This function is the reverse of the decode function. The *encode* function result is the representation of argument B in the number system specified by argument A.

Note: Be sure argument A is long enough to completely represent argument B or an incorrect answer results.

For example, the representation of 1776 in the decimal number system (base 10):

```
      10 10 10 10⊤1776
1 7 7 6
```

The following illustration shows how it was done:

Argument A (number system) specifies the following:

```
10 10 10 10
```

Ten units in each of these positions equal one unit of the next position to the left.

Argument B has this value:

1776

The result is the same as doing the following:

```
                          1
10x10x10 = 1000      |1776
                      1000
                       776

                         7
10x10 =     100      |776
                      700
                       76

                         7
                10     |76
                        70
                         6
```

Note: The value of the leftmost position of argument A can be zero. For example:

```
      0 10 10 10⊤1776
1 7 7 6
```

If both arguments are vectors, the result is a matrix. Each column in the result contains the representation for each element of argument B expressed in the number system specified by argument A:

```
      2 2 ⊤ 3 2 1
1 1 0
1 0 1
      16 16 ⊤ 17 36 24
1 2 1
1 4 8

                  1
             16 |17
                 16
                  1
```

If argument A is a matrix and argument B is a scalar, then the result is a matrix. Each column of the result contains the values of argument B expressed in the number system specified by the corresponding column of argument A:

```
      A←3 2⍴8 16
      A
8 16
8 16
8 16
      B←123
      A⊤B
1  0
7  7
3 11
      ⍉A⊤B
1 7  3
0 7 11
```

The result can be transposed so that each row represents the values of argument B expressed in the number systems specified by argument A.

8 × 8 = 64 ⟌123 → 1
64
59

8 ⟌59 → 7
56
3

If argument A is a scalar or vector and argument B is a matrix, the result is a matrix or N-rank array, with one plane for each element of argument A:

```
      A←1 3 12
      B←2 2⍴35 24 6 12
      A
1 3 12
      B
35 24
 6 12
      A⊤B
 0 0
 0 0

 2 2
 0 1

11 0
 6 0
```

This number system could represent yards, feet, and inches.

3 × 12 = 36 ⟌35 → 0

12 ⟌35 → 2
24
11

2 Feet 11 Inches

If both arguments are matrices, the result is an N-rank array, with one plane for each element of argument A. Each column of argument A represents a number system:

```
      A←2 2⍴10 20
      A
10 20
10 20
      B←2 2⍴ 99 88 77 66
      B
99 88
77 66
      A⊤B
 9  8
 7  6

 4  4
 3  3

 9  8
 7  6

19  8
17  6
```

Result of the number system in column 1 of argument A

Result of the number system in column 2 of argument A

The following example converts seconds to seconds, minutes, and hours:

```
      24 60 60 ⊤ 5415
1 30 15
```

The following illustration shows how it was done:

Argument A (number system)

```
24  60  60
 ↑   ↑   ↑
 │   │   └──────── 60 units (seconds) equal one unit of the next
 │   │             position to the left.
 │   │
 │   └──────────── 60 units (minutes) equal one unit of the next
 │                 position to the left.
 │
 └──────────────── 24 units (hours) equal one unit of the next
                   position to the left.
```

Argument B

5415 (seconds)

The result was obtained as follows:

```
                     1
60 x 60 = 3600  ┌5415
                 3600
                 ────
                 1815
                    30
            60  ┌1815
                 1800
                 ────
                   15
```

The ∊ Function: Membership

Monadic (One-Argument) Form

There is no monadic form.

Dyadic (Two-Argument) Form: Membership A∊B

The *membership* function result is a 1 for each element of argument A that can be found among the elements of argument B and a 0 for every element that cannot be found. The shape of the result is the same as the shape of argument A. Arguments A and B can be any scalar, vector, or array:

```
          4∊1 2 3 4 5
1
          1.5∊1 2 3 4 5
0
          'A'∊'BANNA'
1
          'A'∊'BANANA'
1
          'ABC'∊'BANANA'
1 1 0
          A←2 2⍴1 3 5 7
          B←4 4⍴12 45 78
          A
 1 3
 5 7
          B
 12 45 78 12
 45 78 12 45
 78 12 45 78
 12 45 78 12
          A∊B
 0 0
 0 0
          B←4 4⍴1 2 4 5 7 8
          A
 1 3
 5 7
          B
 1 2 4 5
 7 8 1 2
 4 5 7 8
 1 2 4 5
          A∊B
 1 0
 1 1
```

The ⌹ Function: Matrix Inverse, Matrix Divide

You form the ⌹ symbol by overstriking the ⎕ and the ÷ symbols.

Monadic (One-Argument) Form: Matrix Inverse ⌹B

The *matrix inverse* function inverts a nonsingular matrix or computes the pseudoinverse of a rectangular matrix. The result is a matrix. Argument B must be a numeric matrix, and the number of columns must not exceed the number of rows. The number of columns in the argument is the number of rows in the result, and vice versa.

If argument B is a nonsingular matrix, ⌹B is the inverse of B. If the matrix does not have an inverse, then DOMAIN ERROR results:

```
      A←2 2⍴1 3 5 7
      A
1 3
5 7
      ⌹A
 ¯0.875        0.375
  0.625       ¯0.125
      A←2 2⍴1 2 3 6
      A
1 2
3 6
      ⌹A
DOMAIN ERROR
      ⌹A
      ^
```

If argument B is a rectangular matrix, ⌹B is the pseudoinverse of the matrix (least squares solution):

```
      A←3 2⍴3 5 1 2 2 4
      A
3 5
1 2
2 4
      ⌹A
   2           ¯1           ¯2
  ¯1            0.6          1.2
```

Dyadic (Two-Argument) Form: Matrix Divide A⌹B

The *matrix divide* function solves one or more sets of linear equations with coefficient matrices. Argument B must be a numeric matrix. The number of columns in B must not exceed the number of rows. Argument A must be a numeric vector or a matrix. The length of the first coordinate of argument A must equal the length of the first coordinate of argument B.

The rank of the result is the same as the rank of argument B. The length of the first coordinate of the result is the same as the number of columns in argument B. If argument A is a matrix, then the second coordinate of the result is the same length as the second coordinate of argument A.

If argument B is a square matrix and argument A is a vector, then the result is the solution to the set of linear equations with coefficient matrix B and right-hand sides A:

```
      A←8 3
      B←2 2⍴1 2 3 ¯1
      A⌹B
2 3
      B←2 2⍴3 5 1 2
      B
 3 5
 1 2
      26 9⌹B
7 1
```

If argument B is a square matrix and argument A is a matrix, then the columns of the result are the solution to the sets of linear equations with coefficient matrix B and right-hand sides equal to the columns of A:

```
      A←2 2⍴26 16 9 6
      B←2 2⍴3 5 1 2
      A
26 16
 9  6
      B
3 5
1 2
      A⌹B
     7              2
     1              2
```

If argument B is rectangular, then the result is the least squares solution to one or more sets of linear equations:

```
      A←3 3⍴11 14 ¯2 4 7 ¯1
      B←3 2⍴3 5 1 2 2 4
      A
 11  14  ¯2
  4   7  ¯1
 11  14  ¯2
      B
3 5
1 2
2 4
      A⌹B
   ¯4              ¯7             1
    4.6             7            ¯1
```

The ⍎ Function: Execute [⊥ B] [∘ J]

You form the ⍎ symbol by overstriking the ⊥ and the ∘ symbols.

Monadic (One-Argument) Form: Execute ⍎ B

The *execute* function evaluates and executes argument B as an APL expression. Argument B can be any character scalar or vector:

```
        A←'1+2'
        A
1+2
        ⍎A ←────── The character vector in the variable A is executed.
3
```

```
        C←'((A*2)+(B*2))*.5'
        A←3
        B←4
        ⍎C
5
        A←6
        B←8
        ⍎C
10
```

```
        A←1
        B←2
        ⍎(A=B)/'A+B'
        A←2              A + B is executed only when A equals B.
        ⍎(A=B)/'A+B'
4
```

Dyadic (Two-Argument) Form

There is no dyadic form.

The ⍕ Function: Format [T/N] [∘/J]

You form the ⍕ symbol by overstriking the ⊤ and the ∘ symbols.

Monadic (One-Argument) Form: Format ⍕B

The monadic *format* function result is a character array that is identical in appearance to the one displayed when the value of argument B is requested:

```
      B←3 4⍴⍳12
      B
1  2  3  4
5  6  7  8
9 10 11 12
      X←⍕B
      X
1  2  3  4
5  6  7  8   ←—— This matrix is a character matrix.
9 10 11 12
```

Dyadic (Two-Argument) Form: Format A⍕B

The dyadic *format* function result, like the monadic format function result, is a character array. However, argument A is used to control the format (the spacing and precision) of the result. Argument A is a pair of numbers: the first number determines the total width of the format for each element, and the second number determines the precision used.

If the precision number is positive, the result is in the decimal form, with the number of decimal places specified by the precision number. If the precision number is negative, the result is in scaled form, with the number of digits to the left of the E specified by the precision number:

```
      B←3 2⍴12.34 ¯34.567 0 12 ¯0.26 ¯123.45
      B
 12.34         ¯34.567
  0             12
 ¯0.26        ¯123.45
      X←9 2⍕B
      X
    12.34    ¯34.57
      .00     12.00
    ¯.26    ¯123.45
```

Width of Nine Positions

(left margin)

```
      C←9 ¯2⍕B
      C
  1.2E01   ¯3.5E01
  0.0E¯01   1.2E01
 ¯2.6E¯01  ¯1.2E02
```

Width of Nine Positions

(left margin)

If the width entry in argument A is zero, the 5110 uses a field width such that at least one space will be left between adjacent numbers. If only a single number is used, a width entry of zero is assumed:

```
      E←2⍕B
      E
  12.34   ¯34.57
    .00    12.00
   ¯.26  ¯123.45
```

Width of Eight Positions

(left margin)

If you get a DOMAIN ERROR when using the format function, increase the width (left digit) of the left argument.

You can format each column of an array differently by using a control pair in argument A for each column of the array:

```
      F←6 2 6 1⍕B
      F
 12.34 ¯34.6
   .00  12.0
  ¯.26¯123.4
```

Width of Six Positions

(left margin)

Notes:

1. If the ⎕PW system variable is set to an invalid value, ⎕PW IMPLICIT ERROR will result when the format function is used.
2. Even if the specified format causes all of the significant digits to be truncated, the sign of the original number is retained. For example:

```
      4 2⍕¯.0004
¯.00
```

The sign is retained.

3. The format function should not be used to round a number up or down. Instead, the floor function should be used for rounding. For example, ⌊A+.5 rounds the value of A to the nearest integer.

APL OPERATORS

An APL operator applies one or more dyadic primitive scalar functions to arrays. The operators are *reduction, inner product, outer product,* and *scan.*

Reduction Operator (/)

The symbol for the *reduction* operator is /. The forms of reduction are f/[I]B or f/B or f⌿B, where f can be any primitive dyadic scalar function that is applied between the elements of a single vector.

The rank of the result is one less than the rank of argument B, unless argument B is a scalar or a single-element vector; then the result is the value of the single element of argument B. When argument B is a vector, the reduction of that vector is the same as putting the primitive dyadic function between the elements:

```
      B←1 2 3 4
      +/B
10
      1+2+3+4
10
```

If argument B is an empty vector (see Chapter 3), then the result is the identity element, if one exists, for the specified function. For example:

```
      ×/⍳0
1 ←——————— The identity element of × (times) is 1.
```

When argument B is a multidimensional array, the [I] index entry is used to specify the coordinate acted on. If the index entry is omitted, the last coordinate (columns) is acted on. If the f⌿B form is used, the first coordinate is acted on. Indexing along a nonexistent coordinate will result in INDEX ERROR.

When argument B is a multidimensional array, the coordinate of argument B that is acted on is eliminated:

```
      B←2 3⍴1 2 3 4 5 6
      B
 1 2 3
 4 5 6
      +/B
6 15
      +/[2]B
6 15
      +⌿B
5 7 9
      +/[1]B
5 7 9
      B←2 3 4⍴⍳24
      B
  1  2  3  4
  5  6  7  8
  9 10 11 12

 13 14 15 16
 17 18 19 20
 21 22 23 24
      +/[1]B
 14 16 18 20
 22 24 26 28
 30 32 34 36
      B←⍳0
      B

      +/B
0
      =/B
1
      ⌈/B
¯7.237E75
```

The last coordinate (columns) is assumed; therefore, the reduction is between columns:

1+2+3=6

The second coordinate (in this case, columns) is specified.

The first coordinate (rows) is specified; therefore, the reduction is between rows:

1	2	3
<u>4</u>	<u>5</u>	<u>6</u>
5	7	9

The first coordinate (planes) is specified; therefore, the reduction is between planes.

If argument B is an array that has a coordinate whose dimension is zero, then reduction along that coordinate yields an array whose elements are equal to the right identity element for the function, if the right identity element exists. The identity element for each function is given in the preceding table.

Inner Product Operator (.)

The symbol for the *inner product* operator is **.** (period). The inner product operator is used to combine any two primitive scalar dyadic functions and cause them to operate on an array. An example of its use would be in matrix algebra, in finding the matrix product of two matrices. The form for inner product is: A ⓕ . ⓖ B, where ⓕ and ⓖ are any primitive scalar dyadic functions. Function ⓖ is performed first and then ⓕ reduction is applied between the results of function ⓖ.

The result is an array; the shape of the array is all but the last coordinate of argument A catenated to all but the first coordinate of argument B (¯1↓ρA),(1↓ρB). If argument A and argument B are matrices, the elements in each row of argument A are acted on by the elements in each column of argument B:

```
      A←2 2ρ1 2 3 4
      B←2 2ρ5 6 7 8
      A
1 2
3 4
      B
5 6
7 8 ——————————————(1×5) + (2×7) = 19
      A+.×B
19 22
43 50
```

The above example is the same as doing the following for each element in the result:

```
      (1×5)+(2×7)
19
      (1×6)+(2×8)
22
      (3×5)+(4×7)
43
      (3×6)+(4×8)
50
```

The shapes of arguments A and B must conform to one of the following conditions:

1. Either A or B is a scalar.

```
         A←2 2⍴1 2 3 4
         B←5
         A
1 2
3 4
         B
5                          (1 × 5) + (2 × 5) = 15
         A+.×B
15 35
```

2. The last coordinate of argument A is the same length as the first coordinate of argument B. (If both are matrices, the column coordinate of argument A is the same length as the row coordinate of argument B.)

```
       A←3 2⍴1 2 3 4 5 6
       B←2 3⍴7 8 9 10 11 12
       A
1 2
3 4
5 6
       B
 7  8  9
10 11 12
       A+.×B               (3 × 8) + (4 × 11) = 68
 27  30  33
 61  68  75
 95 106 117
```

```
         A←'DAN'
         B←4 3⍴ 'DONDANMELTOM'
         A
DAN
         B              Transpose B so that the first coordinate of
DON                     B has the same length as the last coordinate
DAN                     of A.
MEL
TOM
         A∧.=⍉B
0 1 0 0                 The second row of B is equal to A.
```

Notice that the inner product operator can be used to find equal names in a matrix (table). However, if you use the inner product operator with the ∨ and ≠ functions, unpredictable results can occur. Instead, you should use a statement like ~A∧.=B. For example:

```
      ~A∧.=⍉B
1 0 1 1
```

If argument A and argument B are N-rank arrays, the elements in each row of argument A are acted on by the elements in each plane of argument B:

```
      A←2 2 2⍴⍳8
      A
1 2
3 4

5 6
7 8
      B←2 2 2⍴8+⍳8
      B
 9 10
11 12

13 14
15 16
      A+.×B
 35  38
 41  44

 79  86
 93 100


123 134
145 156

167 182
197 212
```

Outer Product Operator (∘ .)

The symbols for the *outer product* operator are ∘ . . The outer product operator causes a specified primitive scalar dyadic function to be applied between argument A and argument B so that every element of argument A is evaluated against every element of argument B. The form of the function is: A∘ . ⓕB, where ⓕ is a dyadic primitive scalar function. Arguments A and B can be any expressions. Unless argument A is a scalar, the shape of the result is the shape of argument A catenated to the shape of argument B. If argument A is a scalar, the shape of the result is the same as the shape of argument B:

```
      A←2
      B←1 2 3 4
      A
2
      B
1 2 3 4
      A∘.×B
2 4 6 8
      A←1 2 3
      B←3 4 5
      A
1 2 3
      B
3 4 5
      A∘.×B
 3  4  5
 6  8 10
 9 12 15
```

The above example is the same as doing the following for each element in the result:

```
      1×3
3
      1×4
4
      1×5
5
      2×3
6
      2×4
8
      2×5
10
      3×3
9
      3×4
12
      3×5
15
```

More examples:

```
           A←2
           B←2 3⍴1 2 3 4 5 6
           A∘.+B
     3 4 5
     6 7 8
           A←2 2⍴1 2 3 4
           B←3 3⍴⍳9
           A
     1 2
     3 4
           B
     1 2 3
     4 5 6
     7 8 9
           Z←A∘.+B
           ⍴Z
    2 2 3 3
           Z               1+1=2
      2  3  4
      5  6  7
      8  9 10
                           2+1=3
      3  4  5
      6  7  8
      9 10 11
                           3+1=4

      4  5  6
      7  8  9
     10 11 12
                           4+1=5
      5  6  7
      8  9 10
     11 12 13
```

Scan Operator (\)

The symbol for the *scan* operator is \. The forms of scan are: ⓕ\[I]B, ⓕ\B or ⓕ⍀B, where ⓕ can be any scalar dyadic primitive function and argument B is a numeric vector or other array. The scan operator, like the reduction operator, operates on the elements of a single vector, and is the same as putting the primitive dyadic function between the elements. But the scan operator accumulates the results as the operation is repeated along the vector. The shape of the result is the same as that of the input argument:

```
      +\1 2 3 4 5
1 3 6 10 15
      1
1
      1+2
3
      1+2+3
6
      1+2+3+4
10
      1+2+3+4+5
15
```

This result is the same as doing the following for each element in the result. The first element in the result is the first element of the argument.

When argument B is a multidimensional array, the [I] index entry is used to specify the coordinate the scan is to proceed along. If the index entry is omitted, the last coordinate (columns) is acted on. If the ⓕ⍀B form is used, the first coordinate is acted on.

```
      B←3 4⍴⍳12
      B
 1  2  3  4
 5  6  7  8
 9 10 11 12
      +\[1]B
 1  2  3  4
 6  8 10 12
15 18 21 24
      +\[2]B
 1  3  6 10
 5 11 18 26
 9 19 30 42
      +\B
 1  3  6 10
 5 11 18 26
 9 19 30 42
      +⍀B
 1  2  3  4
 6  8 10 12
15 18 21 24
```

The first coordinate (rows) is specified; therefore, the scan is between rows.

The second coordinate (columns) is specified; therefore, the scan is between columns.

```
      A←2 3 4⍴⍳24
      A
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
      +\[1]A
```

The first coordinate (planes) is specified; therefore, the scan operation is between planes.

```
 1  2  3  4
 5  6  7  8
 9 10 11 12

14 16 18 20
22 24 26 28
30 32 34 36
      +\[2]A
```

The second coordinate (rows) is specified; therefore, the scan operation is between rows for each plane.

```
 1  2  3  4
 6  8 10 12
15 18 21 24

13 14 15 16
30 32 34 36
51 54 57 60
      +\[3]A
```

The third coordinate (columns) is specified; therefore, the scan operation is between columns for each plane.

```
 1  3  6 10
 5 11 18 26
 9 19 30 42

13 27 42 58
17 35 54 74
21 43 66 90
```

SPECIAL SYMBOLS

Assignment Arrow ←

The assignment arrow causes APL to evaluate everything to the right of the arrow and associate that value with the name to the left of the arrow. For example, A←2+3 means that 2+3, or 5, is assigned to the name A. When A is used in a later APL statement, it has a value of 5.

Notes:

1. When a value assigned to a variable is used as the argument for a function, the value assigned to the variable is used by the function, regardless of any previous or future value assigned to the variable. For example:

```
      A←4
      (A←3)+A
6
      A
3
      (A←3)+A←4
7
      A
3
```

2. To avoid confusion, a variable should not be referenced in the same expression it is assigned, except directly to the right of the assignment. For example:

```
      A←2
      A←A+1
      A
3
```

Branch Arrow →

The branch arrow is used for the following:

- To change the order in which the statements are executed in a user-defined function. See *Branching* in Chapter 6 for more information on branching.
- To resume execution of a suspended function.
- To clear the state indicator.

Quad ⎕ [⎕ / L]

The quad is used to ask for input and to display output. To display output, the quad must appear immediately to the left of the assignment arrow. The value of the APL expression to the right of the arrow is assigned to the quad and will be displayed. For example:

```
      5+⎕←4+3
7
12
      A←⎕←'CHARACTER DATA'
CHARACTER DATA
```

The 7 displayed is the value assigned to the quad. The 12 is the final evaluation of the APL expression.

When used to ask for input, the quad can appear anywhere except to the immediate left of the assignment arrow. Execution of the expression stops at the quad and resumes when an expression is entered to replace the quad. When a quad is encountered, the quad and colon symbols (⎕:) are displayed to indicate that input is requested. For example:

```
      25×⎕
⎕:
      4
100
```

See Chapter 6 for more information on quad input or output within a user-defined function.

Quad Quote ⍞ [⎕ / L] [' / K]

You form the quad quote symbol by overstriking the quote symbol ' and the quad symbol ⎕. The quad quote operates the same way as the quad when requesting input, except that the data entered is treated as character data. For example:

```
      X←⍞
CAN'T
      X
CAN'T
      X←⎕
'CAN''T'
      X
'CAN''T'
```

Note: If a system command is entered for a quad quote input request, the system command is treated as a character string and will not be executed.

See Chapter 6 for more information on quad quote input or output within a user-defined function.

Comment ⍝ [∩ C] [∘ J]

The comment symbol is formed by overstriking the ∩ symbol and the ∘ symbol. The comment symbol must be the first nonblank character in a line and indicates that the line should not be executed. For example:

```
               ∇ONE PLUS TWO
[1]  ⍝THE PURPOSE OF THIS FUNCTION IS
[2]  ⍝TO ADD TWO NUMBERS TOGETHER!
[3]  ONE+TWO∇
       12 PLUS 34
46
```

Parentheses ( ) [([] [)]]

Parentheses are used to specify the order of execution. The order of execution is from right to left with the expressions in parentheses resolved (right to left) as they are encountered. For example:

```
       3+4×6
27
       (3+4)×6
42
```

Chapter 5. System Variables and System Functions

SYSTEM VARIABLES

System variables provide controls for the system and information about the system to the user. These variables can be used by a function as arguments the same as any variable.

The following is a list of the system variables and their meanings. A complete description of each follows the list:

Variable Name	Meaning
⎕CT	Comparison tolerance
⎕IO	Index origin
⎕PP	Printing precision
⎕PW	Printing width
⎕RL	Random link
⎕LC	Line counter
⎕WA	Workspace available
⎕LX	Latent expression
⎕AV	Atomic vector

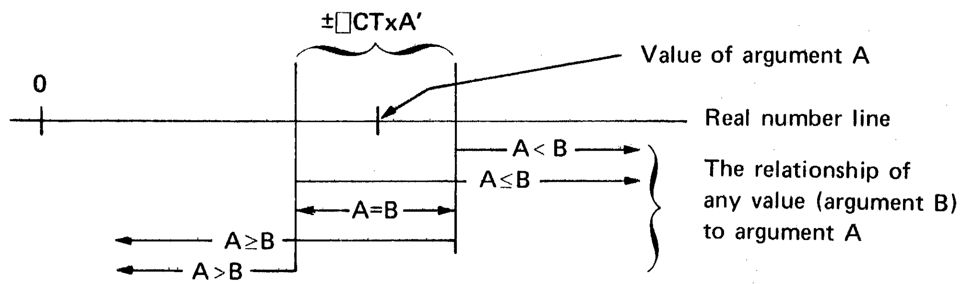
Notes:
1. To find the value assigned to a system variable, enter the variable name. The value assigned to the ⎕CT, ⎕IO, ⎕PP, ⎕PW, ⎕RL, and ⎕LX system variables can be changed by the assignment arrow (←). For example, entering ⎕IO←0 assigns the value 0 to the ⎕IO system variable.
2. The use of any system variable causes an entry to be made in the symbol table for that symbol. Therefore, if the symbol table is full, a SYMBOL TABLE FULL error is generated.
3. The following system variables are implemented as part of the system for the purpose of maintaining syntactic compatibility between the 5110 and APL/SV or VS/APL:

Variable Name	Meaning	Default Value
⎕AI	Accounting information	0 0 0 0
⎕DL	Delay	0
⎕TS	Time stamp	1900 0 0 0 0 0 0
⎕TT	Terminal type	0
⎕UL	User load	1

Although these system variables are not generally used with the 5110, the ⎕AI and ⎕TS system variables can be assigned a new value.

