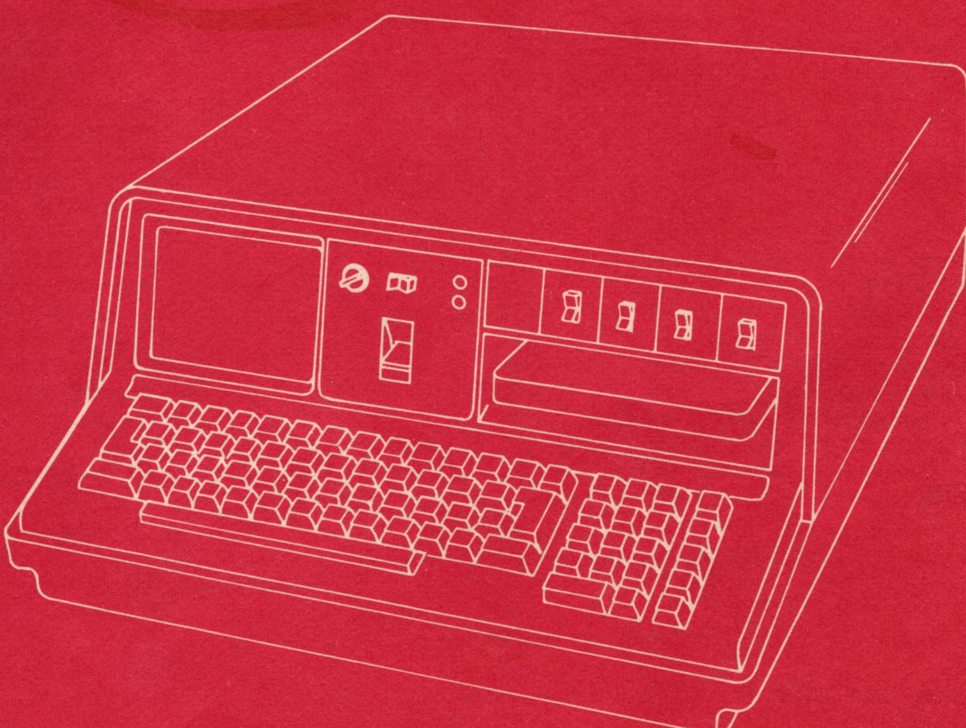
IBM

5100

IBM 5100
APL Reference Manual



IBM 5100
Portable Computer
APL Reference Manual

SA21-9213-2

Preface

This publication is a reference manual that provides specific information about the use of the IBM 5100 Portable Computer, the APL language, and installation planning and procedures. It also provides information about forms insertion and ribbon replacement for the 5103 printer. This publication is intended for users of the 5100 and the APL language.

Prerequisite Publication

IBM 5100 APL Introduction, SA21-9212

Related Publications

- *IBM 5100 APL Reference Card,* GX21-9214
- *APL Language,* GC26-3847

Third Edition (May 1976)

This is a major revision of, and obsoletes, the previous edition SA21-9213-1 and Technical Newsletter SN21-0258.

Changes have been made throughout, so this manual should be reviewed in its entirety.

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

A form for readers' comments is at the back of this publication. If the form is gone, address your comments to IBM Corporation, Publications, Dept 245, Rochester, MN 55901.

Contents

IBM 5100 PORTABLE COMPUTER OVERVIEW

The 5100 (Figure 1) is a portable computer. The 5100 has a display screen, keyboard, a tape unit, 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 5100 are an auxiliary tape unit, a printer, and a communications adapter.