Comparison Tolerance: ⎕CT

The value of this variable determines the maximum tolerance (how different the two numbers must be to be considered unequal) when any relational function is used and at least one argument is a noninteger. For example, two numbers are considered unequal if the relative difference between the two numbers exceeds the comparison tolerance value. The following illustration shows how the comparison tolerance works with the relational functions:



A' is the next lower integer power of 16 for the largest argument. For example:

```
      16*0 1 2 3 4
1 16 256 4096 65536
      ⎕CT←.1
      14=15.
0
```

The First Five Integer Powers of 16.

Next lower integer power (A') is 1; therefore, the difference between the arguments exceeds ±⎕CT×A' (.1×1=.1)

```
      15=16.
1
```

Next lower integer power (A') is 16; therefore, the difference between the arguments does not exceed ±⎕CT×A' (.1×16=1.6).

```
      253=255.
0
```

Next lower integer power (A') is 16; therefore, the difference between the arguments exceeds ±⎕CT×A' (.1×16=1.6).

```
      255=257.
1
```

Next lower integer power (A') is 256; therefore, the difference between the arguments does not exceed ±⎕CT×A' (.1×256=25.6).

Note: The ⎕CT function considers any number in decimal form a noninteger. For example, 1000 is an integer and 1000. is a noninteger.

The value of the comparison tolerance variable also affects the floor and ceiling functions. The comparison tolerance is added to the argument for the floor function and subtracted for the ceiling function. For example:

```
      ⎕CT←.03
      ⌊2.98          2.98 + .03 = 3.01 (The integer 3 is in the range of
3                                    2.98 + .03.)
      ⌊2.96          2.96 + .03 = 2.99
2
      ⌈3.03          3.03 - .03 = 3  (The integer 3 is in the range of
3                                3.03 - .03.)
      ⌈3.04          3.04 - .03 = 3.01
4
```

In a clear workspace, the comparison tolerance value is set to 1E¯13 (see Chapter 3 for an explanation of scaled representation).

Index Origin: ⎕IO

The value of this variable determines the index origin. The value can be either 0 or 1, which means that the first component of a vector or array is indexed with a 0 or 1, depending on what the value is set to. In a clear workspace, the value is set to 1.

The functions affected by index origin are indexing ([ ]), index generator (⍳), index of (⍳), roll (?), deal (?), generalized transpose (⍉), grade up (⍋), and grade down (⍒).

```
      ⎕IO←0                    The index values represented by the
      1 2 3 4⍳1 2 3 4          result start from 0 rather than 1.
0 1 2 3
      ⍋6 3 4 5
1 2 3 0
      ⍳4
0 1 2 3
      ⎕IO←1
```

Note: All other examples in this manual are shown with the index origin set to 1.

Printing Precision: ⎕PP

The value of this variable determines the number of significant digits displayed for decimal numbers and for integers with more than 10 digits. The value of this variable does not affect the internal precison of the system. The value can be from 1 to 16. In a clear workspace, the value is set to 5. This means that the number of significant digits displayed for decimal numbers or for integers with more than 10 digits is limited to 5, and scaled representation (see Chapter 3) is used (if required). For example:

```
                            Decimal Number Examples
        12345.6
12346                       Five digits are displayed, and the least
        12345.67            significant digit is rounded off.
12346
        123456.7
1.2346E5
```

```
                            Integer Examples
        1234567890
1234567890
        12345678901
1.2346E10
```

Print Width: ⎕PW

The value of this variable determines the length of the output line for both the display and printer. The value can be from 30 to 390. In a clear workspace, the value is 64. If this variable is set to a value greater than the length of one line across the display, the output will overflow onto the next line.

If this variable is set to a value greater than the length of one line across the printer, an (O-U-T) symbol is printed in position 132 when the output exceeds the print width.

Notes:

1. During function definition mode (see Chapter 6), the print width variable is automatically set to 128. The variable returns to its original value when the function is closed.
2. When you open, display, and close a function using one statement (∇FUNCTION [⎕]∇), the system uses the current value of the variable.

Random Link: ⎕RL

The value of this variable is used in generating random numbers. The value can be from 1 to $2^{31}-2$. In a clear workspace, the value is 7*5 (16807). This value is changed by the system each time a random number is generated.

Note: Making ⎕RL local to a user-defined function produces unpredictable results.

Line Counter: ⎕LC

This variable is a vector. The first element is the function statement number currently being executed. The next element is the number of the statement (in another function) that invoked the function being executed. The remaining elements follow the same pattern. The user cannot set this variable but can display it. Attempts to modify ⎕LC are ignored by the system.

Workspace Available: ⎕WA

The value in this variable indicates the amount of unused space (the number of unused bytes) in the active workspace. The user cannot set the value for this variable but can display it. Attempts to modify ⎕WA are ignored by the system.

Latent Expression: ⎕LX

A character vector assigned to the latent expression variable is automatically executed as an expression by the execute (⍎) function when a stored workspace containing the latent expression is loaded into the active workspace.

Uses of the latent expression variable include the form ⎕LX←'G', where a function named G is executed when the stored workspace is made active. The form ⎕LX←'''MESSAGE WHEN WORKSPACE IS MADE ACTIVE''' displays the message MESSAGE WHEN WORKSPACE IS MADE ACTIVE when the stored workspace is loaded into the active workspace.

Atomic Vector: ⎕AV

The atomic vector is a 256-element vector that includes all possible APL characters. The following example shows how it can be used to determine the indices of any known characters in the vector (assuming ⎕IO is 1):

```
      ⎕AV⍳'ABC'
87 88 89
```

Appendix B contains a list of the characters in the atomic vector. The most common use of the atomic vector is for generating line feed and cursor return characters to arrange output. The following example shows how the atomic vector can be used to generate these characters.

The function called NAMES will display your first and last name. Each name will start at the left margin, and each character in the name will be one line lower than the previous character:

```
      ∇NAMES[⎕]∇
    ∇ A NAMES B;OUTPUT;I;J;⎕IO
[1]   ⎕IO←1
[2]   J←(⍴,B)+I←⍴,A
[3]  ⍝STATEMENT 5 CATENATES THE ARGUMENTS TOGETHER
[4]  ⍝AND ALSO PUTS A BLANK CHARACTER BETWEEN EACH CHARACTER
[5]   OUTPUT←((2×J)⍴1 0)\(,A),(,B)
[6]  ⍝STATEMENT 8 PLACES A LINE FEED CHARACTER (⎕AV[160])
[7]  ⍝IN EACH BLANK ELEMENT OF OUTPUT
[8]   OUTPUT[2×⍳J-1]←⎕AV[160]
[9]  ⍝STATEMENT 11 PLACES A CURSOR RETURN CHARACTER (⎕AV[157])
[10] ⍝AFTER THE FIRST NAME
[11]  OUTPUT[2×I]←⎕AV[157]
[12] ⍝NOW WHEN THE CHARACTER VECTOR OUTPUT IS DISPLAYED,
[13] ⍝APL RESPONDS WITH THE APPROPRIATE ACTION WHEN A LINE
[14] ⍝FEED CHARACTER (⎕AV[160]) OR CURSOR RETURN CHARACTER
[15] ⍝(⎕AV[157]) IS ENCOUNTERED IN THE CHARACTER STRING
[16]  OUTPUT
    ∇

      'VIRGINIA' NAMES 'WINTER'
V
 I
  R
   G
    I
     N
      I
       A
W
 I
  N
   T
    E
     R
```

SYSTEM FUNCTIONS

System functions are used like the primitive (built-in) functions; they are monadic (one argument) or dyadic (two arguments) and have explicit results.

Following is a list of the system functions and their meanings. A complete description of each follows the list:

System Function			Meaning
type	⎕CC	operation	Console control
	⎕CR	name	Canonical representation
	⎕FX	name	Fix
	⎕EX	name	Expunge
	⎕NL	class	Name list
character	⎕NL	class	Name list beginning with the specified character
	⎕NC	name	Name classification

The ⎕CC Function: Console Control

The ⎕CC function controls some of the interface capabilities between the user and the 5110.

The ⎕CC function can be monadic (takes one argument) or dyadic (takes two arguments). The ⎕CC function returns an explicit result of 1 when the function is successfully executed; otherwise, the result is 0.

In the monadic form, the ⎕CC function sets the system to a unique national character set (see Appendix C). The right argument must be one of the following characters enclosed in single quotes:

Argument	Character Set
. (period)	EBCDIC
0 (zero)	International
1 (one)	Austria/Germany
2 (two)	Belgium
3 (three)	Brazil
4 (four)	Denmark/Norway
5 (five)	Finland/Sweden
6 (six)	France
7 (seven)	Italy
8 (eight)	Japan
9 (nine)	Portugal
÷ (divide)	Spain
× (multiply)	Spanish speaking
- (minus)	United Kingdom
+ (plus)	French Canadian

For example, the following instruction sets the system to United Kingdom character set:

```
⎕CC
```

In the dyadic form, the ⎕CC function is used to:

- Turn the display screen off and on during the execution of a user-defined function
- Sound the audible alarm
- Set the keyboard input to standard APL characters or lowercase alphabetic characters
- Scroll the display screen up or down
- Set the left tab position for printed output

The left argument is a numeric value that specifies the type of operation. The right argument is a numeric scalar or vector that specifies how the operation is performed. If the right argument is a vector, the values in the vector determine the sequence in which the operations are performed, starting with the first element. The operations and the values for the left and right arguments are described in the following sections.

To Turn the Display Screen Off and On During Execution of a User-Defined Function

Left Argument	**Right Argument**	**Operation**
1	0	Turn the display screen off.
1	1	Turn the display screen on.

For example:

```
R← 1 ⎕CC 0
```

turns the display screen off. The display screen remains off until:

- The ⎕CC function is used to turn the display screen on again.
- The user-defined function completes execution.
- The user-defined function requests ⎕ or ⍞ input.
- A function statement that generates displayed output is executed. For this reason, the explicit result of the ⎕CC function must be assigned to a variable so that the display screen remains off.

Note: The internal processing capability of the 5110 is increased by 18% when the display screen is turned off.

To Sound the Audible Alarm

Left Argument	Right Argument	Operation
2	0	Turns the audible alarm off.
	1	Sound the audible alarm until: • The ⎕CC function is used to turn the audible alarm off • Any key on the keyboard (other than the CMD and shift key) is pressed
	2	Sound the audible alarm for approximately 1/4 second.

For example:

```
2 ⎕CC 2 2 2 2
```

sounds the audible alarm four times for approximately 1/4 second each time.

To Set the Keyboard Input to Standard APL Characters or Lowercase Alphabetic Characters

Left Argument	Right Argument	Operation
3	0	Set to standard APL characters.
	1	Set to lowercase alphabetic characters.

For example:

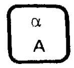
```
3 ⎕CC 1
```

sets the keyboard input to lowercase alphabetic characters until the keyboard input is set to standard APL characters.

Note: Only the alphabetic keys are affected by the ⎕CC function.

When keyboard input is set to lowercase alphabetic characters:

- Pressing the alphabetic keys enters lowercase alphabetic characters into the system. For example:

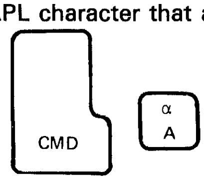


enters the character a.

- When the shift key is held down, pressing the alphabetic keys enters uppercase alphabetic characters into the system. For example:

enters the character A.

- When the CMD key is held down, pressing an alphabetic key enters the APL character that appears on the upper portion of the key. For example:



enters the character α.

The lowercase alphabetic characters are used primarily for ⍞ input.

To Scroll the Display Screen Up or Down

Left Argument	**Right Argument**	**Operation**
4	-16 to 16	Move the information on the display screen up or down for the specified number of lines.

A positive value in the right argument scrolls the information on the display screen up, and a negative value in the right argument scrolls the information on the display screen down.

For example:

```
4 ⎕CC ¯10 5
```

scrolls the information on the display screen down 10 lines and then up 5 lines.

Note: If a value greater than 16 or less than ¯16 is specified, the information on the display screen is not scrolled up or down, and the explicit result of the ⎕CC function is 0.

To Set the Left Tab Position for Printed Output

Left Argument	Right Argument	Operation
5	1 to 131	Set the tab print position where the first character is printed on the print line.

For example:

```
 5  ⎕CC  6
```

specifies that the next output to the printer starts in the sixth position on the print line.

When a tab position is specified, if the line to be printed extends beyond position 132, an ⍯ (O-U-T) symbol is printed in position 132. Except for the I/O error messages (ERROR eee ddd), the specified tab position also determines where the error messages are printed on the print line. See Appendix F for the error messages.

The ⎕CR Function: Canonical Representation

The ⎕CR function formats a user-defined function into a character matrix. This function is monadic (takes one argument); the argument for the ⎕CR function must be a scalar or vector of characters representing the name of an unlocked user-defined function. For example, you have the following user-defined function:

```
        ∇INTG[⎕]∇
      ∇ R←INTG A
[1]     R←Aρ0
[2]     I←1
[3]    START:R[I]←A
[4]     I←I+1
[5]     →(I≤A)/START
      ∇
```

The function INTG is used to create a vector whose length and contents are specified by the input argument:

```
      INTG 4
4 4 4 4
      INTG 7
7 7 7 7 7 7 7
```

To format the function INTG into a character matrix and assign the matrix to a variable named VAR, you would enter the following instruction:

```
VAR←⎕CR 'INTG'
```

VAR is displayed as follows:

```
      VAR
R←INTG A←————First row is line 0 of the function.
R←Aρ0
I←1
START:R[I]←A
I←I+1
→(I≤A)/START
      ρVAR←————Indicates VAR is a 6-row, 12-column matrix.
6 12
```

Notice that the line numbers are removed along with the opening and closing ∇. Also, labels within the function are aligned at the left margin.

Now you can change matrix VAR by simply indexing the elements:

```
      VAR[4;12]←'I'←—The element in row 4, column 12 is changed to I.
      VAR
R←INTG A
R←Aρ0
I←1
START:R[I]←I
I←I+1
→(I≤A)/START
```

To format a matrix created by the ⎕CR function into a user-defined function, use the ⎕FX function. The ⎕FX function is discussed next.

The ⎕FX Function: Fix

The ⎕FX function forms (fixes) a user-defined function from a character matrix (that was most likely formed by the ⎕CR function). This function is monadic (takes one argument); the argument for the ⎕FX function is the name of a matrix to be formed into a user-defined function. If an error is encountered (invalid character or missing single quote, for example) as the matrix is being formed into a user-defined function, the operation is interrupted, the number of the row in error minus one is displayed, and no change takes place in the active workspace (the user-defined function is not formed).

To show how the ⎕FX function works, we will use the matrix created in the previous example (see the ⎕CR function). To form matrix VAR into a user-defined function, enter the following instruction:

```
      ⎕FX VAR        APL responds with the name of the
INTG                 user-defined function.
```

The ⎕FX function produces an explicit result (the vector of characters that represents the name of the user-defined function), and the original definition of the user-defined function (if there was one) is replaced.

Now the function INTG can be displayed and executed:

```
        ∇INTG[⎕]∇
      ∇ R←INTG A
[1]     R←Aρ0
[2]     I←1
[3]    START:R[I]←I
[4]     I←I+1
[5]     →(I≤A)/START
      ∇

        INTG 5
1 2 3 4 5
        INTG 8
1 2 3 4 5 6 7 8
```

Following is an example that shows how the ⎕CR and ⎕FX functions can be used to modify the definition of a function within another function. This example will use the following user-defined function:

```
          ∇INTG[⎕]∇
        ∇ R←INTG A
[1]       R←Aρ0
[2]       I←1
[3]      START:R[I]←A
[4]       I←I+1
[5]       →(I≤A)/START
        ∇

          INTG 4
4  4  4  4
```

Format the function into a matrix:

```
          M←⎕CR 'INTG'←—— Canonical Representation
          M
R←INTG A
R←Aρ0
I←1
START:R[I]←A
I←I+1
→(I≤A)/START
```

Now, define a function called CHANGE, which, when performed, will execute a modified version of INTG.

```
        ∇ CHANGE;INTG;Y←INTG is made a local function so that the
[1]       M[4;12]←'I'    global version will not be changed (the
[2]       Y←⎕FX M        local version will not exist after the
[3]       INTG 4         execution of CHANGE is complete).
        ∇
                         Assign the explicit result of the ⎕FX
                         function to Y so that it will not be
                         displayed.

                         Execute the modified version of INTG.
```

```
          INTG 4 ←——————— Execute INTG.
4  4  4  4
          CHANGE ←——————— Execute CHANGE.
1  2  3  4
          INTG 4 ←——————— Execute INTG again.
4  4  4  4
```

The ⎕EX Function: Expunge

The ⎕EX function erases global objects or active local objects specified by the argument from the active workspace (unless the object is a pendent or suspended function). This function is monadic (takes one argument); the argument must be a scalar, vector, or matrix of characters.

Thus, if object AB is to be erased, the following instruction would be entered:

```
⎕EX 'AB'
```

Note: Even after the object is erased, the name remains in the symbol table (the part of the active workspace that contains all of the symbols used). To clear unused symbols from the symbol table, save and then reload the workspace.

The ⎕EX function returns an explicit result of 1 if the name is available and a 0 if it is not available or if the argument does not represent a valid name. When the ⎕EX function is applied to a matrix of names (each row represents a name), the result is a logical vector (zeros and/or ones) with an element for each name. The ⎕EX function is like the)ERASE command, except that it applies to the active referent (see Chapter 6, *Local and Global Names*) of a name.

Note: If the object being expunged is a shared variable (see Chapter 7), it will be retracted.

The ⎕NL Function: Name List

The ⎕NL function yields a character matrix; each row of the matrix represents the name of a local (active referent) or global object in the active workspace. The ordering of the rows has no special significance. The ⎕NL function can be either monadic (takes one argument) or dyadic (takes two arguments); in both the monadic and dyadic forms, the right argument is an integer, scalar, or vector that determines the class(es) of names that will be included in the result. The values for the input argument and associated classes of names are:

Argument	**Name Class**
1	Names of labels
2	Names of variables
3	Names of user-defined functions

It does not make any difference in what order the class of names appears in the argument. For example, ⎕NL 2 3 or ⎕NL 3 2 results in a matrix of all the variable and user-defined function names.

In the dyadic form, the left argument is a scalar or vector of alphabetic characters that restricts the names produced to those with the same initial character as that of the argument. For example, 'AD' ⎕NL 2 results in a matrix of all the variable names starting with the characters A and D.

Uses of the ⎕NL function include:

- Erasing objects of a certain class (and also beginning with a certain character). For example:

  ```
  ⎕EX 'B' ⎕NL 2
  ```

  erases all the variables whose names start with B.

- Avoiding the choice of a name that already exists.

The ⎕NC Function: Name Classification

The ⎕NC function is monadic (takes one argument); the argument is a scalar or array of characters. The result of the function is a vector of numbers representing the class of the name given in each row of the argument. The classes of names are as follows:

Result	**Meaning**
0	Name is available for use
1	Name of a label
2	Name of a variable
3	Name of a function
4	Name is nonstandard (not available for use)

For example:

```
      ⎕NC  AVERAGE
3
```

This result indicates that the name AVERAGE is the name of a user-defined function.

Chapter 6. User-Defined Functions

APL provides an extensive set of primitive functions; nevertheless, you may want a function to solve a special problem. APL provides a way to create a new function, called *function definition.* During function definition, you use existing APL functions to create new functions called *user-defined* functions.

Normally, the 5110 is in execution mode; that is, after a line has been entered and the EXECUTE key pressed, the 5110 executes that line. To define a function, the mode must be changed to function definition mode; after the function is defined, the mode must be changed back to execution mode before the function can be executed. You change the mode by entering the ∇ (del) symbol. The first ∇ changes the mode to function definition mode; the second ∇ indicates the end of function definition and changes the mode back to execution mode.

No statement error checking is performed during function definition mode. That is, all error checking is performed when the statement is executed.

MECHANICS OF FUNCTION DEFINITION

The following steps are required to define a new function:

1. Enter a ∇ followed by the function header (see *Function Header* in this chapter). After the function header is entered, APL responds with a [1] and waits for the first statement of the function to be entered:

```
           ∇HOME SCORE VISITOR (function header)
[1]
```

2. Enter the statements that define the operations to be performed by the function. As each line is entered, APL automatically responds with the next line number:

```
           ∇ HOME SCORE VISITOR
[1]          'THE FINAL SCORE IS:'
[2]          +/HOME
[3]          'TO'
[4]          +/VISITOR
```

   Notes:
   1. Since problems can occur when you are editing statements that exceed the print width (see ⎕PW system variable in Chapter 5), during function mode, the print width is automatically set to 128. This permits statements up to a maximum of 115 characters in length to be placed on the input line (lines 1 and 0) and edited. Editing statements are discussed later in this chapter. The print width returns to its original value when the function is closed.
   2. When you open the function definition, display the user-defined function, and close the function definition using one statement (see *Reopening Function Definition* in this chapter), the system uses the current value of ⎕PW. This allows user-defined functions with long statements to be printed on narrow paper.
   3. If a user-defined function contains a statement that is greater than 115 characters in length, that statement cannot be edited and the function cannot be written on the media. (See ⎕CR and ⎕FX in Chapter 5 for information on changing a user-defined function to a matrix.)

3. Enter another ∇ when the function definition is complete. The closing ∇ may be entered alone or at the end of a statement. For example:

```
[4]          +/VISITOR∇
   or
[5]          ∇
```

   Note: If the closing ∇ is entered at the end of a comment statement, which begins with a ⍝ symbol, the ∇ will be treated as part of the comment and the function will not be closed.

Function Header

The function header names the function and specifies whether a function has no arguments (niladic), one argument (monadic), or two arguments (dyadic).

Note: Function names should not begin with SΔ or TΔ because SΔ and TΔ are used for stop and trace control (*Stop Control* and *Trace Control* are discussed later in this chapter).

The function header also determines whether or not a function has an explicit result. If a function has an explicit result, the result of the function is temporarily stored in a result variable (names in the function header) for use in calculations outside the function. The result variable must be included in the result statement (the statement that determines the final result of the function) as well as the function header. For example:

```
      ∇ RESULT←X PLUS Y
[1]     RESULT←X+Y∇
        3 PLUS 4
7
      10+3 PLUS 4
17
```

Result Variable

The result of the function is temporarily stored in the result variable so that it can be used by another function.

User-defined functions that do not have an explicit result cannot be used as part of another expression. For example:

```
      ∇ X PLUS1 Y
[1]     X+Y∇
      10+3 PLUS1 4
7
VALUE ERROR
      10+3 PLUS1 4
        ^
```

The following table shows the possible forms of the function header:

Number of Arguments	Type	Format of Header: No Explicit Result	Format of Header: Explicit Result
0	Niladic	∇NAME	∇R←NAME
1	Monadic	∇NAME B	∇R←NAME B
2	Dyadic	∇A NAME B	∇R←A NAME B

There must be a blank between the function name and the arguments. Also, the same symbol cannot appear more than once in the function header; thus, Z←FUNCTION Z is invalid.

For user-defined functions, the order in which the arguments are entered is important. For example, assume that Z←X DIVIDE Y represents a function in which Z is the result of X÷Y. Now if 20 DIVIDE 10 is entered, the result is 2. However, if 10 DIVIDE 20 is entered, the result is 0.5.

Branching and Labels

Statements in a function definition are normally executed in the order indicated by the statement numbers, and execution terminates at the end of the last statement in the sequence. This normal order can be modified by *branching*.

Branching is specified by a right arrow (→) followed by a label (name) that specifies the statement that is to be branched to. For example, the expression →START means branch to a statement labeled START. When a label is assigned to a statement, the label must be followed by a colon (:) and must precede the statement. The colon separates the label from the statement:

```
        .
        .
        .
[2]   START:N←N+1
        .
        .
        .
[5]  →START
        .
        .
        .
```

In the previous example, the label START is assigned to the second statement in the function. In other words, START has a value of 2; however, if the function is edited and the statement is no longer the second statement in the function, START will automatically be given the value (or statement number) of the new statement. (See *Function Editing* later in this chapter.)

Labels are local to a function–which means they can be used only within that function. Following are some additional rules that apply to the use of labels:

- They must not appear in the function header.
- You cannot assign values to them.
- They can be up to 77 characters in length.
- They cannot be used on comments.
- When duplicate labels or labels that duplicate a local name are used, the first use of the label or name is the accepted use.

If the branch is to zero (→0) or any statement number not in the function, the function is exited when the branch statement is executed. If the value to the right of the → is a vector (for example, →L1,L2,L3), the branch is determined by the vector's first element. If the vector is an empty vector (there are no elements), the branch is not executed, and the normal sequence of statement execution continues. For example, the conditional branch →(I≥N)/START is evaluated as follows:

1. First, the condition (I≥N) is evaluated; the result is 1 if the condition is true and 0 if the condition is false.

2. The result of step 1 is then used as the left argument for the compress (A/B) function:
   a. If the result of step 1 was 1, START is selected from the right argument and a branch to the statement labeled START is taken.
   b. If the result of step 1 was 0, nothing is selected from the right argument (an empty vector is the result), and the sequence of execution falls through to the next statement.

Following are three examples of defining and using a function to determine the sum of the first N integers. Each function uses a different method of branching. Remember, the expression to the right of the → is evaluated, and the result determines to what statement the branch is taken:

```
      ∇ S←SUM1 N
[1]      S←0
[2]      I←1
[3]   CHECK:→LABEL×I≤N      Branch to LABEL if I≤N; otherwise,
[4]   LABEL:S←S+I           exit the function.
[5]     I←I+1
[6]     →CHECK∇

      SUM1 5
15

      ∇ S←SUM2 N
[1]      S←0
[2]      I←1
[3]   CHECK:→(I>N)/0        Branch to 0 (terminate the function)
[4]     S←S+I               or fall through.
[5]     I←I+1
[6]     →CHECK∇

      SUM2 5
15

      ∇ S←SUM3 N
[1]      S←0
[2]      I←0
[3]   CHECK:S←S+I
[4]     I←I+1
[5]     →(I≤N)/CHECK∇       Branch to CHECK or fall through.

      SUM3 5
15
```

Several forms of the branch instruction are shown in the following table:

Branch Instruction	**Result**
`→LABEL`	Branches to a statement labeled LABEL
`→0`	Exits function
`→LABEL×X=Y`	Branches to LABEL or exit function
`→((X<Y),(X=Y),(X>Y))/L1,L2,L3`	Branches to L1, L2, or L3
`→(L1,L2)[1+X=Y]`	Branches to L1 or L2
`→(X=Y)/0`	Exits function or falls through to next statement
`→(X=Y)/LABEL`	Branches to LABEL or falls through
`→(X=Y)ρLABEL`	

Note: Branching will also work if a specific statement number is specified to the right of the →. For example, →3 means branch to statement 3; or →X←3×A means X is assigned the value of 3 times the value of A, and the value of X is then used as the branch to statement number. However, these forms of branching (using statement numbers instead of labels) can cause problems if the function is edited and the statements are renumbered.

Local and Global Names

A local name is the name of a variable or user-defined function that is used only within a particular user-defined function. A global name is the name of a variable or user-defined function that can be used within a user-defined function and can also be used outside of it. An example of the use of a local variable name would be the name of a counter used in a user-defined function (which is not required for any use outside the function).

To make a name local to a user-defined function, it must be contained in the function header. For example, the function header ∇Z←EXAMPLE X;J;I establishes the result variable Z, the argument X, and variable J and I as local variables. Notice that the local names, other than the result variable and arguments, follow the right argument (if any) and are preceded by semicolons.

A local name can be the same as a global name (variable or user-defined function) or a local name in another function. However, any reference to the name local to the function will not change the values of any other global or local objects (variables or user-defined functions) or cause them to be used.

After a user-defined function has executed, the following rules apply to the local and global variables used by the function:

- Any value assigned to a local variable is lost.
- If a local variable had the same name as a global variable, the value of the global variable remains unchanged.
- If the value of a global variable was changed by the function, it retains the new value.

For example:

```
        LOC←100
        GLOB←100
        ∇RESULT←EXAMPLE;LOC;X
  [1]  LOC←50
  [2]  X←25
  [3]  GLOB←10
  [4]  RESULT←LOC+GLOB+X∇
        EXAMPLE
  85
        X←──────────X has no value after the function has executed.
  VALUE ERROR
        X
        ^           The global value associated with this name was
        LOC ←       unchanged by the function.
  100
        GLOB ←──────The global value was changed by the function
  10                because the name GLOB was not made local to
                    the function.
```

Because the value of a local name disappears as soon as execution of the function finishes, the only time you can use or display the value of a local name is while the function to which it belongs is still executing, is suspended, or is pendent.

Note: If a name is local to a function that calls another function, the value of that local name can also be used by the called function.

A name local to a function that has not completed execution or that is suspended will be inaccessible if the name is also local to a more recently called function. Putting it another way, the value of a name that you can use or display is always the most recent local value of the name. Of course, as execution of the more recently called functions is completed, the next earlier value of each local variable will again be accessible. A name can therefore be said to have one *active* referent or value, and possibly several *latent* referents or values. For example:

```
      ∇ DAN;XX
[1]     XX←100
[2]     'THE FUNCTION DAN GIVES XX THE VALUE'
[3]     XX
[4]     'AND CALLS THE FUNCTION DAVE'
[5]     DAVE
[6]     'WHEN DAVE IS THROUGH EXECUTING AND EXECUTION RETURNS'
[7]     'TO DAN, XX ONCE AGAIN HAS THE VALUE'
[8]     XX∇

      ∇ DAVE;XX
[1]     XX←200
[2]     'THE FUNCTION DAVE GIVES XX THE VALUE'
[3]     XX
[4]     'AND CALLS THE FUNCTION JERRY'
[5]     JERRY
[6]     'WHEN JERRY IS THROUGH EXECUTING AND EXECUTION RETURNS'
[7]     'TO DAVE, XX ONCE AGAIN HAS THE VALUE'
[8]     XX∇

      ∇ JERRY;XX
[1]     XX←300
[2]     'THE FUNCTION JERRY GIVES XX THE VALUE'
[3]     XX
[4]     'AND RETURNS TO DAVE'∇
```

```
      DAN
THE FUNCTION DAN GIVES XX THE VALUE
100
AND CALLS THE FUNCTION DAVE
THE FUNCTION DAVE GIVES XX THE VALUE
200
AND CALLS THE FUNCTION JERRY
THE FUNCTION JERRY GIVES XX THE VALUE
300
AND RETURNS TO DAVE
WHEN JERRY IS THROUGH EXECUTING AND EXECUTION RETURNS
TO DAVE, XX ONCE AGAIN HAS THE VALUE
200
WHEN DAVE IS THROUGH EXECUTING AND EXECUTION RETURNS
TO DAN, XX ONCE AGAIN HAS THE VALUE
100

      XX
VALUE ERROR
      XX
      ^
```

The)SINL command causes the SINL list (state indicator with name list; see the ⎕NL function in Chapter 5) to be displayed. The SINL list contains a complete set of referents of a name.

Note: See *System Functions* in Chapter 5 for an example of a local user-defined function using the ⎕FX system function.

If the SINL list is scanned downward, the first occurrence of a variable name is its active referent. If the name appears again, it is a latent referent. Global names are not found in this list; they can be displayed with the)VARS command and)FNS command.

In the following SINL display, variable P has referents as follows:

```
        )SINL
 G[7]   *Z X I
 F[4]    P J ←———— The active referent of P is local to function F.
 Q[3]   *C X T
 R[2]    P ←————— First latent referent of P is local to function R.
 G[3]    Z X I
```

As the state indicator is cleared, latent referents become active.

INTERACTIVE FUNCTIONS

User-defined functions can display messages and/or request input from the keyboard. The messages (character data) in the user-defined function are enclosed in quotes. The ⎕ (quad) and ⍞ (quad quote) symbols are used to request input from the keyboard during function execution. The following function is an example of an interactive function that computes the amount of interest on a capital amount for a given number of years:

```
        ∇ CI
[1]       'ENTER THE CAPITAL AMOUNT IN DOLLARS'
[2]       A←⎕
[3]       'ENTER THE INTEREST IN PERCENT'
[4]       I←⎕
[5]       'ENTER THE PERIOD IN YEARS'
[6]       Y←⎕
[7]       'THE RESULT IS'
[8]       A×(1+0.01×I)*Y∇


        CI
ENTER THE CAPITAL AMOUNT IN DOLLARS
⎕:
        100
ENTER THE INTEREST IN PERCENT
⎕:
        8
ENTER THE PERIOD IN YEARS
⎕:
        2
THE RESULT IS
116.64
        CI
ENTER THE CAPITAL AMOUNT IN DOLLARS
⎕:
        1000
ENTER THE INTEREST IN PERCENT
⎕:
        8.88
ENTER THE PERIOD IN YEARS
⎕:
        5
THE RESULT IS
1530.2
```

Requesting Keyboard Input during Function Execution

The ⎕ (quad) appearing anywhere other than immediately to the left of the assignment arrow indicates that keyboard input is required. When the ⎕ is encountered in the function, the two symbols ⎕: (a quad symbol followed by a colon) are displayed, the display is moved up one line, and the cursor appears. The quad and colon symbols are displayed to alert the user that input is required. Any valid expression entered at this point is evaluated, and the result is substituted for the quad. You can escape from a quad input request by entering the right arrow →.

An invalid entry in response to request for input results in an appropriate error message, and the request for input is made again. Any system commands entered will be executed, after which the request for input will again be made. An empty input (no keying) is rejected, and the 5110 again displays the symbols ⎕: and awaits input.

When the quad quote ⍞ (a quad overstruck with a quote) is used, input from the keyboard is treated as character data. The input begins at the left margin of the display; quotes do not need to be entered to define the data as character data. When ⍞ input is requested, the symbols ⎕: do not appear as they did with a ⎕ input request. The input is entered after the flashing cursor appears on the screen. For example:

```
      X←⍞
CAN'T
      X
CAN'T
      X←⍞
'CAN''T'
      X
'CAN''T'
```

Anything you enter in response to a quad quote request for input is considered character input. Therefore, if you enter a system command or a branch arrow (→) to terminate the function, the entry is treated as character data for the function, and the system command or branch will not be executed. This can be a problem if you are trying to escape from a quad quote input request. Therefore, APL provides an escape for this situation. To escape from a quad quote input request, enter the ⍠ symbol by holding the CMD key and pressing the [- +] key. The function is interrupted, and the function name and the line number being executed are displayed. You can then modify the function or terminate it by entering the right arrow →.

ARRANGING THE OUTPUT FROM A USER-DEFINED FUNCTION

The output from user-defined functions can be arranged by the format function (see the ⍕ function in Chapter 4) or bare output. Bare output is discussed next.

Bare Output

After normal output, the cursor is moved to the next line so that the next entry (either input or output) will begin at a standard position. However, bare output, denoted by the form ⍞←X (X can be any expression), does not move the cursor to the next line. Therefore, more than one variable or expression can be displayed on the same line. For example:

```
        ∇ X TIMES Y
[1]       ⍞←X
[2]       ⍞←' TIMES '
[3]       ⍞←Y
[4]       ⍞←' IS '
[5]       X×Y∇
        2 TIMES 4
 2 TIMES 4 IS 8
```

Because the cursor does not return to the next line after bare output, when quad quote (⍞) input is entered following the bare output, the input starts after the last character of the bare output. Then when the input is processed, it is prefixed by any bare output on the input line. For example:

```
        ∇OUTPUT∆INPUT
 [1] ⍞←'THIS IS BARE OUTPUT!!!'
 [2] ⍝THE NEXT STATEMENT REQUESTS ⍞ INPUT
 [3] IN←⍞
 [4] ⍝NOW DISPLAY THE INPUT
 [5] ∇
        OUTPUT∆INPUT
 THIS IS BARE OUTPUT!!!_
 THIS IS BARE OUTPUT!!!THIS IS ⍞ INPUT
```

The cursor appears here. Now enter THIS IS ⍞ INPUT.

After EXECUTE is pressed, the output line looks like this.

Therefore, if quad quote input follows bare output (but only the input is to be processed), the bare output must be removed from the input line. Following is an example of a function that will remove the bare output:

```
      ∇R←BAREΔOUTPUT MSG;⎕IO;J
[1]   ⎕IO←1
[2]   ⍞←MSG
[3]   ⍝CHECK THE BARE OUTPUT FOR EMBEDDED CURSOR RETURNS
[4]   J←¯1+(⌽MSG)⍳⎕AV[157]
[5]   ⍝DROP ANY BARE OUTPUT PREFIX FROM THE INPUT
[6]   R←((64|J-1)+1)↓⍞∇
```

This is how the function works:

```
      ∇OUTAIN                                     ← The Bare Output
[1]   BAREΔOUTPUT 'THIS IS BARE OUTPUT!!!'
[2]   ∇          ← This function will remove the bare output.
      OUTAIN
THIS IS BARE OUTPUT!!! ← The cursor appears here.  Now enter THIS IS ⍞ INPUT.

THIS IS BARE OUTPUT!!!THIS IS ⍞ INPUT
THIS IS ⍞ INPUT ← This is the final result.
```

If bare output consists of a character vector and backspace characters, the display screen is scrolled down one line when the number of backspace characters is greater than the number of characters in the character vector. For example:

```
⎕IO←1
⍞←(10⍴'A'), 11⍴ ⎕AV[159] ←—— Backspace Character from the
                             Atomic Vector.
```

LOCKED FUNCTIONS

A locked function can only be executed, copied, or erased; it cannot be revised or displayed in any way, nor can trace control and stop control (see *Trace Control* and *Stop Control* later in this chapter) be changed. A function is locked, or protected, when the function definition is opened or closed with a ⍫ (∇ overstruck with ~) instead of a ∇.

When an error is encountered in a locked function, execution of that function is abandoned (not suspended). If this function was invoked by another locked function, execution of the second function is abandoned also, and so on, until either (1) a statement in an unlocked function or (2) an input statement is reached. Then an error message is displayed. In the first case, the execution of the unlocked function is suspended at the statement; in the second case, the 5110 waits for input.

Note: A locked function cannot be unlocked; therefore, if the function contains an error, the function cannot be edited and the error corrected.

FUNCTION EDITING

Several methods are used when the system is in function definition mode to display and revise a user-defined function. Also, after a function definition has been closed, the definition can be reopened and the same methods used for further revisions or displays. (See *Reopening Function Definition* in this chapter.)

Displaying a User-Defined Function

Once the system is in function definition mode, part or all of a user-defined function can be displayed as follows:

- To display the entire function, including the function header and the opening and closing ∇, enter [⎕]. APL responds by displaying the function, then waiting for the entry of additional statements.
- To display from a specified statement to the end of the function, enter [⎕n], where n is the specified statement number. APL responds by displaying the function from statement n to the end of the function, then waiting for the last statement displayed to be edited (see *Editing Statements* in this chapter).
- To display only one statement of the function, enter [n⎕], where n is the statement number to be displayed. APL responds by displaying statement n and waiting for the statement to be edited (see *Editing Statements* in this chapter).

The following table summarizes function display when the system is in function definition mode:

Entry	**Result**
[⎕]	Displays all statements
[⎕n]	Displays all statements from n onward
[n⎕]	Displays statement n

Revising a User-Defined Function

Statements in a user-defined function can be replaced, added, inserted, deleted, or edited as follows:

- To replace statement number n, enter [n] and the replacement statement. If just [n] is entered, APL responds with [n], then waits for the replacement statement to be entered. If the function header is to be replaced, enter [0] and the new function header.

- To add a statement, enter [n] (n can be any statement number beyond the last existing statement number) and the new statement. APL will respond with the next statement number, and additional statements can be entered if required.

- To insert a statement between existing statements, enter [n] and the new statement. n can be any decimal number with up to four decimal digits. For example, to insert a statement between statements 8 and 9, any decimal number between 8.0000 and 9.0000 can be used. APL will respond with another decimal statement number, and additional statements can be inserted between statements 8 and 9 if required. (These and the following statements are automatically renumbered when the function definition is closed.)

  Note: The statement number 9999.9999 is the last valid statement number.

- To delete statement n, enter [∆n] or display statement n, press the ATTN key to delete the line, and then press the EXECUTE key.

  Note: The [∆n] and closing ∇ cannot be entered on the same line. If the function definition is to be closed immediately after a statement has been deleted, the closing ∇ must be entered on the next line.

- To edit a specific statement, use the following procedure:

  1. Enter [n⎕] (where n is a statement number). Statement n is displayed.

     Note: The scroll keys (↑ ↓) can be used to position a statement on the display screen to the input lines 0 and 1 for editing. To do this, use the scroll down (↓) key to position the statement to lines 0 and 1. If the statement does not require two lines, position the statement on line 1 and make sure another statement is not displayed on line 0.

  2. Choose one of the following options:
     a. To change a character, position the cursor (flashing character) at the character to be changed. Enter the correct character.
     b. To delete a character, position the cursor at the character to be deleted. Then press the backspace (←) key while holding the command (CMD) key. The character at the cursor is deleted from the line, and the characters that were to the right of the deleted character are moved one position to the left.
     c. To insert a character, position the cursor to the position where the character is to be inserted. Then press the forward space (→) key while holding down the command (CMD) key. The characters from the cursor position to the end of the line are moved one position to the right. For example: [1] A←1245 should be [1] A←12345. Position the cursor at the 4, and press the forward space and command (CMD) keys simultaneously. The display will look like this: [1] A←12_45. Now enter the 3.
     d. To delete all or part of a line, press the ATTN key to delete everything from the cursor position to the end of the line.

  3. Press the EXECUTE key. The next statement number is displayed.

Note: If more than one statement number is entered on the same line, only the last statement number is used. For example, if a line contained [3] [8] [4] 'NEW LINE', only statement 4 is replaced when the EXECUTE key is pressed.

Reopening Function Definition

If you want to edit a function that has previously been closed, the function definition must be reopened. For example, if function R is already defined, you reopen the function definition for function R by entering ∇R. The rest of the function header must *not* be entered or the error message DEFN ERROR is displayed and the function definition is not reopened. The 5110 responds by displaying [n+1], where n is the number of statements in R. Function editing then proceeds in the normal manner.

Function definition can also be reopened and the editing or display requested on the same line. For example, ∇R[3]S←S+1 edits the function by entering the new line 3 (S←S+1) immediately. Then the 5110 responds by displaying [4] and awaiting continuation. The entire process can be accomplished on a single line: ∇R[3]S←S+1∇ opens the definition of function R, enters a new line 3, and terminates function definition. ∇R[⎕]∇ causes the entire definition of R to be displayed, after which the 5110 returns to execution mode.

Note: You cannot reopen the definition of a function, delete a statement, and close the function (for example, ∇R[∆4]∇) on the same line, because the closing ∇ cannot be on the same line as the [∆n].

When an error occurs in a function, the function name, the line number, and the statement in error are displayed. A caret on the following line indicates where the 5110 stopped execution of the statement. The statement in error can be corrected as follows:

1. Scroll down until the caret is removed from the screen.

2. Scroll up one line.

3. Insert a ∇ before the function name.

4. Correct the error in the statement.

5. Place a ∇ after the statement.

6. Press the EXECUTE key.

This procedure works only if the complete statement is displayed.

An Example of Function Editing

In this example, the user-defined function AVERAGE is used to show how the methods used to revise and display functions work:

```
      ∇AVERAGE X                                              Define the function.
[1]  'THIS FUNCTION CALCULATES AVERAGES'
[2]  +/X÷(+/X=X)∇
      AVERAGE 2 4 6 8                                         Execute and test the function.
THIS FUNCTION CALCULATES AVERAGES
5

      ∇AVERAGE[1.7] 'THE AVERAGE IS'                          Insert a statement.
[1.8] [1⎕]                                                    Display statement 1.
[1]   'THIS FUNCTION CALCULATES AVERAGES'
[1]   'THIS FUNCTION CALCULATES AVERAGES AND SUMS'            Statement 1 was edited
[2]  [3] 'THE SUM IS'                                         to look like this.
[4]  +/X∇                                                     Add statements 3 and 4.
      ∇AVERAGE[⎕]∇                                            Display the function.
    ∇ AVERAGE X
[1]   'THIS FUNCTION CALCULATES AVERAGES AND SUMS'
[2]   'THE AVERAGE IS'
[3]   +/X÷(+/X=X)
[4]   'THE SUM IS'
[5]   +/X
    ∇

      AVERAGE 2 4 6 8                                         Execute average.
THIS FUNCTION CALCULATES AVERAGES AND SUMS
THE AVERAGE IS
5
THE SUM IS
20

      ∇AVERAGE [3] +/X÷ρX                                     Replace statement 3.
[4]  [∆1]                                                     Delete statement 1.
[2]  [0]AVERAGE∆SUM X∇                                        Replace the function header.

      ∇AVERAGE∆SUM[⎕]∇                                        Display the function.
    ∇ AVERAGE∆SUM X
[1]   'THE AVERAGE IS'
[2]   +/X÷ρX
[3]   'THE SUM IS'
[4]   +/X
    ∇

      AVERAGE∆SUM 2 4 6 8
THE AVERAGE IS
5
THE SUM IS
20

      ∇AVERAGE∆SUM [⎕3]∇                                      Display the function from
[3]   'THE SUM IS'                                            statement 3 to the end.
[4]   +/X
```

TRACE AND STOP CONTROLS

APL provides the ability to trace or stop execution of user-defined functions, providing the functions are not locked (see *Locked Functions* in this chapter).

Trace Control

Trace control is used to display the results of selected statements as a function executes. The display consists of the function name followed by the number and results of the selected statement. For example:

```
  STEVE[1] 2←———Result
Function   Statement
Name       Name
```

Statements to be treated are specified by a trace vector. The format of the trace control function is TΔ STEVE←V, where STEVE is the name of the function and V is the vector specifying the statement numbers to be traced. For example, if TΔ STEVE←2 3 5 is entered, the statement 2, 3, and 5 are traced each time function STEVE is executed. TΔ STEVE←⍳0 must be entered to discontinue the tracing of function STEVE. To trace each statement of the function, enter TΔ STEVE←⍳N, where N is the number of statement in the function:

```
      ∇STEVE I
[1]  A←1×I
[2]  B←2×I
[3]  C←3×I
[4]  D←4×I
[5]  A+B+C+D∇
      STEVE 2
20
      TΔSTEVE←⍳4
      STEVE 2          Trace the first four statements
STEVE[1] 2             in function STEVE.
STEVE[2] 4
STEVE[3] 6
STEVE[4] 8
20
      TΔSTEVE←2 ←———Trace statement 2 in function STEVE.
      STEVE 2
STEVE[2] 4
20
      TΔSTEVE←⍳0 ←——— Discontinue tracing in function STEVE.
      STEVE 2
20
```

Trace control can also be set by statements within a function. These statements initiate tracing when a variable contains a certain value. For example:

```
        ∇STEVE[⎕]∇
      ∇ STEVE I
[1]     A←1×I
[2]     T∆STEVE←3×A=2←——— Trace statement 3 in function STEVE
[3]     C←3×I                when A equals 2.
[4]     D←4×I
[5]     A+C+D
      ∇
        STEVE 2
STEVE[3] 6
16
        STEVE 3
24
```

Note: The following instruction establishes trace control for the first statement of each user-defined function in the active workspace:

```
⍎¯1↓,'0','ρ','T','∆',(((⎕NL 3),'←'),'1'),','
```

This instruction can be used to find out what functions are called by another function.

The following user-defined function named TRACE establishes a trace vector for each statement in a specified user-defined function:

```
        ∇TRACE NAME
[1]  ⍎'T∆',NAME,'←⍳1↑ρ⎕CR''',NAME,''''∇
```

When the function TRACE is executed, the argument must be entered in single quotes. For example:

```
        ∇STEVE[⎕]∇
     ∇ STEVE I
[1]    A←1×I
[2]    B←2×I
[3]    C←3×I
[4]    D←4×I
[5]    A+B+C+D
     ∇
        TRACE 'STEVE'←— Establish a trace vector for each
        STEVE 2          statement in function STEVE.
STEVE[1] 2
STEVE[2] 4←———————— Each statement of function
STEVE[3] 6               STEVE has been traced.
STEVE[4] 8
STEVE[5] 20
        T∆STEVE←⍳0
        STEVE 2
20
```

Stop Control

Stop control is used to stop the execution of a function just before specified statements. At each stop, the function name and statement number of the statement to be executed next is displayed. The statements are specified by a stop vector. The format of the stop control function is S∆ STEVE←V, where STEVE is the name of the function and V is the vector specifying the statements. After the stop, the system is in the suspended state; to resume execution, →⎕LC must be entered (see Chapter 5). S∆STEVE←⍳0 (STEVE is the function name) must be entered to discontinue the stop control function.

Stop control can be set by statements within a function. These statements initiate halts when a variable contains a certain value. For example, S∆ STEVE←4×N>8 means stop before statement 4 in function STEVE when N is greater than 8.

Trace control and stop control can both be used in the same user-defined function. An attempt to set trace control or stop control for a nonexistent function creates a variable and causes a syntax error. For example:

```
        )CLEAR
CLEAR WS
        S∆F←1 2 3
SYNTAX ERROR
        S∆F← 1 2 3
        ^
        )VARS
F
        F
1 2 3
```

Chapter 7. Tape, Diskette, Display Screen, and Printer Input and Output

When input and output involve the tape, diskette, display screen, or printer, an APL shared variable can be established as the connection between the active workspace and the I/O processor. That is, the value of the shared variable is shared between the active workspace and the I/O processor.

There are two types of input/output operations: sequential access to tape and diskette files, and direct access to diskette files and the display screen. There is also an output operation to the printer. Sequential access means that the records must be written to and read from the data file in sequential order. Direct access means that specified records are directly read from or written to the data file or display screen.

Input and output operations require the followiṇg steps:

1. Make a shared variable offer.

2. Specify the operation to be performed.

3. Transfer the data (perform the input/output operations).

4. Terminate the operation.

5. Retract the shared variable offer.

MAKING A SHARED VARIABLE OFFER

To establish a shared variable between the actvie workspace and the I/O processor, the ⎕SVO (share variable offer) system function is used. This function is dyadic (requires two arguments), where the left argument is a 1 (which specifies the 5110 I/O processor) and the right argument is character data that specifies the variable name(s) being offered as a shared variable. The 5110 allows up to 12 variable names to be shared. These variable names can be offered in one or more statements. However, if more than one name is used as the right argument of the ⎕SVO function, the names must be entered as a character matrix with each row representing one name. For example:

```
1 ⎕SVO 4 1 ⍴'ABCD'
```

or   Each row represents a separate variable name.

```
NAMES←4 1⍴'ABCD'
1 ⎕SVO NAMES
```

Direct access input and output operations require a pair of shared variables. One variable name must start with the characters CTL, and the other name must start with the characters DAT. After the first 3 characters, the next 15 characters in each pair of names must be identical. For example, 1 ⎕SVO 2 4⍴ 'CTLXDATX' offers CTLX and DATX as shared variables.

The explicit result of the ⎕SVO system function is a 2, 1, or 0 for each variable name offered to be shared. For example:

```
      1 ⎕SVO 4 1⍴'ABCD'
2 2 2 2
```

A 2 indicates that the name is successfully shared with the I/O processor, a 1 indicates that a value other than 1 was used as the left argument for the ⎕SVO function, and a 0 indicates the shared variable offer failed. When a 1 is the result, the variable name is not established as a shared variable. When a 0 is the result, an error message is also displayed (see Appendix F).

Note: Using the ⎕SVO function *without* a left argument returns an explicit result of 2 or 0 for each name specified in the right argument. A 2 indicates that the name is shared with the 5110 I/O processor, and a 0 indicates that the name is not a shared variable.

SURROGATE NAMES

When offering variable names to be shared, you can establish surrogate names by specifying two variable names in the character vector or in each row of the character matrix. The first name is the surrogate, and the second name is the variable to be shared.

For example:

```
      1 ⎕SVO 'C CTL'
2
```

CTL is the variable name to be shared, and C is the surrogate.

```
NAMES←2 5⍴'C CTLD DAT'
```

NAMES

C CTL — CTL and DAT are the variable names to be

D DAT — shared, and C and D are the surrogates.

1 ⎕SVO NAMES

2 2 — An explicit result is returned for the shared variable names only.

Once a surrogate name is established, the surrogate name must be used in place of the shared variable name during input/output operations.

SPECIFYING THE OPERATION TO BE PERFORMED

Once the shared variables are established, the next step is to specify the operation to be performed. These operations consist of:

- Sequential access to tape or diskette files
- Direct access to diskette files
- Direct access to the display screen
- Sending data to the printer

After the operation is specified, the I/O processor assigns a return code to the shared variable that specified the operation. A 0 n or 22 n return code indicates that the operation was specified successfully, and any other return code indicates that specifying the operation failed. (See *Return Codes* later in this chapter.) How to specify the input/output operations is discussed next in this chapter.

Note: If a variable has a value when the name is offered to be shared, the 5110 establishes the name as a shared variable and then attempts to specify the operation to be performed using the value already assigned to the shared variable. If the data assigned to the shared variable is not valid for specifying an operation to be performed, an error message is displayed and the appropriate return code is assigned to the shared variable. To prevent this error condition, you must make sure that the data assigned to the variable is valid for specifying the operation to be performed or expunge (⎕EX) the name before the shared variable offer is made.

Sequential Access to Tape or Diskette Files

When you are creating, adding data to, or reading a sequential access data file, you must use a single shared variable to access the file. When using tape, you can have only one open file on the tape at a time. An *open file* is a data file that is currently being used for input/output operations. When using diskette, you can have up to 12 open files at a time (using one shared variable for each file).

The first data assigned to the shared variable must be a character vector that specifies one of the following operations:

- Create a new sequential access data file.
- Add data to an existing sequential access data file.
- Read data from a sequential access data file.

To specify one of these operations, the character vector must be in one of the following formats (enclosed in single quotes):

A←'OUT device/file number ID=(file ID) [MSG=OFF] $\left[\text{TYPE=}\begin{matrix}\text{A}\\ \text{I}\end{matrix}\right]$'
A←'ADD [device/file number] [ID=(file ID)] [MSG=OFF]'
A←'IN [device/file number] [ID=(file ID)] [MSG=OFF]'

There must be at least one blank between parameters. Following is a description of each parameter.

OUT, ADD, or IN

This first parameter specifies whether the data is to be written to a new sequential access file (OUT), written to an existing sequential access file following the last record in the data file (ADD), or read from an existing sequential access file (IN).

device/file number

This parameter specifies the appropriate diskette drive or tape unit and file number. For example:

11003
File Number 3
Diskette Drive 1

If fewer than four digits are used, the default device is assumed (see Chapter 2), and this parameter represents only the file number. This parameter is optional for diskette files only. See *Name Processing* in Chapter 2 for more information on specifying the device/file number for diskette files.

ID = (file ID)

This parameter specifies the file identification (the name is enclosed in parenthesis) and is used as follows:

- When a new data file (OUT operation) is created, the name is put in the file ID field of the file header (see *The)LIB Command* in Chapter 3). If this parameter is not specified for tape operations, the file ID defaults to DATA. This parameter must be specified for diskette operations.

  It is a good practice to give the data files meaningful names; for example, a file that contains sales data could be named sales–ID=(SALES).

  Notes:
  1. See *Variable and File Names* and *Name Processing* in Chapter 2 for more information on specifying the file ID.
  2. For an OUT operation to an existing data file (writing new data over the existing data), the file ID specified must match the existing file ID for the data file or the)DROP command must be used to drop the data file (see Chapter 3).

- When you are adding to or reading from an existing data file (ADD or IN operations), the file ID, if specified, is compared to the file ID in the file header; the operations cannot be performed if these file IDs do not match.

MSG = OFF

This parameter is optional. When this parameter is specified, no error message is displayed for nonzero return codes (see *Return Codes* later in this chapter), but the return code is still assigned to the shared variable.

TYPE = A or I

This parameter is specified only when you are creating a new data file (OUT operations). This parameter specifies the type of file and data format to be used for writing data to a sequential access file:

- When TYPE=A is specified, the APL internal data format is used (file type 08–see *The)LIB Command*); that is, the data is written to the data file in the same format it has in the active workspace.

  Note: TYPE=A is the default if this parameter is not specifed.

- When TYPE=I is specified, the general exchange data format is used (file type 02). When the general exchange data format is used, only character scalars and vectors can be assigned to the shared variable. Therefore, before you store numeric data or arrays on data files using the exchange data format, you must first use the format (⍕) function to change the data to a character scalar or vector. This data file can also be used as a BASIC language source file. When you are adding to or reading from a data file (IN and ADD operations), the data format is determined by the existing data file type.

Direct Access to Diskette Files

When you are creating or adding data to a direct-access diskette file, you use a single shared variable to sequentially write the records to the file. When you are reading from or writing to an existing direct-access data file, you use a pair of shared variables to read or write specified records. In this case, the shared variable name with the CTL prefix must be used to specify the operation. You can have from 6 to 12 open diskette data files at one time. The number of open diskette files allowed depends on how many of the files are existing direct access files, because a pair of shared variables is required to access these files.

The first data assigned to the shared variable must be a character vector that specifies one of the following operations:

- Create a new direct access data file.

- Add data to an existing direct access file.

- Read data from a direct access data file.

- Update (read from and write to) a direct access data file.

To specify one of these operations, the character vector must be in one of the following formats (enclosed in single quotes):

```
                                                        ┌     A ┐
A←'OUTF  device/file number  ID=(file ID)  [MSG=OFF]    │      I │'
                                                        │TYPE= U │
                                                        └     M ┘
A←'ADD  [device/file number]  [ID=(file ID)]  [MSG=OFF]'
CTL←'INR  [device/file number]  [ID=(file ID)]  [MSG=OFF]  [TYPE=N]'
CTL←'IOR  [device/file number]  [ID=(file ID)]  [MSG=OFF]  [TYPE=N]'
CTL←'IORH  [device/file number]  [ID=(file ID)]  [MSG=OFF]  [TYPE=N]'
```

There must be at least one blank between parameters. Following is a description of each parameter.

OUTF, ADD, INR, IOR, IORH

This first parameter specifies whether the data is to be:

- Written to a new direct access file (OUTF)
- Added to an existing direct access file after the last record in the data file (ADD)
- Read only from a direct access file (INR)
- Read from and/or written to a direct access file (IOR and IORH)

The results of an IOR and IORH operation are the same. When you specify IORH, an update operation (read and write) is performed faster than if you had specified IOR. However, the IORH operation requires more storage in the active workspace than does the IOR operation.

Note: When INR, IOR, or IORH operations are specified, a 0 n or 22 n return code indicates that the operation was successfully specified. The second element of the return code (n) indicates the number of records in the data file.

device/file number

This parameter specifies the appropriate diskette drive and file number. For example:

```
11003
 ↑  ↖
 │    File Number 3
 │
Diskette Drive 1
```

If fewer than four digits are used, the default device is assumed (see Chapter 2), and this parameter represents only the file number.

This parameter is optional for diskette only. See *Name Processing* in Chapter 2 for more information on specifying the device/file number for diskette files.

ID = (file ID)

This parameter specifies the file identification (the name is enclosed in parentheses) and is used as follows:

- When a new data file (OUTF operation) is created, the name is put in the file ID field of the file header (see *The)LIB Command* in Chapter 3). This parameter must be specified for OUTF operations.

  It is a good practice to give the data files meaningful names; for example, a file that contains sales data could be named sales–ID = (SALES).

  Notes:
  1. See *Variable and File Names* and *Name Processing* in Chapter 2 for more information on specifying the file ID.
  2. For an OUTF operation to an existing data file (writing new data over the existing data), the file ID specified must match the existing file ID for the data file or the)DROP command must be used to drop the data file (see Chapter 3).

- When you are adding to (ADD operations) or reading from (INR, IOR, IORH operations) an existing data file, the file ID, if specified, is compared to the file ID in the file header; the operations cannot be performed if these file IDs do not match.

MSG = OFF

This parameter is optional. When this parameter is specified, no error message is displayed for nonzero return codes (see *Return Codes* later in this chapter).

TYPE = A, I, U, or M

This parameter is specified only when you are creating a new direct access data file (OUTF operations). This parameter specifies the type of file and data format to be used for writing data to a direct access file:

- When TYPE=A is specified, the APL internal data format is used (file type 10–see *The)LIB Command* in Chapter 3); that is, the data is written to the data file in the same format it has in the active workspace. The first record written to the data file determines the shape (⍴DATA) and representation that the remaining records must have when these records are written to the data file. Each record written to the data file must have the same shape and internal representation because if more than one record at a time is to be read from the file, the records are laminated together (see *Update (Read and Write) a Direct Access Data File (IOR and IORH)* later in this chapter). Following are the classifications for data internal representation:
  - Character
  - Numeric binary (all zeros and ones)
  - Numeric fixed point (all integers in the range -2^{31} to $2^{31}-1$
  - Numeric floating point (all other values)

  Data can be converted from one internal representation to another (by APL built-in functions) as follows:

  ```
  CHARACTER←⍕NUMERIC
  BINARY←1∧NUMERIC ←——This numeric data must be zeros and ones;
  FIXED←⌊NUMERIC ←—┐  however, the zeros and ones might be
  FLOAT←NUMERIC + 0.0 │  fixed-point or floating-point internal
  NUMERIC←⍎CHARACTER  │  representation.
                      │
                      └─ This numeric data must be in the
                         range -2³¹ to 2³¹ -1.
  ```

  Note: TYPE=A is the default if this parameter is not specified.

- When TYPE=I is specified, the exchange data format is used (file type 09). When the exchange data format is used, only character scalars or vectors can be assigned to the shared variable. Therefore, when you are storing numeric data on data files using the exchange data format, you must first use the format (⍕) function to change the data to a character scalar or vector. The first record written to the data file determines the shape (⍴DATA) that the remaining records must have when the records are written to the data file. Each record written to the data file must have the same shape because if more than one record at a time is to be read from the file, the records are laminated together (see *Update (Read and Write) a Direct Access Data File (IOR and IORH)* later in this chapter). The data file can also be used as a BASIC language source file.

- When TYPE=U is specified, the data assigned to the shared variable can only be character scalars or vectors. The data is unblocked and unspanned as it is written to the data file (file type 09 or B9); that is, each record starts on a sector boundary, and a record cannot exceed one sector. Therefore, each record must be equal to or less than one sector.

  Note: The file type B9 is the basis for general exchange with other products. The 5110 assigns the file type B9 to a data file when:

  - TYPE=U is specified when you are creating the data file.
  - The file name is a simple name; that is, the file name cannot exceed 8 alphameric characters.
  - The records in the file are not greater than 128 characters.
  - The diskette sector size is 128 or 256 bytes.

  Also, you can read TYPE=U (file types 09 and B9) data files sequentially (see *IN Operations* in this chapter) using one shared variable.

- When TYPE=M is specified, the APL internal data format is used (file type 15); that is, the data is written to the data file in the same format it has in the active workspace. The first record written to the file determines how much file storage is allocated for each additional record. For example, if the first record written to the file uses 100 bytes of file storage, each additional record written to the file is allocated 100 bytes of file storage (see *Storage Requirements* in Chapter 2 for information on how many bytes of storage are required for each data type). For TYPE=M data files, the records do not have to be the same shape or representation; therefore, only one record at a time can be read from a TYPE=M data file.

When you are adding records to or updating (reading and writing) a data file (ADD, IOR, IORH, and INR operations), the data format is determined by the existing data file type.

TYPE = N

This parameter is optional and can be specified only when you are reading from or writing to an existing diskette file (IOR, IORH, and INR operations). When TYPE=N is specified, all attributes of the file are ignored and data is not translated from APL internal code to EBCDIC during output operations, or from EBCDIC to APL internal code during input operations.

When TYPE=N is specified, the system treats each sector in the file as one individual record. For example, for a 10K data file on a 512-byte-per-sector diskette, there would be 20 records in the file. Also, when TYPE=N is specified, only character scalars or vectors can be assigned to the shared variable.

Note: When TYPE=N is specified, the file header can be read (but not written) if you specify record number -1. (For information on specifying record numbers, see *Direct Access (OUTF, ADD, INR, IOR, IORH, and DISPLAY)* in this chapter.) The file header is always a 128-byte record.

Direct Access to the Display Screen

When you are reading data from or writing data to the display screen, specified records are read or written using a pair of shared variables. The shared variable name with the CTL prefix is used to specify the operation. The first data assigned to the shared variable must be a character vector in the following format (enclosed in single quotes):

```
CTL←'DISPLAY  [MSG=OFF]'
```

There must be at least one blank between the parameters. Following is a description of each parameter.

DISPLAY

This first parameter specifies that the pair of shared variables are to be used to read data from and/or write data to the display screen.

MSG = OFF

This parameter is optional. When this parameter is specified, no error message is displayed for nonzero return codes (see *Return Codes* later in this chapter).

Send Data to the Printer

The first data assigned to the shared variable must be a character vector in the following format (enclosed in single quotes):

```
A←'PRT  [MSG=OFF]'      A can be any shared variable.
```

There must be at least one blank between the parameters. Following is a description of each parameter.

PRT

This first parameter specifies that the shared variable is to be used to send data to the printer.

MSG = OFF

This parameter is optional. When this parameter is specified, no error message is displayed for nonzero return codes (see *Return Codes* later in this chapter).

INPUT/OUTPUT OPERATIONS

After the input/output operation is specified, the actual read and/or write operations can be performed.

Sequential Access Operations

When a shared variable is used to sepcify a sequential access input or output operation, that shared variable is used to transfer the data.

Writing Data to a Data File (OUT and ADD)

Each time data is assigned to the shared variable, the data is written to the data file and the I/O processor assigns a return code to the shared variable. A 0 0 return indicates that the data was written to the data file successfully, and a nonzero return code indicates that the operation failed. See *Return Codes* in this chapter for a description of each return code.

For OUT operations to an existing data file (writing new data over the existing data), all the previous data in the data file can no longer be accessed.

Reading Data from a Data File (IN)

Each time the shared variable is referenced in the active workspace, the I/O processor assigns a new record to the shared variable. The data is read from the data file and assigned to the shared variable in the same sequence as the data was previously written to the data file.

Because a new record is assigned to the shared variable each time the shared variable is referenced, no return code is assigned to the shared variable after each IN operation. An empty vector is returned to indicate an end-of-file or an error condition, and the proper return code is then assigned to the shared variable.

Note: Unpredictable results occur if ⎕AV[157] (hex 15) or ⎕AV[245] (hex 1E) characters are written to and then read from a TYPE=I (file type 2) data file. These characters are special record delimiters for the 5110.

Direct Access Operations

Creating a Direct Access Data File (OUTF and ADD)

When a shared variable is used to specify an OUTF operation, or an ADD operation to an existing direct access file, that shared variable is then used to write data to the data file. Each time data is assigned to the shared variable, the data is sequentially written to the data file and the I/O processor assigns a return code to the share variable. A 0 0 return code indicates that the data was written to the data file successfully, and a nonzero return code indicates that the operation failed. See *Return Codes* in this chapter for a description of each return code.

For an OUTF operation to an existing data file (writing new data over the existing data), any existing data following the new data cannot be used again.

Updating (Read and Write) a Direct Access File (IOR and IORH)

To read from and write to a direct access data file requires a pair of shared variables. One of the shared variable names must have the prefix CTL, and the other name must have the prefix DAT. After the prefix, the next 15 characters in each name must be identical, if specified. For example:

CTLANAME

DATANAME

The shared variable name with the CTL prefix is used to send control information to the I/O processor. Also, the I/O processor assigns the return codes to this shared variable. The shared variable with the prefix DAT is used to send data to or receive data from the data file.

Note: When updating a direct access file, you cannot add records to the end of the file. Instead, you must do an ADD operation.

The control information assigned to the shared variable with the CTL prefix must be integers in a two- or three-element vector as follows:

```
      0
CTL←1  starting record number   [number of records]
      2
```

Where:

- The first element specifies to read from (0), write to (1), or search by key (2) the data file.

- The second element specifies the starting record number to read, write, or search by key. For direct access data files, the first record in the file is always number 0. Therefore, if record number 1 is specified, the operation starts with the second record in the data file.

- The third element is optional. This element specifies the number of records to read, write, or search by key. If this element is not specified, one record is assumed.

Reading records, writing records, and search by key procedures are discussed next.

Reading Records

When the control shared variable specifies that records are to be read, the I/O processor assigns the specified records to the shared variable name with the DAT prefix. If more than one record is read, the records are laminated along a new first dimension before they are assigned to the DAT shared variable. For example, assume that the characters A, B, and C are the first three records in the file:

```
        CTLX ← 0 0 3
        CTLX
0 0
        ⍴DATX
3 1
        DATX
A
B
C
        DATX[1;]
A
        DATX[2;]
B
        DATX[3;]
C
```

Read three records, starting with the first record in the file.

The I/O processor assigns the return code to the CTL shared variable.

Find the shape of the data assigned to the DAT shared variable.

The new first dimension. This value also represents the number of records read.

You can specify each individual record by indexing the new dimension.

Once the data read from the data file is assigned to the DAT shared variable, the shared variable can be used like any other variable. That is, the DAT shared variable retains its value until more data is read from the data file or you assign a new value to the DAT shared variable. For example:

```
        CTLX←0 1 1
        CTLX
0 0
        DATX
B
        CTLX←0 2 1
        CTLX
0 0
        DATX
C
        DATX←'A'
        DATX
A
```

Read the second record in the file.

DATX retains this value until another record is read or you assign a new value to DATX.

Read the third record.

DATX now has this value.

Assign a new value to DATX.

DATX now has this value.

Writing Records

When the CTL shared variable specifies that records are to be written, the I/O processor writes the current value of the DAT shared variable to the data file. If more than one record is specified to be written to the data file, the records must be joined along a new first dimension before they are assigned to the DAT shared variable. For example, assume you want to write the three records A, B, and C to a data file:

```
DATX←3 1ρ'ABC'
DATX
A
B
C
```

Join the records along a new first dimension.

```
      CTLX←1 0 3
      CTLX
0 0
```

Write the three records currently assigned to the DAT shared variable to the data file.

Now the first three records in the data file are A, B, and C.

To update a direct access data file, you can read the record(s) from the data file, modify the record(s), and then write the modified record(s) back to the data file. For example:

```
CTLX←0 0 3
CTLX
0 0
      DATX
A
B
C
```

The first three records in the data file.

```
      DATX[1 2 3;]←'DEF'
      DATX
D
E
F
```

Modify the three records read from the data file.

```
      CTLX←1 0 3
      CTLX
0 0
```

Write the modified records back to the data file at a specified location.

Now, the first three records in the data file are D, E, and F.

Search by Key

Search by key can be used with direct access interchange, general interchange, unblocked/unspanned, and APL internal format data files. When the CTL shared variable specifies a search by key, the key value presently assigned to the DAT shared variable is used to search the data file. The key must be:

- The same representation as the data in the data file (character, binary, fixed point, or floating point)
- Less than or equal to 255 bytes for 256-byte-per-sector diskettes
- Less than or equal to 128 bytes for 128-byte-per-sector diskettes
- Less than or equal to the record size for unblocked/unspanned format data files (TYPE=U)

For a search by key, the records in the data file cannot span any sector boundaries, and the records should be sorted in ascending sequence. That is, one sector can contain multiple records; however, the records cannot span a sector boundary.

The key is compared with the equivalent number of beginning bytes in each sector. The search starts with the sector containing the first record specified and continues through the sector containing the last record specified (determined by the number of records specified). The search is completed when:

- The key is equal to the equivalent number of beginning bytes in the sector. In this case, the records in the sector are assigned to the DAT shared variable.
- The key is less than the equivalent number of beginning bytes in the sector. In this case, the record(s) from the previous sector are assigned to the DAT shared variable. The comparison between the key and the beginning bytes of each sector is an unsigned binary comparison (bit by bit). Therefore, the following collating sequences are implied:
  - For interchange, general interchange, and unblocked/unspanned format data files, the EBCDIC character sequence.
  - For character data in APL internal format files, the atomic vector (⎕AV) character sequence.
  - For binary data, unsigned integer sequence.
  - For numeric data, an unsigned integer sequence. That is, for fixed point data, the negative values follow the positive values. For floating point data, the negative values are in reverse order and follow the positive values.

- The EOD address is encountered as the specified records are being searched. In this case, the record(s) in the last sector of the file are assigned to the shared variable.

- All of the specified records are searched without one of the previous conditions occurring. In this case, the search is unsuccessful, and a 1 029 return code is assigned to the CTL shared variable (see *Return Codes* later in this chapter).

Notes:
1. If the key is equal to the beginning bytes of a sector, but the beginning bytes in a previous sector searched are greater than the key, the sector containing the bytes equal to the key will not be reached.
2. Characters records should be sorted in ascending sequence for search by key operations.

If the search is successful and the sector contains more than one record, the records are laminated (joined) together along a new first dimension and assigned to the DAT shared variable. Therefore, the first dimension of the DAT shared variable represents the number of records read and assigned to DAT.

After the search is successfully completed, the return code assigned to the CTL shared variable is 0 n, where n represents the record number of the first record assigned to the DAT shared variable.

Reading a Direct Access Data File (INR)

Reading the data from a direct access file using an INR operation requires a pair of shared variables.

Note: Generally, an INR operation is specified when the data is to be read from and not written to the file. This prevents data from accidently being written to the data file.

The control information assigned to the shared variable with the CTL prefix must be integers in a two- or three-element vector as follows:

$$\text{CTL} \leftarrow \begin{matrix}0\\2\end{matrix} \ \text{starting record number} \quad [\text{number of records}]$$

Where:

- The first element must be 0 (specifying that data is to be read from the file) or 2 (specifying a search by key operation).

- The second element specifies the starting record number. For direct access data files, the first record in the file is always number 0. Therefore, if record number 3 is specified, the operation starts with the fourth record in the data file.

- The third element is optional. This element specifies the number of records. If this element is not specified, one record is assumed.

See *Reading Records* and *Search by Key* earlier in this chapter for a description of how records are read from a data file or data files are searched by key, respectively.

Reading from and Writing to the Display Screen (DISPLAY)

To read from and write to the display screen requires a pair of shared variables. One of the shared variable names must have the prefix CTL, and the other name must have the prefix DAT. After the prefix, the next 15 characters in each name must be identical, if specified. For example:

CTLDISPLAY
DATDISPLAY

The shared variable name with the prefix CTL is used to send control information to the I/O processor. Also, the I/O processor assigns the return code to this shared variable. The shared variable name with the prefix DAT is used to send data to or receive data from the display screen.

The control information assigned to the shared variable with the CTL prefix must be integers in a two- or three-element vector as follows:

CTL← $\genfrac{}{}{0pt}{}{0}{1}$ starting character number [number of characters]

Where:

- The first element specifies that data is to be read from (0) or written to (1) the display screen.

- The second element specifies the starting character number to read or write. The characters are numbered as follows:

Line	First character	Last character
15	0	63
14	64	127
13	128	191
12	192	255
11	256	319
10	320	383
9	384	447
8	448	511
7	512	575
6	576	639
5	640	703
4	704	767
3	768	831
2	832	895
1	896	959
0	960	1023

- The third element is optional. This element specifies the number of consecutive characters to read or write. If this element is not specified, only one character is assumed. If the specified number of characters plus the starting character number exceeds the value 1024 (the total number of characters that fit on the display screen), a 15 0 return code is assigned to the CTL shared variable.

Reading Records from the Display Screen

When the CTL shared variable specifies that characters are to be read, the cursor appears on the display screen at the character position specified by the starting record number. You can then modify the information on the display screen for the specified number of characters. For example, pressing ATTN erases any data from the cursor position to the last specified character. Now, when the EXECUTE key is pressed, the specified characters are read from the display screen and assigned to the DAT shared variable as a character vector.

For example, assume the first 3 characters on the display screen are A, B, and C:

```
CTLDISPLAY←0 0 3
```

Read 3 characters.

Start with the first character.

Once the previous statement is executed, the cursor (flashing character) is at the first character position on the display screen. Now, if you enter the 3 characters DEF and press EXECUTE:

```
      DATDISPLAY
DEF
      CTLDISPLAY
0 0
```

The specified characters currently on the display are read and assigned to the DAT shared variable.

The return code is assigned to the CTL shared variable.

Once the character vector read from the display screen is assigned to the DAT shared variable, the shared variable can be used like any other variable in the active workspace.

Note: When you are reading data from the display screen using a loop, you should program an escape from the loop in your user-defined function. For example:

```
LOOP:CTLDISPLAY←0 0 20
→(0≠1↑CTLDISPLAY)/0
→(∧/'OUT'=3↑DATDISPLAY)/OUT
X←X,[1]DATDISPLAY
→LOOP
OUT:
    •
    •
    •
    •
```

Branch out of the loop if the first 3 characters read are OUT.

This is necessary because you cannot interrupt reading data from the display screen using the ATTN key.

Writing Records to the Display Screen

When the CTL shared variable specifies that records are to be written, the I/O processor writes the current value of the DAT shared variable to the display screen. For example:

```
      DATDISPLAY←'NEW'
      CTLDISPLAY←1 0 3 ←—— Write the three records N, E, and W to the first
      CTLDISPLAY             three positions of the display screen.
0 0 ←——————————————————— The return code is assigned to the CTL shared
                             variable.
```

If the DAT shared variable contains more data than the CTL shared variable specifies to be written to the display screen, only the specified number of records are written to the display screen, starting with the first record of the character vector assigned to the DAT shared variable. If the DAT shared variable does not contain as many characters as specified by the CTL shared variable, the error messages IO STATUS: INVALID DAT and INTERRUPT are displayed (unless MSG=OFF is specified).

Sending Data to the Printer (PRT)

When a shared variable is used to specify that data is to be sent to the printer, data (character scalar or vectors) assigned to that shared variable is sent to the printer. Then the I/O processor assigns a return code to the shared variable. A 0 0 return code indicates that the data was printed successfully, and a nonzero return code indicates that the print operation failed. See *Return Codes* later in this chapter for a description of each return code.

Note: The)OUTSEL OFF command is automatically issued by the system during PRT operations. The)OUTSEL option returns to its previous setting after the PRT operation is terminated.

TERMINATING THE OPERATION

When you no longer need a shared variable (or pair of shared variables) for a specified operation, you can terminate the operation by:

- Assigning an empty vector to the shared variable
- Completing execution of a user-defined function in which the shared variable is made local to that function
- Using the ⎕EX function or)ERASE command
- Using the ⎕SVR function

Assigning an Empty Vector to the Shared Variable

For sequential access operations (IN, OUT, and ADD), the I/O processor terminates the operation when you assign an empty vector to the shared variable. For output operations, the file header is updated with information about the file (see *The)LIB Command* in Chapter 3). For input operations, the operation is also terminated when an end-of-file empty vector is read from the data file or an error occurs due to the device (and the proper return code is assigned to the shared variable).

CAUTION
For OUT, OUTF, and ADD operations, if the media is removed from the system before the operation is terminated, the data file will be unusable. Any subsequent attempt to read from the data file will give unpredictable results.

When creating a direct access data file (OUTF and ADD operations), the I/O processor terminates the operation and updates the file header with the EOD address when you assign an empty vector to the shared variable.

For direct access operations to read from and write to a data file or the display screen (IOR, IORH, INR, and DISPLAY), the I/O processor terminates the operation when you assign an empty vector to the CTL shared variable.

For printer operations (PRT), the I/O processor terminates the operation when you assign an empty vector to the shared variable.

The I/O processor assigns a return code to the shared variable after the operation is terminated. The 0 0 return code indicates that the operation was terminated successfully. See *Return Codes* later in this chapter for a description of each return code.

After an empty vector is used to terminate the operation, the shared variables can be used for specifying and performing other input/output operations.

Making the Shared Variable Local to a User-Defined Function

If a shared variable name is local to a user-defined function, the input/output operation is automatically terminated when the user-defined function completes execution. See *Assigning an Empty Vector to the Shared Variable* for a description of what happens when the operation is terminated. In this case, there is no return code assigned to the shared variable.

Using the ⎕EX Function or)ERASE Command

If the shared variable is expunged (⎕EX) or erased ()ERASED), the input/output operation is automatically terminated. See *Assigning an Empty Vector to the Shared Variable* for a description of what happens when the operation is terminated. In this case, there is no return code assigned to the shared variable.

Using the ⎕SVR Function

When you no longer need a shared variable for input/output operations, you should use the ⎕SVR system function to retract the shared variable offer. If the shared variable offer is retracted before you have terminated the input/output operation, the system will automatically terminate the operation. See *Assigning an Empty Vector to the Shared Variable* for a description of what happens when the operation is terminated. In this case, there is no return code to the share variable.

RETRACTING THE SHARED VARIABLE OFFER

To retract the offer to share a variable between the I/O processor and the active workspace, the ⎕SVR function is used. This function is monadic (requires one argument); the argument is character data that specifies the shared variable name(s) being retracted. For example:

```
⎕SVR'NAME'
```

If more than one name is used in the argument of the ⎕SVR function, the names must be in a character matrix with each row representing one name. For example:

```
⎕SVR 4 1⍴'ABCD'
```

or Each row represents a separate variable name.

```
NAMES←4 1⍴'ABCD'
⎕SVR NAMES
```

The explicit result of the ⎕SVR system function is a 2 or 0 for each shared variable name being retracted as a shared variable.

For example:

```
      ⎕SVR 4 1⍴'ABCD'
2 2 2 2
```

A 2 indicates that the name is successfully retracted as a shared variable with the I/O processor. A 0 indicates that the name is not successfully retracted as a shared variable. Normally, if a shared variable offer cannot be retracted successfully, it was never properly established as a shared variable.

Retracting the shared variable offer does not erase the value assigned to the variable. That is, the variable still has a value after the shared variable offer is retracted, until the variable is expunged or erased (⎕EX or)ERASE).

RETURN CODES

The I/O processor assigns a return code to the shared variable after each of the following operations:

- Specifying the operation to be performed
- Writing records to a sequential access file
- Writing records to and reading records from a direct access file
- Writing records to and reading records from the display screen
- Sending data to the printer
- Terminating the operation by assigning an empty vector to the shared variable
- Reading the last record of a sequential access file
- Reading a sequential access file when an input error occurs

The return code indicates whether or not the operation was successful. If the return is nonzero and MSG=OFF is not specified, in most cases an error message is also displayed. You can check the return code by referencing the shared variable once after the operation is complete. When the shared variable is referenced more than once to check a return code, the error messages IO STATUS: INVALID OPERATION and INTERRUPT are generated. Therefore, if you need to check the return code more than once, you must first assign the return code to another variable. For example:

```
      A←CTL
      A
0 0
      A
0 0
      CTL
IO STATUS:INVALID OPERATION
INTERRUPT
      CTL
      ^
```

In this example, CTL is a shared variable that has been assigned a 0 0 return code.

The shared variable cannot be referenced more than once for a return code.

Execution of a user-defined function does not stop when a nonzero return code is assigned to the shared variable. Therefore, the user-defined function should check the return code that was assigned to the shared variable to make sure that each operation is successful. For example:

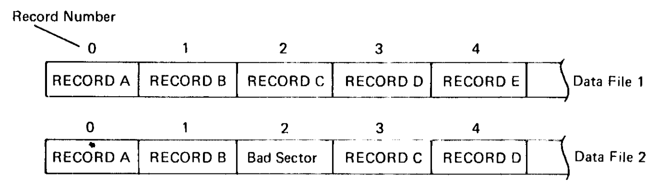
```
          :
[7]   DATX←'OUTF 11001 ID=(NAME) TYPE=U'
[8]   →(0≠1↑X←DATX)/ERROR
          :
[16]  ERROR:'SPECIFYING THE OPERATION FAILED--THE RETURN CODE IS:
         ',⍕X
[17]  →0
          :
```

Following is a description and/or user's response for each return code and error message:

Code	Error Message	Description and/or User's Response
0 N		Operation successful. When you are specifying direct access read and write operations (IOR, IORH, and INR), the second element is the number of records in the data file. When you are doing search by key operations, the second element is the number of the first record read. Otherwise, the second element is zero.
1 eee		Device error; the second element (eee) is the error code (see ERROR eee ddd in Appendix F).
2 0	IO STATUS: INVALID FILE	The specified file cannot be used for input/output operations.
3 0	IO STATUS: INVALID DEVICE or IO STATUS: INVALID DEVICE NUMBER	Specify the operation again, using the correct device number.
4 0	IO STATUS: INVALID FILE NUMBER	Specify the operation again, using a valid file number.
5 0	IO STATUS: NOT WITH OPEN FILE	The specified file, device, or corresponding DAT shared variable is already being used for input/output operations; the current operation must be terminated before a new operation can be specified.
6 0	IO STATUS: INVALID PARAMETER	The parameters required to specify the operation were entered incorrectly; specify the operation again, correcting any keying errors.
7 0	IO STATUS: WS FULL	There is not enough space in the active workspace for data items and buffers associated with the I/O operation. Use the)ERASE command or the ⎕EX function to erase any unwanted objects; then specify the operation to be performed again.
8 0	IO STATUS: DEVICE NOT OPEN	An attempt was made to use a shared variable, and an I/O operation has not been specified. Specify the I/O operation to be performed, or retract the shared variable if you want to use it as an ordinary variable.
9 0		An empty vector was read from the data file, but the empty vector is not the end-of-file empty vector. This return code is only a warning.

Code	Error Message	Description and/or User's Response
10 0	IO STATUS: EXCEEDED MAXIMUM RECORD LENGTH	The error was probably caused by the tape being removed before the file was closed. The remaining data in the file cannot be read.
11 0	IO STATUS: INVALID DATA TYPE	The wrong type or shape of data was used; for example, noncharacter data was sent to an exchange file, or noncharacter data was used as the parameters when you specified the operation to be performed.
12 0	IO STATUS: FILE NOT FOUND	The diskette files were searched by file name, but the file name was not found. Specify the operation again, using the correct file name, or insert the proper diskette.
13 0	IO STATUS: FILE NOT UNIQUE	The diskette files on all the diskette drives were searched by file name, and there is more than one file with that name. Specify the operation again, using the device/file number parameter.
14 0	IO STATUS: WRITE PROTECT	The data file is write-protected. If you want to write on the file, use the)PROTECT command to remove the write-protect indicator.
15 0	IO STATUS: CTL DOMAIN ERROR	Invalid control information was assigned to the CTL shared variable; for example, a record number greater than the number of records in the data file was specifed. *Note:* For a TYPE=M (file type 15), anytime you specify the number of records this return code is generated. Instead, you must let the 5110 use the default value (1).
16 0	IO STATUS: INVALID DAT	The data assigned to the DAT shared variable cannot be written to the data file. For example, the data has the wrong shape and/or representation for the data file.
18 0	IO STATUS: NO DAT	The DAT shared variable is not established as a shared variable before the CTL shared variable is used to specify a direct access input/output operation.

Code	Error Message	Description and/or User's Response
22 N		The operation is specified successfully, and the second element is the number of records in the file. However, the records were written to the file such that if the file contains a bad sector, that sector is skipped and the record is written to the next sector. If there are no bad sectors, the file can be read without any problems. If there are bad sectors, a 1 44 return code is issued if you try to access a bad sector. If the file contains a bad sector, you must determine the correct record location. For example:



Code	Error Message	Description and/or User's Response
		In data file 1, if you specify to read record number 2, RECORD C is read. In data file 2, if you specify that record number 2 should be read, a 1 44 return code is issued. Instead, you must specify record number 3 in order to read RECORD C.
23 0		The end of the file was encountered when the system was creating a direct access exchange, unblocked/unspanned, or internal format data file (type 9 or 10). The system automatically terminates the operation and updates the file header when this return code is issued.
24 0	10 STATUS: WRONG VOLUME	An attempt was made to read from or write to a diskette data file, and the diskette is not on the same diskette drive as when the operation was specified. Place the diskette containing the data file in the same diskette drive as when the input/output operation was specified, and try the operation again.

INPUT/OUTPUT SUMMARY

Following is a summary and examples of the steps required for input and output:

1. Make a shared variable offer and check the result.

```
X←1⎕SVO 2 4 ⍴'CTLXDATX'
→(∨/2≠X)/ERROR1
   ⋄
   ⋄
   ⋄
ERROR1:'THE SHARED VARIABLE OFFER FAILED.'
```

2. Specify the operation to be performed and check the return code.

Sequential Access:

```
X←'OUT 11001 ID=(SEQUENTIAL) MSG=OFF TYPE=I'
X←'ADD 11001 ID=(SEQUENTIAL) MSG=OFF'
X←'IN 11001 ID=(SEQUENTIAL) MSG=OFF'
```

Direct Access:

```
X←'OUTF 11001 ID=(DIRECT) MSG=OFF TYPE=U'
DATX←'ADD 11001 ID=(DIRECT) MSG=OFF'
CTLX←'INR 11001 ID=(DIRECT) MSG=OFF TYPE=N'
CTLX←'IOR 11001 ID=(DIRECT) MSG=OFF TYPE=N'
CTLX←'IORH 11001 ID=(DIRECT) MSG=OFF TYPE=N'
CTLX←'DISPLAY MSG=OFF'
```

Printer Output:

```
DATX←'PRT MSG=OFF'
```

Check Return Code:

```
→(0≠1↑DATX)/ERROR2
   ⋄
   ⋄
   ⋄
ERROR2:'SPECIFYING THE OPERATION FAILED.'
```

3. Perform the input and output operations, and check the return code after each operation.

`CTLX←0 0 5`	Direct access—Read five records, starting with the first record.
`CTLX←1 1 5`	Direct access—Write five records, starting with the second record.
`CTLX←2 10 5`	Direct access—Search by key five records, starting at the sector that contains the eleventh record.

Check Return Code:

```
→(0≠1↑SV←CTLX)/ERROR3
   ⋄
   ⋄
   ⋄
ERROR3:'I/O OPERATION FAILED, THE RETURN CODE IS:  ',⍕SV
```

4. Terminate the operation and check the return code.

```
CTLX←⍳0
→(0≠1↑CTLX)/ERROR4
   ⋄
   ⋄
   ⋄
ERROR4:'TERMINATING THE OPERATION FAILED.
```

5. Retract the shared variable offer and check the results.

```
X←⎕SVR 2 4⍴'CTLXDATX'
→(∨/2≠X)/ERROR5
   ⋄
   ⋄
   ⋄
ERROR5:'RETRACTING THE SHARED VARIABLE OFFER FAILED.
```

Note: Even after the shared variable offer is retracted, the variables still have a value.

Chapter 8. The 5103 Printer

The IBM 5103 Printer is available as a feature attachment and has these characteristics:

- Bidirectional printing (left to right, then right to left). The 5103 bidirectional printing operates as follows:

  The print head moves from the left margin and prints a line. Succeeding lines are printed in either direction depending on which end of the new line is closest to the current position of the print head. The print head is returned to the left margin periodically when printing is not imminent.

- 132 characters across the print line.

  Note: If the width of the forms is less than 132 characters and the ⎕PW system variable (see Chapter 5) is greater than the width of the forms, loss of data will occur as the print head leaves the form.

- Capability of using individual or continuous forms. Maximum number of copies is six, but for optimum feeding and stacking, IBM recommends a maximum of four parts per form.

- Adjustable forms tractor that allows the use of various width forms. The forms can be from 3 to 14.5 inches (76.2 to 368.3 millimeters) wide for individual forms and from 3 to 15 inches (76.2 to 381 millimeters) wide for continuous forms.

- Print position spacing of 10 characters per inch (25.4 millimeters) and line spacing of six lines per inch (25.4 millimeters).

- Stapled forms or continuous card stock cannot be used.

- The character printing rate is 80 or 120 characters per second. The throughput in lines per minute is function-dependent.

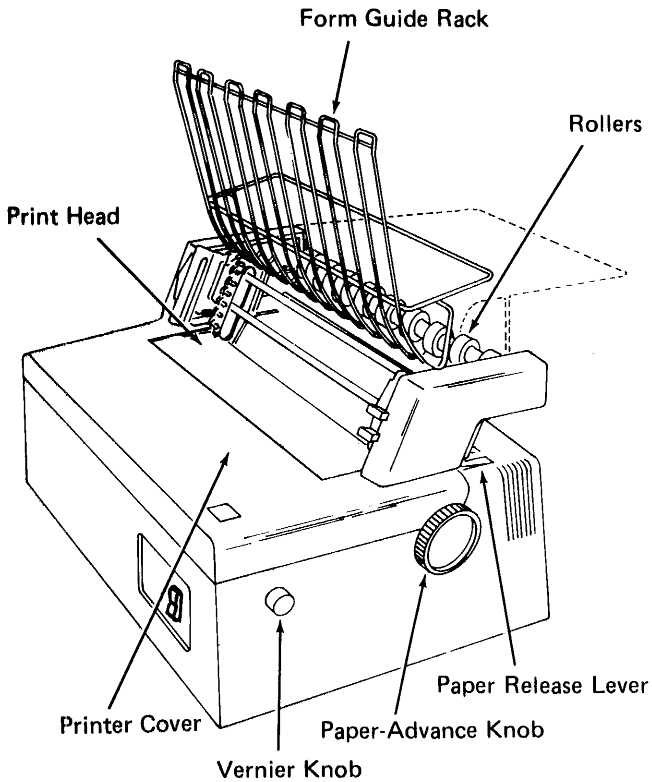
- A vernier knob (located on the right side of the printer) that allows for fine adjustment of the printing position. This knob should only be used when the print head is in its leftmost position.

- The forms must meet the following requirements:

	Singlepart Cut Forms	Multipart Cut Forms	Continuous Forms
Maximum Thickness	0.0075 inch (0.019 cm)	0.018 inch (0.0457 cm)	0.018 inch (0.0457 cm)
Minimum Thickness	0.003 inch (0.0076 cm)	0.003 inch (0.0076 cm)	0.003 inch (0.0076 cm)
Maximum Width	14.5 inches (36.83 cm)	14.5 inches (36.83 cm)	14.87 inches (37.76 cm)
Minimum Width	6 inches (15.24 cm)	6 inches (15.24 cm)	3 inches (7.62 cm)
Maximum length	14 inches (35.56 cm)	14 inches (35.56 cm)	
Minimum length	3 inches (7.62 cm)	3 inches (7.62 cm)	
Maximum Copies		4	4
Maximum Form Weight			15 lb ream (6804 kg)
Maximum Distance Between Folds			14 inches (35.56 cm)

HOW TO INSERT FORMS

Continuous Forms



1. Slide the top cover forward.

2. Push the print head to the extreme left position.

3. For singlepart forms, pivot the form guide rack up and forward to a vertical position. For multipart forms, leave the forms guide rack in the horizontal position.

   The diagrams below and to the left show the proper forms path for singlepart and multipart forms.

4. Push the paper release lever to the rear to activate the friction feed rolls.

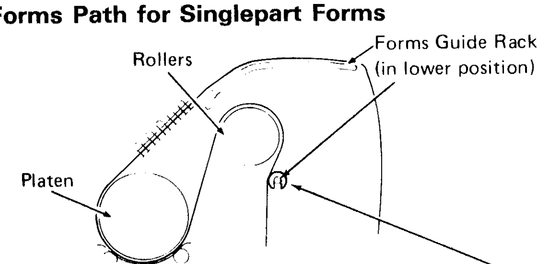
5. Place the forms on the table behind the printer.

   Note: The forms must be positioned behind the printer so that the forms feed squarely into the printer.

6. Thread the paper down, over the rollers, behind the tractors, and behind the platen.

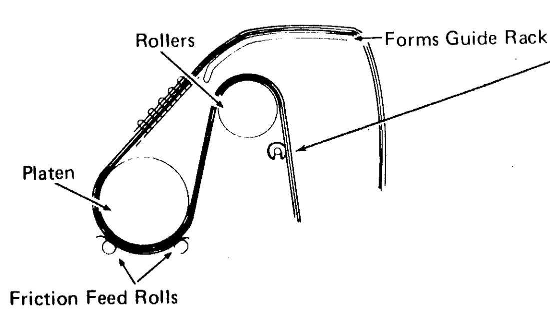
7. Turn the paper-advance knob to move the paper around the platen until you can grasp it with your fingers.

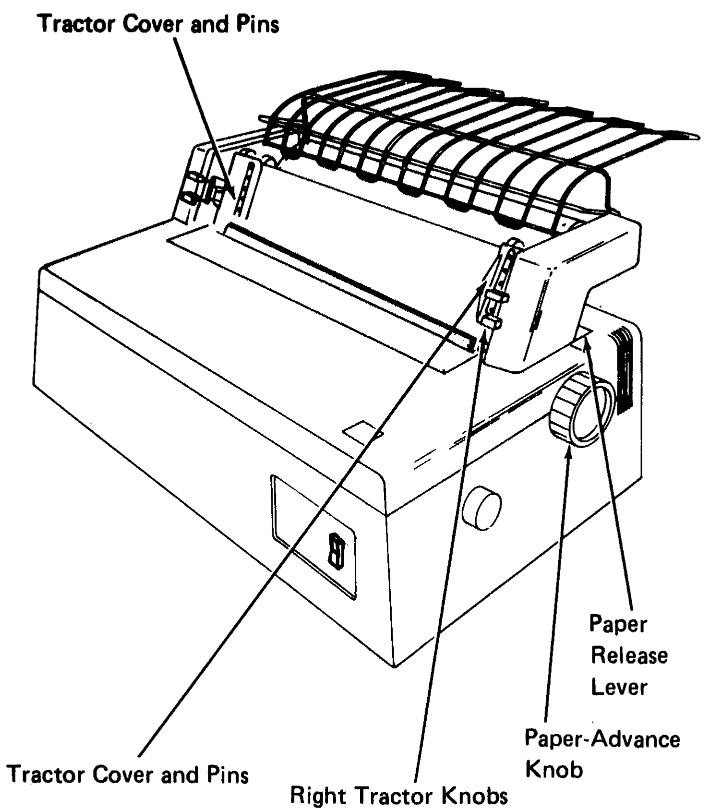
Forms Path for Singlepart Forms



Position these guides at the edge of the paper.

Forms Path for Multipart Forms





8. Open both tractor covers.

9. Pull the paper release lever forward to disengage the friction feed rolls.

10. Pull the paper up and place the left margin holes over the tractor pins. Be sure the left tractor is in its leftmost position.

11. Close the left tractor cover.

12. Squeeze the two knobs on the right tractor, and slide the tractor to align the pins with the right margin holes.

13. Place the right margin holes over the tractor pins.

14. Close the right tractor cover.

15. For singlepart forms, pivot the forms guide rack to a horizontal position.

16. Turn the paper-advance knob to position the form for the first line to be printed. The paper should exit over the forms guide rack.

    Note: To move the form backward, turn either paper-advance knob backward and pull the form from behind the printer to keep the form from buckling at the print head.

17. Close the top cover.

18. The plastic guides on the rear of the wire rack should be positioned (one on each side of the forms) so as to aid in guiding the forms for proper feeding. These guides are positioned by sliding them back and forth.

CAUTION
The switch that senses end of forms is deactivated when the friction feed rolls are engaged. Thus, the print wires could hit the base platen if no forms are in the printer.

Cut Forms

1. Remove the forms tractor by tilting it back and lifting it off.

2. Move the cut forms guide forward.

3. Slide the top cover forward.

4. Push the print head to the extreme left position.

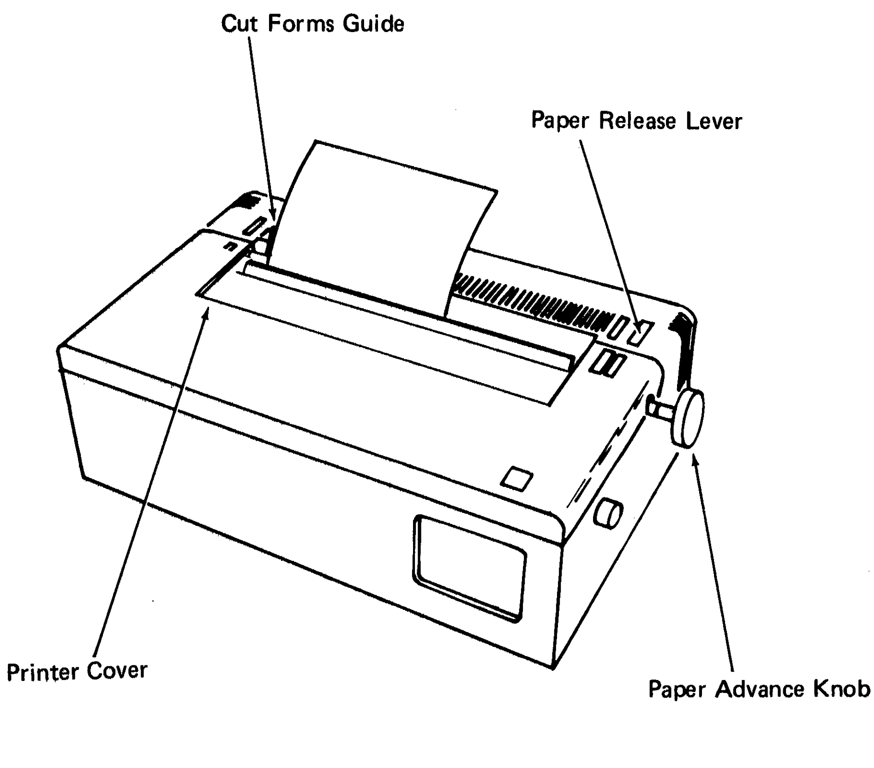
5. Push the paper release lever to the rear to activate the friction feed rolls.

6. Place the form in position behind the platen and against the cut forms guide.

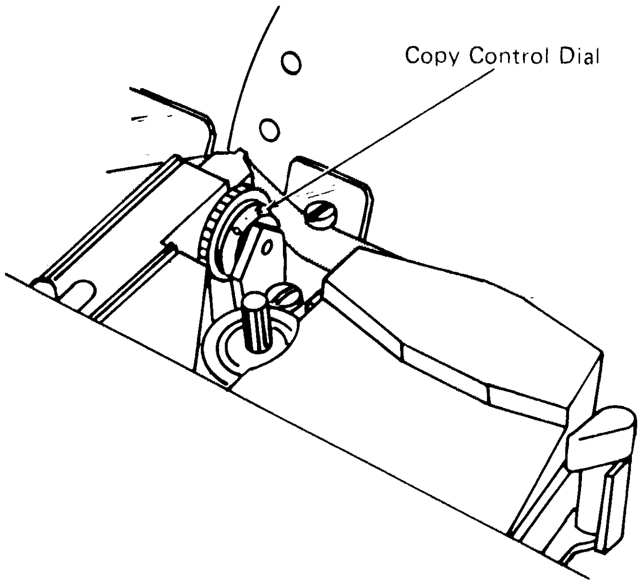
7. Turn the paper-advance knob to position the form for the first line to be printed. Improve the paper alignment if necessary by using the paper release lever.

8. Close the top cover.

CAUTION
The switch that senses end of forms is deactivated when the friction feed rolls are engaged. Thus, the print wires could hit the base platen if no forms are in the printer.

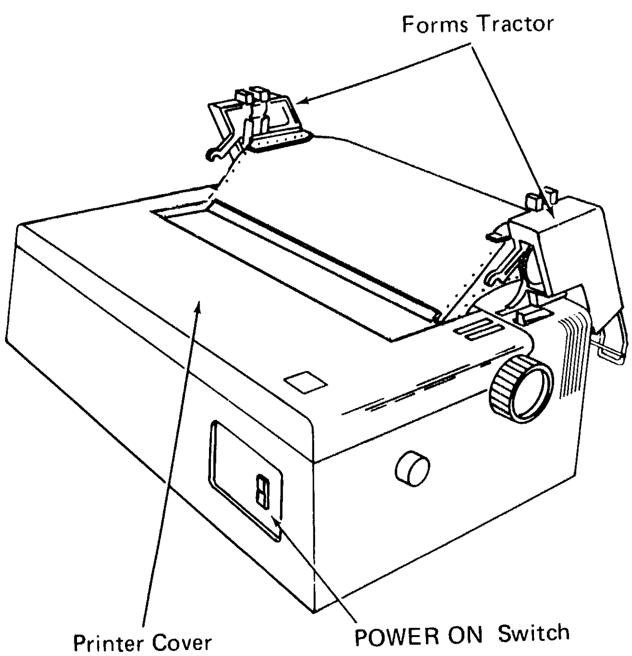


HOW TO ADJUST THE COPY CONTROL DIAL FOR FORMS THICKNESS

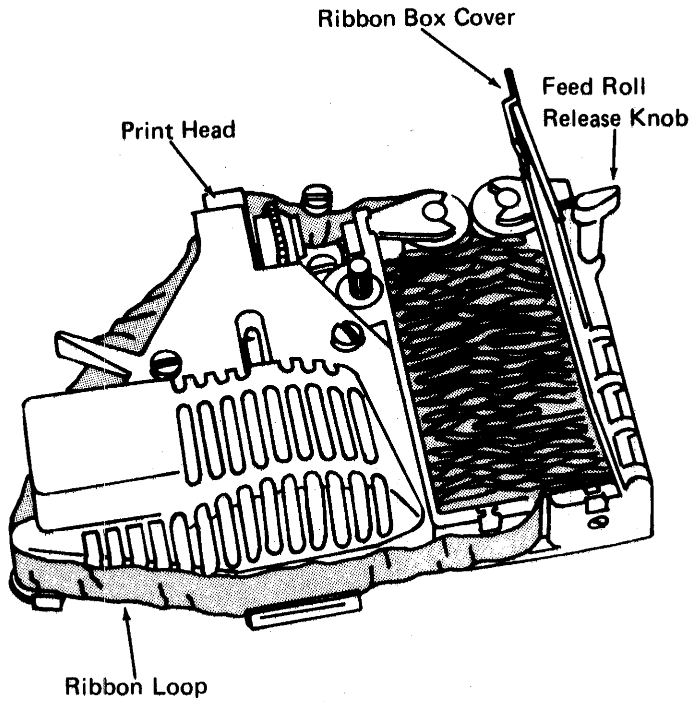


1. If you are using singlepart forms, set the copy control dial on 0.
2. If you are using multipart forms and the last sheet is not legible, rotate the copy control dial toward 0 one click at a time to obtain the legibility you desire.
3. If you are using multipart forms and the ribbon is smudging the first sheet, rotate the copy control dial toward 8 one click at a time until smudging stops.

HOW TO REPLACE A RIBBON (PART NUMBER 1136653)



1. Turn off power to the printer.
2. Tilt the forms tractor back by lifting both sides at the front.
3. Slide the top cover forward, then lift the front edge of the top cover and remove it.



4. Be sure that the print head is to the extreme left.

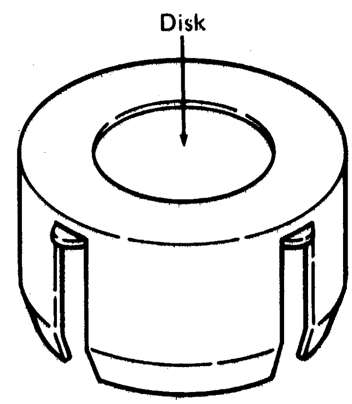
5. Turn the feed roll release knob counterclockwise until it points to the right.

6. Open the ribbon box cover.

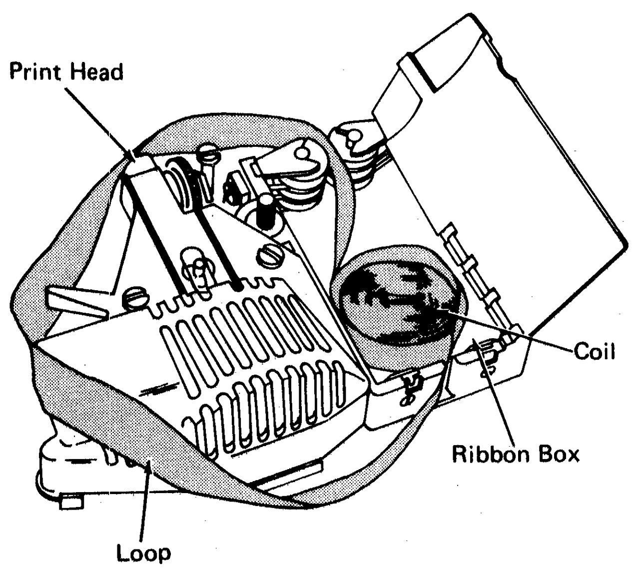
7. Put on the gloves supplied with the new ribbon.

8. Remove the old ribbon from the guides being careful to disengage it from the clip on the print head.

9. Lay the ribbon loop on the top of the ribbon in the ribbon box. Pick up the entire ribbon and discard it.



Ribbon Holder

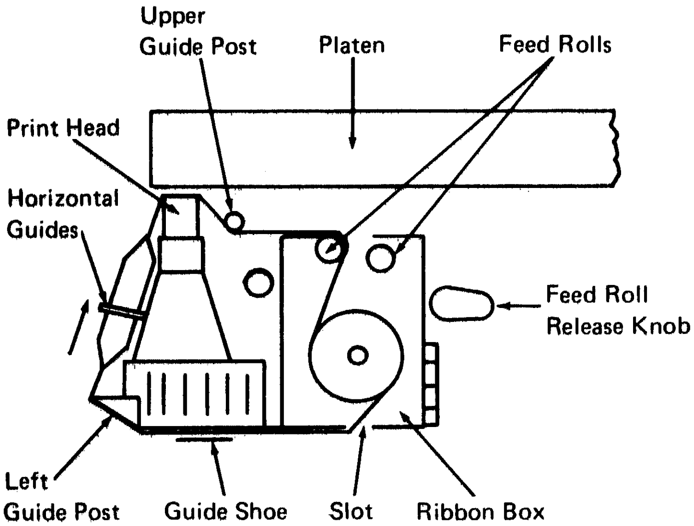


10. Eject the new ribbon from its holder into the ribbon box by pressing on the disk.

11. Remove the disk from the ribbon and discard the disk and the holder.

12. Hold the coil lightly with one hand and pull about 10 inches (254 mm) of ribbon from the coil.

13. Form a loop from the ribbon across the print head.



14. Thread the part of the loop nearest the platen between the feed rolls and on the inside of the upper guide post.

15. Turn the feed roll release knob clockwise to close the feed rolls.

16. Thread the ribbon between the print head and the platen. Be sure the ribbon is under the clip on the print head.

17. Thread the other part of the loop through the slot in the bottom of the ribbon box.

18. Thread the ribbon through the guide shoe and around the left guide post.

19. Insert the horizontal part of the ribbon twist (bottom edge first) between the two horizontal guides.

20. Move the print head back and forth across the platen to remove the slack from the ribbon. Continue moving the print head until you are sure that the ribbon feeds properly. Leave the print head at the extreme left.

21. Close the ribbon box cover.

22. Close the printer cover and turn the power on.

23. Reposition the form tractor.

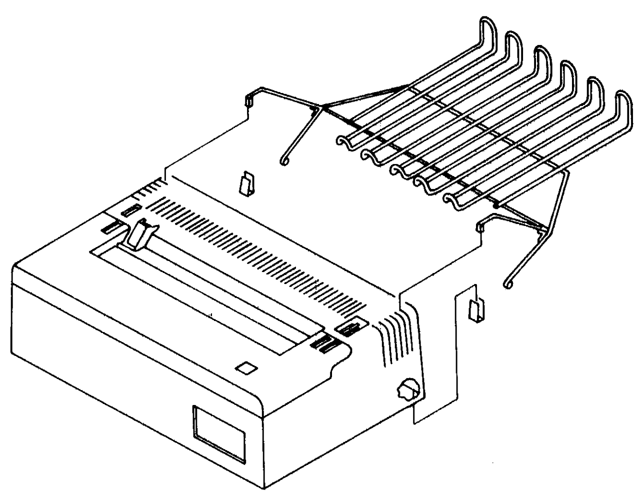
Note: When the printer is left unattended (such as during lunch or overnight), make sure the print head is to the extreme left. This prevents the ribbon ink from bleeding onto the paper.

HOW TO INSTALL THE 5103 PRINTER STACKER

A folded-form paper stacker is supplied with 5103 printers. The wire stacker hooks onto the back of the printer cover as shown in the drawing. The lower wires on the stacker should contact the metal clips on the cover.

The stacker can be bent if too much weight is applied. Under normal conditions, printed forms should not be allowed to accumulate higher than 1 inch in the stacker.

Note that, because of the relatively small free-fall distance of the paper as it leaves the printer, you may have to manually fold the first two or three sheets to get the folding operation started.



Appendix A. 5110 APL Character Set and Overstruck Characters

To form overstruck characters, enter one character, backspace and enter the other character or use the applicable CMD/key combinations. The 5110 APL character set consists of all the characters represented on the 5110 keyboard plus the following overstruck characters:

Function	Character	Keys Used
Comment	⍝	∩ C, ∘ J
Execute	⍎	⊥ B, ∘ J
Factorial, combination	!	' K, : .
Format	⍕	⊤ N, ∘ J
Grade down	⍒	∇ G, | M
Grade up	⍋	Δ H, | M
Logarithm	⍟	* P, ○ O
Matrix division	⌹	⎕ L, ÷ X
Nand	⍲	∧ 0, ~ T
Nor	⍱	∨ 9, ~ T
Protected function	⍫	∇ G, ~ T
Quad quote	⍞	⎕ L, ' K
Rotate, reverse	⌽	| M, ○ O
Transpose	⍉	\ /, ○ O
Compress	⌿ (see note)	\ /, - +
Expand	⍀ (see note)	\ /, - +
Rotate, reverse	⊖ (see note)	○ O, - +

Note: These are variations of the symbols for these functions; they are used when the function is acting on the first coordinate of an array.

Appendix B. Atomic Vector

CHARACTER	CHARACTER NAME	INDEX (⎕IO←1)
	RESERVED	1
	RESERVED	2
	RESERVED	3
	RESERVED	4
	RESERVED	5
	RESERVED	6
	RESERVED	7
	RESERVED	8
	RESERVED	9
	RESERVED	10
	RESERVED	11
	RESERVED	12
	RESERVED	13
	RESERVED	14
[	OPEN BRACKET	15
]	CLOSE BRACKET	16
(	OPEN PARENTHESIS	17
)	CLOSE PARENTHESIS	18
;	SEMICOLON	19
/	SLASH	20
\	SLOPE	21
←	LEFT ARROW	22
→	RIGHT ARROW	23
	RESERVED	24
	RESERVED	25
¨	DIERESIS (UPPERSHIFT 1)	26
+	PLUS	27
-	BAR	28
×	TIMES	29
÷	DIVIDE	30
*	STAR	31
⌈	UPSTILE	32
⌊	DOWNSTILE	33
|	STILE	34
∧	AND	35
∨	OR	36
<	LESS THAN	37
≤	NOT GREATER	38
=	EQUAL	39
≥	NOT LESS	40
>	GREATER THAN	41
≠	NOT EQUAL	42
⍺	ALPHA	43
∊	EPSILON	44
⍳	IOTA	45
⍴	RHO	46
⍵	OMEGA	47

,	COMMA	48
!	QUOTE DOT (EXCLAMATION)	49
⌽	CIRCLE STILE.	50
⊥	BASE.	51
⊤	TOP	52
○	CIRCLE.	53
?	QUERY	54
~	TILDE	55
↑	UP ARROW.	56
↓	DOWN ARROW.	57
⊂	OPEN SHOE	58
⊃	CLOSE SHOE.	59
∩	CAP	60
∪	CUP	61
_	UNDERSCORE.	62
⊂∘	CIRCLE SHOE	63
⌶	⌶-BEAM.	64
∘	NULL (SMALL CIRCLE)	65
⎕	QUAD.	66
⍞	QUOTE QUAD.	67
⍟	LOG	68
⍲	NAND.	69
⍱	NOR	70
⍝	LAMP-COMMENT.	71
⍋	DELTA STILE	72
⍒	DEL STILE	73
⊖	CIRCLE BAR.	74
⌿	SLASH BAR	75
⍀	SLOPE BAR	76
⌹	DOMINO.	77
⍕	TOP NUL	78
⍎	BASE NUL.	79
&	AMPERSAND	80
@	AT.	81
#	POUND	82
$	DOLLAR.	83
	UNUSED.	84
TΔ	TRACE (T DELTA)	85
SΔ	STOP (S DELTA).	86
A	A	87
B	B	88
C	C	89
D	D	90
E	E	91
F	F	92
G	G	93
H	H	94
I	I	95
J	J	96
K	K	97
L	L	98
M	M	99
N	N	100
O	O	101
P	P	102

Q	Q	103
R	R	104
S	S	105
T	T	106
U	U	107
V	V	108
W	W	109
X	X	110
Y	Y	111
Z	Z	112
Δ	DELTA	113
A̲	A-UNDERSCORE	114
B̲	B-UNDERSCORE	115
C̲	C-UNDERSCORE	116
D̲	D-UNDERSCORE	117
E̲	E-UNDERSCORE	118
F̲	F-UNDERSCORE	119
G̲	G-UNDERSCORE	120
H̲	H-UNDERSCORE	121
I̲	I-UNDERSCORE	122
J̲	J-UNDERSCORE	123
K̲	K-UNDERSCORE	124
L̲	L-UNDERSCORE	125
M̲	M-UNDERSCORE	126
N̲	N-UNDERSCORE	127
O̲	O-UNDERSCORE	128
P̲	P-UNDERSCORE	129
Q̲	Q-UNDERSCORE	130
R̲	R-UNDERSCORE	131
S̲	S-UNDERSCORE	132
T̲	T-UNDERSCORE	133
U̲	U-UNDERSCORE	134
V̲	V-UNDERSCORE	135
W̲	W-UNDERSCORE	136
X̲	X-UNDERSCORE	137
Y̲	Y-UNDERSCORE	138
Z̲	Z-UNDERSCORE	139
Δ̲	DELTA-UNDERSCORE	140
0	0	141
1	1	142
2	2	143
3	3	144
4	4	145
5	5	146
6	6	147
7	7	148
8	8	149
9	9	150
.	DOT	151
¯	OVERBAR	152
	SPACE	153
'	QUOTE	154
:	COLON	155
∇	DEL (FN DEF CHAR)	156

	CURSOR RETURN	157
	END OF BLOCK(CANNOT BE DISPLAYED)	158
	BACKSPACE	159
	LINEFEED.	160
✧	DEL TILDE	161
	. . . .	162
	. . . .	163
	. . . .	164
	. . . .	165
	. . . .	166
	. . . .	167
	. . . .	168
	LENGTH OF Z-SYMBOL TABLE.	169
𝕎	O-U-T FOR COMMUNICATION TAPE. . .	170
¬	LOGICAL NOT	171
"	DOUBLE QUOTE.	172
%	PERCENT	173
	. . . .	174
	. . . .	175
	. . . .	176
	. . . .	177
	. . . .	178
	. . . .	179
	. . . .	180
	. . . .	181
	. . . .	182
	. . . .	183
¢	CENT.	184
	. . . .	185
	. . . .	186
a	a	187
b	b	188
c	c	189
d	d	190
e	e	191
f	f	192
g	g	193
h	h	194
i	i	195
j	j	196
k	k	197
l	l	198
m	m	199
n	n	200
o	o	201
p	p	202
q	q	203
r	r	204
s	s	205
t	t	206
u	u	207
v	v	208

⍵	⍵	209
x	x	210
y	y	211
z	z	212
{	LEFT BRACE.	213
}	RIGHT BRACE	214
ʃ	HOOK.	215
Ψ	FORK.	216
⑁	CHAIR	217
|	LONG VERTICAL LINE.	218
⦚	LOGICAL OR.	219
¦	VERTICAL BAR.	220
`	GRAVE ACCENT.	221
&̲	. . . .	222
π̲	Note:These unique graphics are. .	223
⊢	for maintenance use only. They. .	224
⊣	are incompatible with other . . .	225
⋄	systems and cannot be used for. .	226
Σ	exchange purposes. These graphics	227
∫	may be removed or changed as a. .	228
´	result of maintenance or new. . .	229
±	versions of this product.	230
∂	. . . .	231
	. . . .	232
	. . . .	233
	. . . .	234
	. . . .	235
	. . . .	236
	. . . .	237
	. . . .	238
	. . . .	239
	. . . .	240
	. . . .	241
!	ALTERNATE SHRIEK.	242
	. . . .	243
	. . . .	244
	. . . .	245
	. . . .	246
	. . . .	247
~	ALTERNATE TILDE	248
	. . . .	249
	. . . .	250
\	ALTERNATE SLOPE	251
	. . . .	252
	. . . .	253
	. . . .	254
	. . . .	255

Note: The characters | (⎕AV[218]), ⦚ (⎕AV[219]), ! (⎕AV[242]), ~ (⎕AV[248]), and \ (⎕AV[251]) are similar in appearance to the symbols for the APL built-in functions. However, if these characters are used as APL built-in functions, a ENTRY ERROR is displayed.

Appendix C. EBCDIC Character Set and Special National Graphics

The system translates the APL internal code into EBCDIC (extended binary coded decimal interchange code) when data is written to tape or diskette. The following chart shows the character that is displayed or printed for each EBCDIC hexadecimal value:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4		A̲	B̲	C̲	D̲	E̲	F̲	G̲	H̲	I̲	¢	.	<	(	+	⌶
5	&	J̲	K̲	L̲	M̲	N̲	O̲	P̲	Q̲	R̲	!	$	*	)	;	¬
6	-	/	S̲	T̲	U̲	V̲	W̲	X̲	Y̲	Z̲	¦	,	%	_	>	?
7	&̲	∧	¨	⍙		⍫			∨	`	:	#	@	'	=	"
8	~	a	b	c	d	e	f	g	h	i	↑	↓	≤	⌈	⌊	→
9	⎕	j	k	l	m	n	o	p	q	r	⊃	⊂		○		←
A	¯	~	s	t	u	v	w	x	y	z	∩	∪	⊥	[	≥	∘
B	α	∊	⍳	⍴	⍵		×	\	÷		∇	∆	⊤	]	≠	∣
C	{	A	B	C	D	E	F	G	H	I	⍲	⍱	⌷	⌽	⍋	⍉
D	}	J	K	L	M	N	O	P	Q	R	⌹	!	⍕	⍎	⍞	⍝
E	\		S	T	U	V	W	X	Y	Z	⌿	⍀	⊣	⊖	⌸	⍑
F	0	1	2	3	4	5	6	7	8	9	|	⍒		⍟	⍏	

For example, The EBCDIC character 0 has a hex value F0.

In the previous chart, graphics are assigned to all blank positions (except hex 40) for maintenance use only. These graphics are incompatible with other systems and cannot be used for exchange purposes.

Some of the graphics in the previous chart change when a national character set, other than EBCDIC, is specified. You can specify other national character sets by:

- Using the ⎕CC system function (see Chapter 5)
- Pressing the HOLD key, and then holding down the shift key and pressing:

1 to select Austria/Germany	9 to select Portugal
2 to select Belgium	0 to select International
3 to select Brazil	÷ to select Spain
4 to select Denmark/Norway	× to select Spanish Speaking
5 to select Finland/Sweden	- to select United Kingdom
6 to select France	+ to select French Canadian
7 to select Italy	. to select EBCDIC
8 to select Japan	

- Having your service representative change the character set that is in effect when the power is turned on or RESTART is pressed. In this case, you can still select any of the other character sets from the keyboard or a user-defined function.

The following chart shows the hexadecimal value and associated national graphics for each unique national character set:

Country	Hex Position											
	4A	5A	5B	5F	6A	79	7B	7C	A1	C0	D0	E0
Austria/Germany	Ä	Ü	$	^	ö	`	#	§	ß	ä	ü	ö
Belgium	[	]	$	^	ù	`	#	à	¨	é	è	ç
Brazil	É	$	Ç	^	ç	ã	Õ	Ã	~	õ	é	\
Denmark/Norway	#	¤	Å	^	ø	`	Æ	Ø	ü	æ	å	\
Finland/Sweden	§	¤	Å	^	ö	é	Ä	Ö	ü	ä	å	É
France	°	§	$	^	ù	`	£	à	¨	é	è	ç
Italy	°	é	$	^	ò	ù	£	§	ì	à	è	ç
Japan	£	!	¥	¬	¦	`	#	@	‾	{	}	$
Portugal	[	]	$	^	õ	`	Ã	Õ	ç	ã	´	Ç
International	[	]	$	^	¦	`	#	@	~	{	}	\
Spain	[	]	₧	¬	ñ	`	Ñ	@	¨	{	}	\
Spanish Speaking	[	]	$	¬	ñ	`	Ñ	@	¨	{	}	\
United Kingdom	$	!	£	¬	¦	`	#	@	‾	{	}	\
French Canadian	à	´	$	^	ù	`	#	@	¨	é	è	¸
EBCDIC	¢	!	$	¬	¦	`	#	@	~	{	}	\

Appendix D. 5110 Compatibility with the IBM 5100

The 5110 APL system differs from the 5100 APL system as follows:

- The following system commands have been added to support the diskette:

)FILEID	)PROTECT
)FREE	)VOLID

- The system commands)PROC,)LINK,)RESUME, and)SORT have been added to enhance the system.

- The following system functions and variables have been added to maintain syntactic compatibility between the 5110 and APLSV:

	Default Value
⎕AI	0 0 0 0
⎕DL	0
⎕TS	1900 0 0 0 0 0 0
⎕TT	0
⎕UL	1

- The system variable ⎕CC has been added to enhance the system.

- For system commands that have device/file number and workspace ID parameters, only one or the other parameter needs to be specified for diskette interface.

- When shared variables are used, surrogate names can be specified and used.

- Overlapped printing (one line only) occurs as the 5110 processes data.

- The EBCDIC character set is used for diskette files.

- Lowercase characters can be entered from the keyboard.

- The ⎕PW value is used during ∇FUNCTION[⎕]∇.

- Single or multiple records on diskette direct access data files can be read or written when you specify the record number using a shared variable.

- Records on the display screen can be read or written when you specify the record number using a shared variable.

- A strong interrupt is caused when you hold down the SHIFT key and press the ATTN key.

- References to a shared variable that has not been used to specify an operation to be performed causes an INTERRUPT error message.

- If the format function is used with a shared variable during input operations, alternate records are *not* skipped if the input data is already in character form.

- File types 2, 7, and 8 can be written on tape by the 5100 and read by the 5110, and vice versa.

- Only workspaces written to tape with the)SAVE command by the 5100 can be loaded or copied into the 5110 active workspace. 5100 continued workspaces (file type 6) and 5110 continued workspaces (file type 26) are *not* compatible.

- The 5110 requires 224 bytes more overhead from the user work area than did the 5100. Therefore, some programs that executed on the 5100 might generate a WS FULL error on the 5110.

- Programs written for 5100 shared variable operations that permit you to specify the device/file number and then use the default file name DATA will *not* work for 5110 diskette files. The program must be changed so that you must also specify a simple or complex file name.

Appendix E. 5110 APL Compatibility with IBM APLSV

The 5110 APL system differs from the IBM APLSV system primarily because the 5110 is a single user system with different input/output devices and it has display screen output rather than typewriter output. The differences are as follows:

- Turning power on signs the user on; therefore, no sign-on or ID number is required.

- The 5110 active workspace is generally smaller than APLSV active workspace. It is further limited by the I/O processor and shared variable processor, which use it for input/output buffers and work areas.

- The default number of symbols is 125 instead of 256, which increases the available workspace for most users.

- The library number that appears in system commands has been redefined to a device/file number. It is a one- to five-digit number that specifies the device and file number where a workspace is to be saved or loaded. If the number is less than four digits, it is only the file number; default device is assumed; otherwise, the high-order one or two digits is the device number.

- The following commands are not supported because they apply only to multiterminal systems and remote systems:

)OFF;)OFF HOLD;)CONTINUE HOLD;)PORTS;)MSGN;)MSG;)OPRN;)OPR; all special system operator commands

- The following commands are not supported because the function is not supported:

)GROUP;)GRPS;)GRP

- The following commands are not supported:

)ORIGIN;)WIDTH;)DIGITS

  They are available with the system variables ⎕IO, ⎕PW, and ⎕PP, respectively.

- The following commands have been added to support the 5110 processor and its input/output devices:

)MARK	– To format files
)OUTSEL	– To specify which transactions are to be printed
)REWIND	– To rewind the tape unit
)RESUME	– To load a stored workspace and reestablish the system environment
)FILEID	– To change the file name on a diskette volume
)FREE	– To make a diskette file space available for reallocation
)PROTECT	– To change the write-protect indicator
)VOLID	– To change the volume ID or access protect indicator
)LINK	– To load microcoded programs
)PROC	– To open a procedure file
)SORT	– To transfer control to the sort utility

- The)CONTINUE command has been changed to save workspaces with suspended functions. The parameters are the same as)SAVE but the stored workspace cannot be copied or loaded into a 5110 with a smaller active workspace.

- Since the 5110 system is not in a communications environment, the RESEND message will not occur.

- 5110 is implemented with only one workspace area (no spare); therefore, the following error messages have been added:

  1. Function name [statement number] LINE TOO LONG – Cannot save functions with statements greater than 115 characters.

  2. WS TOO BIG – Workspace is too big to fit in the active workspace.

  3. NOT WITH SUSPENDED FUNCTION – Only the)CONTINUE command will work to write the workspace to tape.

- For diagnostic reasons, occurrence of SYSTEM ERROR does not clear the workspace. The following message occurs when you attempt anything other than)CLEAR after a system error:

  NOT WITH SYSTEM ERROR

- Saved workspaces are not time-stamped and dated because the information is not available in this system; therefore, the following messages now occur after library operations:

COPIED	device/file	wsid
LOADED	device/file	wsid
SAVED	device/file	wsid
CONTINUED	device/file	wsid
DROPPED	device/file	wsid

- The)LIB command does more than list the saved workspaces. It lists all the files on the specified device. The response, therefore, contains more information (see)LIB command is Chapter 3).

- The following system messages have been added for the new system commands and input/output operations:

ALREADY MARKED
DEVICE NOT OPEN
ERROR eee d
EXCEEDED MAXIMUM RECORD LENGTH
INVALID DATA TYPE
INVALID DEVICE
INVALID DEVICE NUMBER
INVALID FILE
INVALID FILE NUMBER
INVALID OPERATION
INVALID PARAMETER
MARKED b n
NOT WITH OPEN FILE
NOT WITH OPEN DEVICE
FILE NOT FOUND
FILE NOT UNIQUE
WRITE PROTECT
CTL DOMAIN ERROR
INVALID DAT
NO DAT
WRONG VOLUME
INVALID DISK FILE NAME
LINE TOO LONG

- The shared variable processor on the 5110 is designed to provide an interface between only one APL user and one I/O processor. Thus, only one processor number is supported (1).

  The response to ⎕SVO is 2, since, if it is a valid share, it is always accepted before the APL user regains control. (If an unsupported processor is specified, the response is 1.)

  The response to ⎕SVR is the same as the response to ⎕SVO.

  Being strictly a sequential machine, the only mode of interaction is reversing half-duplex; that is, the I/O processor always responds to each action by the APL user. Therefore, the access control vector (⎕SVC) is always 1 1 1 1 for sequential operations and the CTL shared variable for direct access operation.

  Because there are never any outstanding offers, ⎕SVQ always returns an empty vector.

- This is a single user system without an internal clock; therefore, the following system variables and functions are supported as follows:

	Default Value
⎕TS – Time stamp	1900 0 0 0 0 0 0
⎕AI – Accounting information	0 0 0 0
⎕TT – Terminal type	
⎕UL – User list	1
⎕DL – Delay	0

- The I-beam functions have been replaced with system variables or system functions and are not supported.

- Catenation using semicolons had been replaced for format, but it is still supported on the 5110.

- Data can be exchanged between APL and BASIC or other systems via communications; therefore, the following characters have been added to the APL character set:

  $, #, @, &, ¬, %, "

- The display screen is 64 characters wide; therefore, the initial values of ⎕PW and ⎕PP system variables are 64 and 5 instead of 120 and 10.

  If the print width is altered to something greater than 64, any output that exceeds 64 characters is wrapped to another line on the display screen.

- Bare (⍞) output followed by bare (⍞) input yields a different reply. For APLSV, the ⍞ input is prefixed by the same number of blanks as the previous ⍞ output. For 5110 APL, the ⍞ input is prefixed by the previous ⍞ output. (See Chapter 6 for more information on bare output followed by bare input.)

- The display screen provides the ability to edit lines of data directly; therefore, the following changes were made to function definition:

[N⎕]	– Now displays line N in the display screen lines 1 and 0 for editing.
[N⎕M]	– Has the same result as [N⎕]; the M is erased when the EXECUTE key is pressed.
[∆N]	– Allows line N to be deleted. N must be a single line number.

  The use of the ATTN key to delete a line works, but only in function definition mode, not while entering function definition mode.

  To prevent problems when displaying or editing statements in a user-defined function, the print width (⎕PW) is automatically set to 128 when the 5110 is in function definition mode. The print width automatically returns to its previous setting when the function definition is closed.

  There is only limited editing space; therefore, function statements that are greater than 115 characters cannot be edited, and the message LINE TOO LONG is displayed.

- The 5110 will insert a quote if an uneven number of quotes is entered.

Appendix F. Error Messages

Error messages can result when you incorrectly use APL primitive (built-in) functions, user-defined functions, system commands, system variables, or input/output operations. The following list contains the APL error messages along with some possible causes for the error condition and a suggested user's response.

Error Message	Cause	User's Response
ALREADY MARKED	The specified file was previously marked.	If the file(s) is to be re-marked, enter GO. *Note:* For tape files, any existing data in the files following the last re-marked file will no longer be available.
DEFN ERROR	An invalid request to use the function definition mode was made:	
	• A ∇symbol was erroneously used in a statement.	If the statement was intended to open or close a function, the ∇ is valid only in the beginning and ending positions.
	• An attempt was made to reopen a locked function.	
	• An attempt was made to reopen a function using more than just the function name.	Enter a corrected statement.
	• An attempt was made to open a new function definition using the name of a previously defined global variable name.	Enter a different function name or erase the global variable.
	• An invalid edit request was made in function definition mode.	Enter a valid edit request.
	• An attempt was made to edit a pendent function.	If the suspended function execution can be terminated, clear the state indicator, then edit the function.

Error Message	Cause	User's Response
DEVICE NOT OPEN	An attempt was made to read a data file, and an input operation to the file has not been specified.	Assign the information required to specify the input/output operation to the shared variable.
DOMAIN ERROR	The function indicated by the caret (∧) cannot operate on the arguments given: • The result exceeds the capacity of the 5110 (<¯7.237 . . . E75 or >7.237 . . . E75). • A character argument cannot be used in an arithmetic operation. • The argument is not mathematically defined for the function (that is, 12÷ 0). • Numeric and character data cannot be cantenated together. • An error occurred in a locked function. • Format length is incorrect.	Determine the correct arguments for the function in error. Then correct the statement in error.
ENTRY ERROR	An invalid character was entered.	Enter a corrected statement.

The input/output device errors are displayed in the format ERROR eee ddd, where eee is the error code and ddd is the device number. The device numbers are: 500–printer; 001–built-in tape unit; 002–auxiliary tape unit; 011–diskette drive 1; 012–diskette drive 2; 013–diskette drive 3; 014–diskette drive 4.

Many of the input/output errors relating to tape or diskette can be the result of a machine problem or a problem with the tape or diskette itself. Some problems that can be encountered with the tape or diskette are:

- Poor or worn oxide coating
- Binding tape cartridge spools
- Tape cartridge drive band off
- Diskette binding in the envelope
- Diskette drive hole worn too large or off center
- Diskette index hole off location

Error codes 004 to 008 for tape and error codes 015 and 045 for diskette can be caused by the conditions listed previously, as well as by the causes listed next. You might save yourself time and any expenses associated with a service call if you make a thorough check of your media (tape or diskette) before calling your service representative.

Following is a list, additional causes, and user's response for the input/output device error messages.

Error Message	Cause	User's Response
ERROR 001 ddd	Diskette drive timing malfunction.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 002 ddd	Command error.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 003 ddd	Tape error.	
ERROR 004 ddd	Tape error or second tape not ready.	
	An attempt was made to perform tape operations with an unmarked tape.	Use the)MARK command to mark the tape. Then try the operation again.
	Uneven winding of the tape.	Move the tape to its limits using the procedure described under *Tape Data Cartridge Handling and Care*, in Chapter 2.
	The tape head needs cleaning.	Clean the tape head using the procedure described under *Tape Head Cleaning Procedure*, in Chapter 2.

Error Message	Cause	User's Response
ERROR 005 ddd	The tape cartridge is not inserted in the indicated tape unit.	Insert a tape cartridge and try the operation again.
ERROR 006 ddd	An attempt was made to write on a tape that is file-protected. (The SAFE switch on the tape cartridge is in the SAFE position.)	If you want to write on the tape, set the SAFE switch on the tape cartridge off the SAFE position.
ERROR 007 ddd	Tape read error.	Use the)LINK command to load the tape recovery program to recover as much data as possible.
ERROR 008 ddd	A machine malfunction might have occurred when data was being written or the tape cartridge might have been removed from the tape unit when data or a workspace was being written to tape.	Try the operation again. If the error occurs again, copy the files following the file that caused the errors onto another tape. Also, if the file is a saved workspace file, some of the data in the file might be retrieved if you use the)COPY command. Then use the)MARK command and re-mark the tape from the file that caused the error.
ERROR 009 ddd	An attempt was made to read a record after the EOD address.	
ERROR 010 ddd	Data is to be written to a data file, but all the space in the file has been used.	Use the)MARK command to format a larger file and do the operation again.
	An attempt was made to write the active workspace on tape with a)SAVE command, but the specified file could not contain all the information from the active workspace.	Use a larger file to save the active workspace.
	For record I/O operations, an attempt was made to write a record after the end of data.	
ERROR 011 ddd	A file number was specified that does not exist on the tape.	Specify the correct file or use the)MARK command to mark the tape.
ERROR 012 ddd	The end of the tape has been reached.	Use another tape cartridge.

Error Message	Cause	User's Response
ERROR 013 ddd	The specified device is not attached.	
ERROR 014 ddd	Device error.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 015 ddd	An input/output error occurred when:	
	• The system was reading or writing the volume label.	Use the initialization utility to reinitialize the diskette.
	• The system was reading or writing a file header.	Copy all available data onto another diskette, and then use the initialization utility to reinitialize the diskette.
ERROR 016-019 ddd	Faulty diskette.	Copy all available data onto another diskette, and then use the initialization utility to reinitialize the diskette.
ERROR 020-023 ddd	Hardware malfunction.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 024 ddd	There is no VOL1 label on the diskette.	Use the initialization utility to initialize the diskette.
ERROR 025 ddd	Invalid diskette.	Use the initialization utility to initialize the diskette.
ERROR 026-028 ddd	Hardware malfunction.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 029 ddd	A match was not found for a search by key operation.	See Chapter 7 for a description of search by key operation.
ERROR 030 ddd	The diskette drive cover was opened between accesses to the device.	Close the diskette drive cover, and try the operation again.
ERROR 031 ddd	Faulty diskette.	Copy all available data onto another diskette, and then use the initialization function to reinitialize the diskette.
ERROR 032-035 ddd	I/O processor error.	Try the operation again. If the error occurs a second time, call your service representative.

Error Message	Cause	User's Response
ERROR 036 ddd	An attempt was made to use the)MARK command to mark a file, and one of the following conditions occurred:	
	• More than the maximum number of files allowed on the diskette was specified.	Mark the files on another diskette.
	• Not enough continuous space was available on the diskette for the file.	Use the compress function to place all the unallocated diskette space in one continuous area, or mark the file on another diskette.
	I/O processor error.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 037 ddd	I/O processor error.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 038 ddd	Invalid file header.	Copy all available data onto another diskette, and then use the initialization function to reinitialize the diskette.
ERROR 039 ddd	An attempt was made to write to a write-protected file.	If you want to write to the file, use the)PROTECT command to remove the write-protect indicator and then write to the file.
ERROR 040 ddd	An I/O operation was specified with only the file ID, and more than one file on the diskette have the same file ID.	Issue the command using the appropriate file number.
ERROR 041 ddd	An I/O operation was specified with only the file ID, and no matching file ID was found.	Insert the correct diskette and reissue the command, or reissue the command using the correct file ID.
	The file name already exists on the diskette.	Issue the command or statement using a new file ID.
ERROR 042 ddd	A)MARK command was issued, but there is not enough unallocated continuous bytes of storage available on the diskette to format the file(s).	Use the compress function to position all the unallocated storage as continuous bytes. Then try the)MARK command again. If the error occurs again, use another diskette.
ERROR 043 ddd	An attempt was made to access a volume protected diskette.	Use the)VOLID command to turn off the volume protection indicator.
ERROR 044 ddd	An attempt was made to access a bad sector in the data file.	See return code 22 n in Chapter 7.

Error Message	Cause	User's Response
ERROR 045 ddd	The diskette is inserted incorrectly.	Make sure the diskette is inserted correctly and try the operation again.
ERROR 047-049 500	Printer errors.	Try the operation again. If the error occurs a second time, call your service representative.
ERROR 050 500	The printer has run out of forms.	Insert forms in the printer (see Chapter 8).
ERROR 051 500	The printer POWER ON/OFF switch is turned off.	Turn the POWER ON/OFF switch on.
ERROR 052-053 500	Printer errors.	Check to be certain that both the forms tractor pins and platen are not engaged simultaneously. If the error occurs a second time, call your service representative.
ERROR 054 500	The printer was turned off within two seconds after a system command was entered. The error continues to occur when the printer is turned on again.	Save any data or user-defined functions in the active workspace, then enter the)CLEAR command to get the printer into synchronism with the 5110.
ERROR 055-059 500	Printer errors.	See ERROR 052-053 500.
ERROR 070-072	Processor error.	Try the operation again. If the error occurs a second time, call your service representative.
EXCEEDED MAXIMUM RECORD LENGTH	The tape or diskette was removed before an input/output operation was terminated (see Chapter 7).	The data in the file cannot be used.
	When the tape or diskette copy utility was used, a header record was changed to specify a maximum record length less than the largest record in the file.	Use the tape or diskette copy utility and copy the tape containing the invalid header record again, this time specifying a larger maximum record length.

Error Message	Cause	User's Response
FILE NOT FOUND	A system command or I/O operation was specified with only the file ID, and no matching file ID was found.	Insert the correct diskette and reissue the command, or reissue the command using the correct file ID.
FILE NOT UNIQUE	The file name already exists on the diskette.	Issue the command or statement using a new file ID.
	A system command was issued specifying only the file ID, and more than one file on the diskette has the same file ID.	Issue the command using the appropriate file number.
IMPLICIT ERROR	The system variable that precedes the error message was previously assigned an invalid value or was undefined in a function due to the system variable being made local to the fucntion. *Note:* This error message is not displayed until the system variable in error is used by the APL system.	Assign a valid value to the system variable (see Chapter 5).

Error Message	Cause	User's Response
INCORRECT COMMAND	A system command was entered incorrectly: • The command keyword was not a valid keyword. • One of the parameters was entered incorrectly. • Too many parameters were entered for the command.	Enter the command in its correct form.
INDEX ERROR	The index values given are outside the boundaries of the array, or a primitive function or APL operator being subscripted by index [I] has been given an argument that does not have an I^{th} dimension.	If a variable is being indexed, check its shape (ρ A) against the index values. If a primitive function or operator is being indexed, determine the rank(s) (ρ ρ A) of its argument(s); then check the index to see if it is equal to or less than the required rank. Check the index origin (⎕*IO*) to ensure that it is consistent with the statement being executed.
INTERFACE QUOTA EXHAUSTED	An attempt was made to establish more than 12 variable names as shared variables.	Retract any unused share variable names.
INTERRUPT	The shift key was held down and the ATTN key pressed when the 5110 was processing data, or an invalid input/output operation was attempted.	If an invalid input/output operation was attempted, check to make sure the operation was specified correctly.
INVALID DATA TYPE	An attempt was made to write to an interchange format file (TYPE = I), but the data is not a character scalar or vector.	Change the data to a character scalar or vector.
	An attempt was made to specify an input/output operation with other than character data.	Enclose the information required to specify the input/output operation in single quotes.
	The data assigned to the shared variable is not the correct shape and representation for the creation of direct access APL internal format data file.	You must assign data to the shared variable that is the same shape and representation as the first record written to the file.
INVALID DEVICE	A device was specified that does not exist or is incorrect for the operation to be performed.	Specify the correct device number.

Error Message	Cause	User's Response
INVALID DEVICE NUMBER	A device number that does not exist was specified.	Specify the correct device number.
INVALID DISK FILENAME	An attempt was made to use an invalid diskette file name.	Change the name to a valid file name (see *Variable and File Names* in Chapter 2).
INVALID FILE	The file type is not valid for the attempted operation. For example, an attempt was made to load a data file or read a workspace file.	Use the)LIB command to determine the file type.
	An attempt was made to access a file number that is 1 larger that the last marked file.	Specify the correct file number.
	An attempt was made to load or copy a damaged file. The file was probably damaged by the tape being removed from the tape unit before a save operation was complete.	The data in the file is unusable. The file can be dropped (use the)DROP command) and reused.
	The wrong file ID was specified.	Use the)LIB command to find the correct file ID, and reenter the statement.
	An attempt was made to load a 5100 continued file (file type 6) into a 5110.	Load the file into a 5100, and store the file on tape using the)SAVE command. Then load the saved file into the 5110.
	An attempt was made to use a file type 2 (interchange file) as a procedure file. However, the file has records that exceed 128 characters, or the BASIC SAVE DATA command was used to create the file.	Use shared variables to read the file, and then rewrite the file with no records exceeding 128 characters.
INVALID FILE NUMBER	The file number 0 was specified for a)LOAD,)SAVE,)CONT,)DROP,)COPY, or)PCOPY command.	Reenter the command specifying the correct file number.
	An attempt was made to specify an input/output operation to a data file, but the file number was not valid.	Use the)LIB command to find the correct file number. Then reenter the information required to specify the input/output operation.
INVALID PARAMETER	A keying error was made, or an incorrect parameter was entered when you specified an input/output operation.	Enter the information required for the input/output operation or system command again, correcting the keying errors.
	A keying error was made when you entered the parameters for a system command.	

Error Message	Cause	User's Response
IO STATUS: INVALID OPERATION	An invalid input/output operation was attempted, such as:	
	• An attempt was made to read the file header in other than nontranslate mode (TYPE = N).	Respecify the operation using the TYPE = N parameter, and then try the operation again.
	• The shared variable was referenced more than once to check the return code.	Assign the return code to another variable if the return code must be checked more than once.
	• A processor other than 1 (the left argument for the ⎕ SVO function) was established for the shared variable.	Retract the shared variable offer, and then make the offer again using a 1 as the left argument for the ⎕ SVO function.
	This message is followed by the INTERRUPT error message.	
LENGTH ERROR	The shapes of the two arguments are not valid for the function indicated by the caret (∧).	Make sure the arguments are valid for the function. Then reshape (restructure) the arguments.
LINE TOO LONG	An attempt was made to edit a statement (in a user-defined function) that is greater than 115 characters.	Break up the statement into two statements, or use the ⎕CR and ⎕FX functions to edit the statement.
	An attempt was made to save a workspace that contained a user-defined function with a statement having more than 115 characters. In this case, the error message is preceded by the function name and the statement number that caused the error.	Use the ⎕CR function to make the user-defined function a matrix; then save the workspace on tape.
NONCE	An I-beam function was used. These functions are not used in the 5110 APL system.	Do not use the I-beam functions.
	An attempt was made to index a portion of an array with a rank greater than 14.	Display the entire array, or break the array into smaller sections.
	An attempt was made to use a take or drop operator on an array with a rank greater than 9.	Break the array into smaller rank arrays.
	An attempt was made to laminate an array with a rank greater than 20.	Break the array into smaller rank arrays and reshape.

Error Message	Cause	User's Response
NOT COPIED: names	A)PCOPY was issued, but each object named in the message was not copied. The active workspace already contained a global object with the same name.	Issue a)COPY command if the named objects should be copied.
	A)COPY was issued, but each object named in the message was not copied. The active workspace already contained a shared variable with the same name.	Establish and use another shared variable name, and erase the existing shared variable name if the objects should be copied.
NOT FOUND: names	An)ERASE command was issued, but the global objects named in the message were not found in the active workspace.	Reissue the command using the correct object names.
	A)COPY or)PCOPY command was issued, but the specified global object does not exist in this specified workspace.	Reissue the command using the correct object name or stored workspace.
NOT SAVED, THIS WS IS workspace ID	A)SAVE or)CONTINUE command was issued, but the stored workspace ID is not the same as the active workspace ID.	Use the correct ID or change identification of the active workspace, using the)WSID command; then reissue the)SAVE command.
NOT WITH OPEN DEVICE	An attempt was made to issue a system command or specify an input/output operation to a tape unit that is already being used for input/output operations.	Terminate the input/output operation, or wait until the input/output operation is complete before issuing the command or specifying another input/output operation to the tape unit.
	An)OUTSEL command was issued, but printer output has been specified for a shared variable.	Retract the printer shared variable offer.
NOT WITH SUSPENDED FUNCTIONS	An attempt was made to do a)SAVE,)COPY, or)PCOPY operation, and the active workspace contains a suspended function or an open request for quad input.	Clear the suspended function or request for quad input by using → (right arrow).
NOT WITH SYSTEM ERROR	An attempt was made to do an operation other than)CLEAR after a SYSTEM ERROR occurred.	(See SYSTEM ERROR.)
RANK ERROR	An attempt was made to use a function that requires the rank of the arguments to conform, but they do not. For example, a function requires the rank of the arguments to be the same, but they are not.	Make sure the arguments are valid. Then reshape (restructure) the arguments so that they have the correct rank (⍴⍴A).
	An attempt was made to use an argument whose rank is too large for the operation.	
	The number of semicolons in the index does not equal the rank minus 1.	Use the correct number of semicolons.

Error Message	Cause	User's Response
SI DAMAGE	The state indicator was made invalid because one of the following occurred:	
	• A function exists in the state indicator list, but the function was erased.	Use the)SI or)SINL command to display the state indicator. Clear out the state indicator by entering → for each * in the)SI list.
	• A suspended function's header was changed or displayed [O⎕].	
	• A label was added, removed, or changed on the suspended statement.	
SYMBOL TABLE FULL	More symbols were used than the number of symbols allowed. The symbol table in the stored workspace is full, and a load operation was attempted. This error is caused by the latent expression variable even if it has not been assigned.	Use the)SAVE command to save the workspace, use the)CLEAR command to clear the active workspace, increase the number of symbols allowed by using the)SYMBOLS command, then use the)COPY command to copy the stored workspace into the active workspace. *Note:* Erasing a symbol from the active workspace does not remove it from the symbol table however, saving the active workspace and loading it again will remove any unused symbols from the symbol table.
SYNTAX ERROR	The part of the statement indicated by the caret (∧) is syntactically invalid.	Enter a corrected statement. Be sure to check for the correct number of parentheses and arguments.
	The symbol used for the APL built-in function is not a valid symbol.	See Appendix B for a list of the characters that can be confused with valid APL built-in functions. Reenter the statement using the correct symbol for the APL built-in function.

Error Message	Cause	User's Response
SYSTEM ERROR	A malfunction occurred in the APL system program, and the data in the active workspace is lost.	Enter the)CLEAR command; if the error continues to occur, call your service representative. *Note:* There might be some system data areas displayed with this message. If SYSTEM ERROR occurred on a load or copy operation, the error may be caused by a bad stored workspace file. Try loading or copying another stored workspace file to see if the error occurs again.
	An attempt was made to load a stored workspace file after the file header was changed by the tape or diskette copy utility.	Enter)CLEAR and use the)COPY command to copy the stored workspace into the active workspace. Then use the)SAVE command to write the data back onto the stored workspace file. Now, the)LOAD command can be used to load the stored workspace into the active workspace.
VALUE ERROR	The object indicated by the caret (∧) has not been given a value: • If the object is a variable name, the variable was not previously assigned a value. • If the object is a function name, the function header did not specify a result, the function did not assign a value to the result variable, or the function does not exist.	Assign a value for the indicated variable or correct the function so that it has an explicit result. The value must be assigned before the object is used.
WRITE PROTECTED	An attempt was made to write information to a diskette file, and the write-protect indicator is on.	If the information is to be written to the file, enter)PROTECT *file number* OFF to remove the write-protect indicator.
WRONG VOLUME	An attempt was made to read from or write to a diskette data file, and the diskette is not on the same diskette drive as when the operation was specified.	Place the diskette containing the data file in the same diskette drive as when the input/output operation was specified, and try the operation again.
	The RESUME command was issued and the diskette with the data file is not in the same drive as when the original input/output operation was specified.	

Error Message	Cause	User's Response
WS TOO BIG	One of the following conditions occurred:	
	• An attempt was made to load a workspace stored with the)CONTINUE command into a 5110 with less internal storage.	Use a 5110 with enough internal storage.
	• An attempt was made to load a workspace stored with the)CONTINUE command into the active workspace, but IMFs have been applied, reducing the available internal storage.	Restart to clear the IMF, load the stored workspace into the active workspace,)SAVE the active workspace, apply the IMFs, then load the stored workspace again or copy only the required objects.
	• An attempt was made to write the active workspace (using the)CONTINUE command) into a file that is too small.	Use a file that is large enough.

Error Message	Cause	User's Response
WS FULL	One of the following conditions occurred:	Erase unnecessary objects. If there is still not enough space:
	• A)COPY or)PCOPY command was issued, but the active workspace could not contain all of the objects requested.	– Partition the workspace into two or more workspaces with related functions.
	• The active workspace could not contain all the information required to build a define function.	– Store data in a separate workspace or in a data file.
	• The active workspace could not contain the intermediate results of an APL expression.	– Reprogram using smaller intermediate results.
	• The active workspace could not contain information required to do input/output operations.	– Clear the state indicator with → if suspended functions exist. – Reduce the size of the symbol table. See note under)SYMBOLS.
	• A workspace was written to tape with a)SAVE command, but the extra storage required to load the stored workspace back into the active workspace exceeds the available storage.	Use the)COPY command to make the stored workspace into two workspaces.
	• Too many symbols were specified in a)SYMBOLS command.	Reenter the command with the correct password specified.
WS LOCKED	The workspace is password-protected, but no password or the wrong password was specified in the command.	Reenter the command with the correct password specified.
	The workspace is not password-protected, and a password was specified.	Enter without a password.
WS NOT FOUND	A)LOAD,)DROP,)COPY, or)PCOPY command was issued, but there is no stored workspace with the identification specified in the command.	Reenter the command with the correct workspace identification.

Appendix G. Attaching a TV Monitor

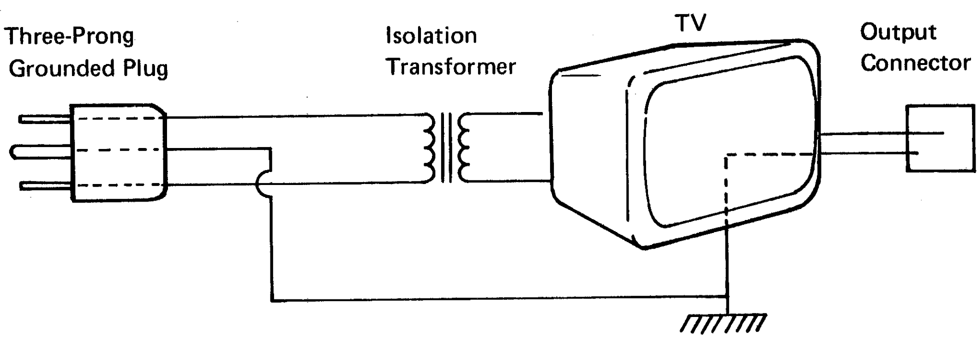
Generally, modifying a standard TV set and using it as a video monitor yields less satisfactory results than a regular video monitor. This is because the same level of quality is not built into TV receivers as is found in monitor class units. For example, the contrast and resolution are not as good on a modified TV receiver; thus the image is not as sharp and usually more difficult to read.

However, if you choose to modify a TV receiver and use it as a video monitor you must observe the following or you may damage the 5110 and expose yourself to a severe electric shock when you attempt to hook up the TV set to the IBM 5110.

A modified TV set must have isolation between the primary line voltage and the set's chassis and circuitry. You can usually accomplish this by using an isolation transformer between your outlet line voltage and the input voltage to the TV set. This transformer should be permanently wired into the circuit. The new input power plug must be a three-prong grounded plug with the ground connected to the chassis of the TV set. This grounding circuit must be electrically connected to the 5110 grounding circuit.

Before the video input is connected to anything, it should be tested to verify that the connector's external shell is at ground potential and that no line voltage is present on either the external shell or the center conductor.

It is the responsibility of the TV modifier to ensure that the input circuit meets the requirements of the 5110 output and will not damage the 5110.



Note: If you do elect to modify a TV receiver for use as a video monitor, IBM accepts no responsibility for safety precautions during conversion and hookup, for damages incurred to the TV receiver or 5110, or for the quality of the TV receiver as a video monitor.

Glossary

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the *American National Standard Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3.

ANSI definitions are identified by an asterisk. An asterisk to the right of the term indicates that the entire entry is reprinted from the *American National Standard Vocabulary for Information Processing*.

active referent: The usage of a name that was most recently localized, or the global usage if the name is not localized.

active workspace: A part of internal storage where data and user-defined functions are stored and calculations are performed.

ADD operation: An operation using a shared variable to add information to an existing data file.

alphameric keys: The keys on the left side of the keyboard whose arrangement is similar to that of a typewriter keyboard.

APL internal data format: See *internal data format.*

arguments: Data supplied to APL functions.

array: A collection of data that can range from a single item to a multidimensional data configuration. Each element of an array must be the same type as the other elements (all characters, all numeric, or all logical).

assign: To use the ← (assignment arrow) to associate a name with a value.

available storage: The number of unused 1024-byte blocks of storage in a file on tape.

bare output: Output displayed without the cursor returning to the next line.

basic data exchange: A diskette data exchange format that uses 128-byte sectors. The basic exchange format allows you to exchange data between the 5110 and other IBM systems. For example, data recorded on the diskette using a 3741 can be processed using a 5110.

branch instruction: An instruction that modifies the normal order of execution indicated by the statement members. Branch instructions always begin with a → (branch arrow).

branching: Modifying the normal order of execution indicated by the statement numbers.

built-in function: See *primitive function.*

byte: A unit of storage. For example, a character takes 1 byte of storage.

character constants: Characters that do not represent numbers, variables, or functions. Character constants are enclosed in single quotes when they are entered (except for ⍞input); however, the single quotes do not appear when the character constants are displayed.

command keyword: The name of a system command including the right parenthesis. For example, the command keyword for the)MARK command is)MARK.

comment: An instruction or statement that is not to be executed. A comment is indicated by a ⍝ as the first character.

conditional branch: A branch that is taken only when a certain condition is true.

coordinate: A subset of data elements in an array. For example, a matrix has a row coordinate and a column coordinate.

cursor: The flashing character on the display that indicates where the next input from the keyboard will be displayed.

data file: A file on tape or diskette (file type 02, 08, 09, 10, or 15) where data is stored by means of a share variable.

defective record: A 512-byte block of storage on tape that cannot be read.

device/file number: A number that specifies the device and file to be used for input or output operations.

direct access data file: A diskette data file in which records can be read from or written to by means of a specified record number.

DISPLAY operation: An operation using a pair of shared variables to read and write data on the display screen.

dual-language machine: A 5110 that can execute either APL or BASIC statements.

dyadic functions: Functions that require two arguments (a right and a left argument).

editing: Modifying an instruction or statement that already exists.

element: The single item of data in an array.

empty array: A variable that has a zero in its shape vector. The array has no (zero) elements.

execute: To press the EXECUTE key to process data on the input line.

execution: The processing of data.

execution mode: The mode that is operative when statements or functions are executed. Contrast with *function definition mode.*

explicit result: The result of a function that can be used in further calculations. The function must contain a result variable if it is to have an explicit result.

file: A specified amount of storage on the media. The media is formatted into files by the)MARK command.

file ID: The name of a file on the media. If the file contains a stored workspace, the file ID is the same as the stored workspace ID.

file type: The identification of the type of data stored in a file.

function body: The statements within a user-defined function. These statements determine the operation(s) performed by the function.

function definition: The definition of a new function (a user-defined function) to solve a problem.

function definition mode: The mode that is used for defining or editing user-defined functions. The ∇ symbol is used to change the mode of operation. Contrast with *execution mode.*

function header: A definition that states the function name, number of arguments, local names, and whether or not the function will have an explicit result.

general exchange data file: A file with data in the general exchange format.

general exchange data format: A data format consisting of all character scalars or vectors. This file can be used as a 5110 BASIC source file.

global names: Names having a value or function that can be used within or outside a user-defined function unless the name has been made local to a user-defined function that is executing, suspended, or pendent. Contrast with *local name.*

identity element: The value that generates a result equal to the other argument of a function.

IN operation: An operation using a shared variable to read information from a data file.

index entry [I]: (1) A value or values enclosed in brackets that select(s) certain elements from an array. (2) A value enclosed in brackets that determines the coordinate of an array to be acted on by a primitive mixed function.

index origin: Either 0 or 1, and the lowest value of an index. The index origin is set to 1 in a clear workspace and can be changed by means of the ⎕IO system variable.

INR operation: An operation using a pair of shared variables to read records from a direct access data file.

IOR operation: An operation using a pair of shared variables to read records from or write records to a direct access data file.

IORH operation: An operation using a pair of shared variables to read records from or write records to a direct access data file.

input: Information entered from the keyboard or read from tape by means of a shared variable.

input line: A line consisting of the 128 positions on lines 0 and 1 of the display screen. Any information on the input line will be processed when the EXECUTE key is pressed.

instruction: A function or series of functions to be performed.

integer: A whole number.

interactive function: A user-defined function that requests input from the keyboard as it executes.

internal data file: A file containing data in the internal data format.

internal data format: The format in which the data is stored in the 5110.

keyword: See *command keyword*.

labels: Names that are placed on statements in a user-defined function for use in branching.

latent referent: The usage of a name that has been made unavailable by a more recently called function. The usage for that name cannot be accessed.

length: (1) The length of a vector is the number of elements in the vector. (2) The length of a coordinate of other arrays is the number of items specified by that coordinate. For example, a matrix has a row coordinate with the length of 2; therefore, the matrix has two rows.

library: A tape cartridge or diskette where data is stored for future use.

local name: A name that is contained in the function header and has a value specified to the user-defined function during the execution of that user-defined function.

locked function: A function that cannot be revised or displayed in any way. The opening or closing ∇ was overstruck with a ~.

logical data: (Boolean data) Data that consists of all ones and zeros.

matrix: A collection of data arranged in rows and columns (rank 2).

media: Tape or diskette storage.

mixed function: A function having results that may differ from the arguments in both rank and shape.

monadic functions: Functions that require one argument. The argument must be to the right of the function symbol.

multidimensional array: An array that has two or more coordinates.

n-rank array: An array that has more than two coordinates (a rank of more than 2).

niladic function: A user-defined function that does not require any arguments.

numeric keys: The keys on the right side of the keyboard whose arrangement is similar to that of a calculator keyboard.

object: A user-defined function or variable name.

open data files: Tape or diskette data files that are currently being used for input and/or output operations.

operators: Controls that have as their arguments dyadic primitive scalar functions. These arguments are applied to arrays in a specified way.

OUT operation: An operation using a shared variable to write information into a sequential access data file.

OUTF operation: An operation using a shared variable to write information (sequentially) to a direct access data file.

output: The results of statements processed by the 5110.

overstruck character: A character formed by entering one character, backspacing, and entering another character. Only certain combinations of characters can form overstruck characters.

parameters: (1) Information needed by a system command (such as device/file number). (2) Information required to open a data file or to specify printer output.

password: A sequence of characters that must be matched before the contents of a stored workspace can be loaded or copied into the active workspace.

pendent function: Any function in the state indicator list that is not a suspended function.

physical record: A 512-byte block of storage on tape or one sector on diskette.

plane: The third coordinate of an n-rank array (planes, rows, and columns).

primitive function: The functions that are part of the APL language (such as , + - ÷ ×).

PRT operation: An operation using a shared variable to output data on the printer.

rank: The number of coordinates of an array (ρ ρ).

record: Data assigned to a shared variable.

result variable: A variable to the left of the assignment arrow in the function header where the results of the function are temporarily stored for use in further calculations.

return code: A code assigned to a shared variable to indicate whether or not the operation was successful.

scalar: A single data item that does not have a dimension (0 = ρ ρ DATA).

scalar function: A function that is applied to corresponding elements in arguments. The results of scalar functions are the same shape as the arguments.

scale: An integer representing the power of 10 in scaled representation.

scaled representation: A value stated in a convenient range and multiplied by the appropriate power of 10.

scroll: To move the information on the display screen up or down.

shape: The length of each coordinate of an array.

sector: A portion of a diskette cylinder. A sector can be 128, 256, 512, or 1024 bytes; however, all sectors on a diskette cylinder must have the same number of bytes.

sequential access data file: The data in the data file must be retrieved in the same order as the data was written to the data file.

shared variable: A variable shared by the active workspace and shared variable processor. Used to transfer data for input/output operations.

significant digit: *A digit that is needed for a certain purpose, particularly one that must be kept to preserve a specific accuracy or precision.

single-element array: An array with a shape of all 1's; for example, a matrix with one row and one column.

state indicator: An entry that contains information on the progress (statement number of the statement being executed) of user-defined function execution. Can be displayed to show all suspended and pendent user-defined functions and localized names.

statement: A numbered instruction within a user-defined function.

statement number: The number of a statement within a user-defined function.

stop control (SΔ): A function that stops execution of a user-defined function before the execution of a specified statement.

stop vector: A vector that specifies the statements when stop control is used.

stored workspace: The contents of the active workspace stored on tape.

strong interrupt: The result of holding down the shift key and pressing ATTN. The execution of an expression or a user-defined function is abandoned. Also, for user-defined functions, the pendent functions are cleared from the state indicator.

suspended: See *suspended function.*

suspended execution: See *suspended function.*

suspended function: A function whose execution has stopped because of an error condition, the ATTN key being pressed, or stop control being used.

system commands: A set of commands used to manage the active workspace and tape or printer operations.

system functions: Functions used to change or provide information about the system.

system operation: An operation that processes input data.

system variable: A variable that provides controls for the system and information about the system to the user.

trace control (T∆): A function that displays the results of specified statements during the execution of a user-defined function.

trace vector: A vector that specifies the statements used in trace control.

transferring data: Using a shared variable to write data to tape, read data from tape, or output data to the printer.

user-defined functions: New functions using the primitive functions but defined by the user. See *function definition mode.*

variable name: A name associated with the value of a variable.

variables: Data stored in the 5110.

vector: An array with one dimension (1 = ⍴⍴DATA).

weak interrupt: The result of pressing the ATTN key. The execution of a user-defined function is suspended. Contrast with *strong interrupt.*

workspace: See *active workspace.*

workspace available: The amount of unused storage (number of unused bytes) in the active workspace (⎕WA).

workspace ID: A name given to the contents of the active workspace. A stored workspace has the same name as the active workspace when the contents of the active workspace were written to tape.

Index

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

Name ____________________

Address ____________________

____

● No postage necessary if mailed in the U.S.A.

SA21-9303-0

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM 5110 APL Reference Manual Printed in U.S.A. SA21-9303-0

IBM®

International Business Machines Corporation

General Systems Division
5775D Glenridge Drive N. E.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM 5110 APL Reference Manual Printed in U.S.A. SA21-9303-0

SA21-9303-0