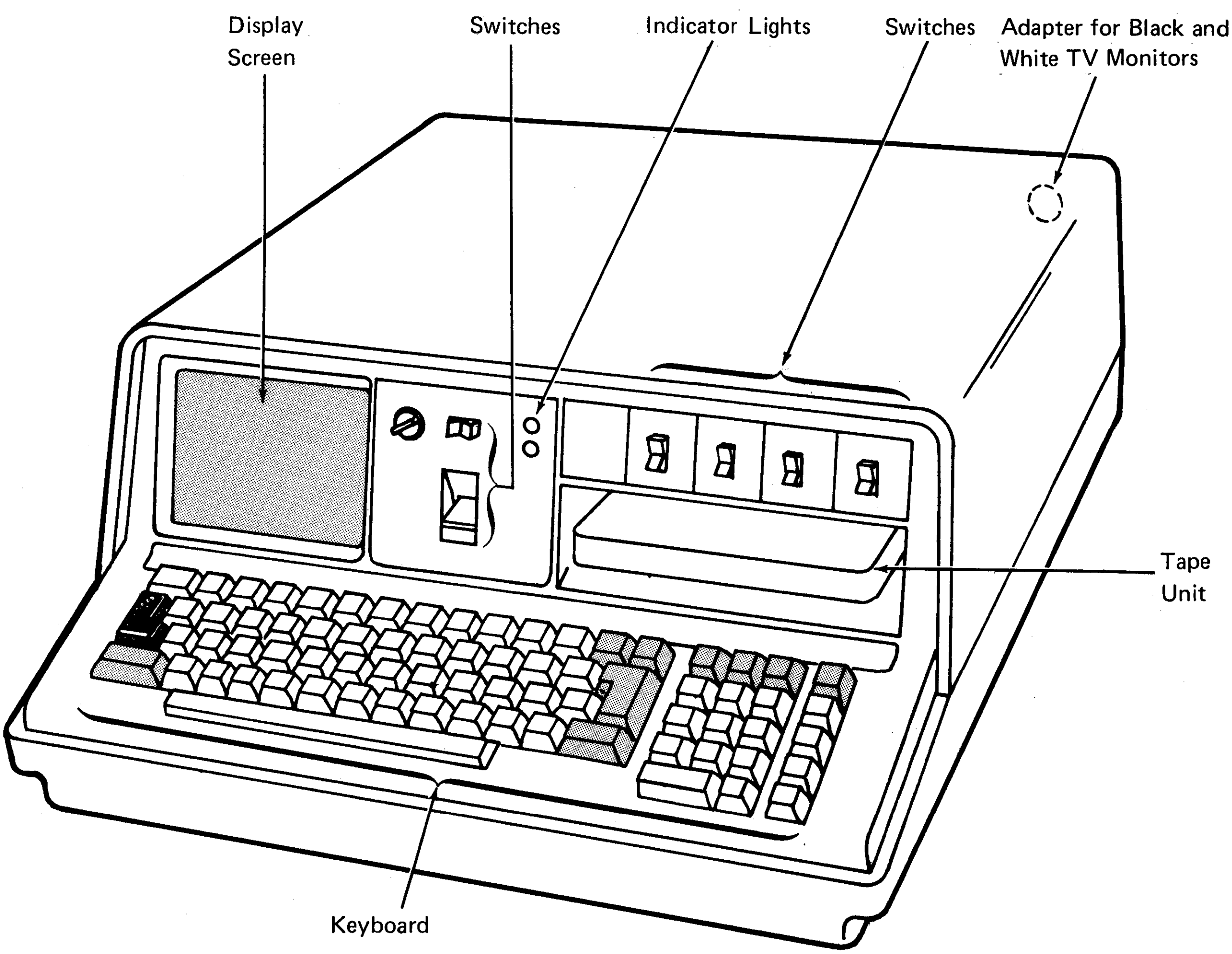


Figure 1. IBM 5100 Portable Computer

DISPLAY SCREEN

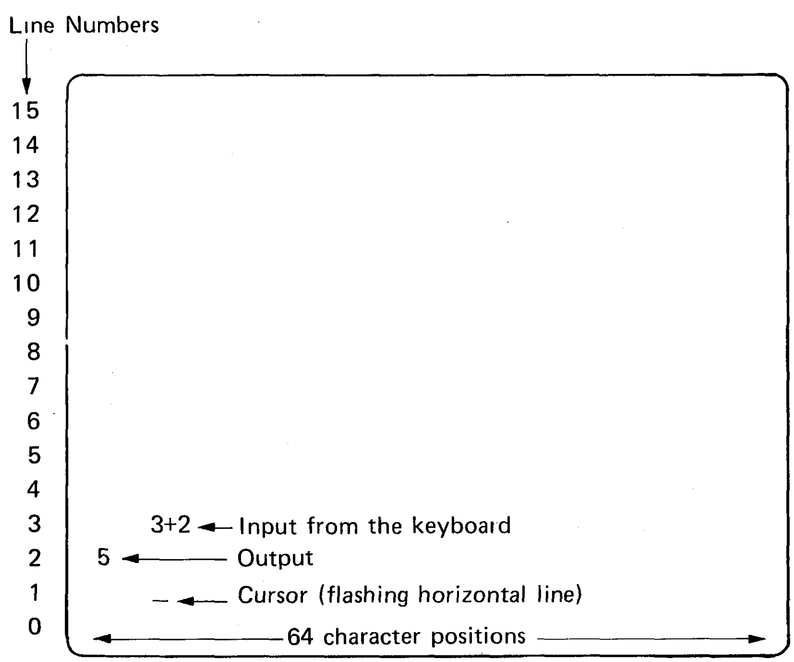
The display screen (Figure 2) can display 16 lines of information at a time, with up to 64 characters in each line. Input (information supplied by the user) as well as output (processed information) is displayed. The bottom two lines (lines 1 and 0) of the display contain information entered from the 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 5100 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 of the display screen.

SWITCHES

The switches on the 5100 console (Figure 3) 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 when 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 5100 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 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 15-20 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 15-20 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 (see Chapter 2) 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

This 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.

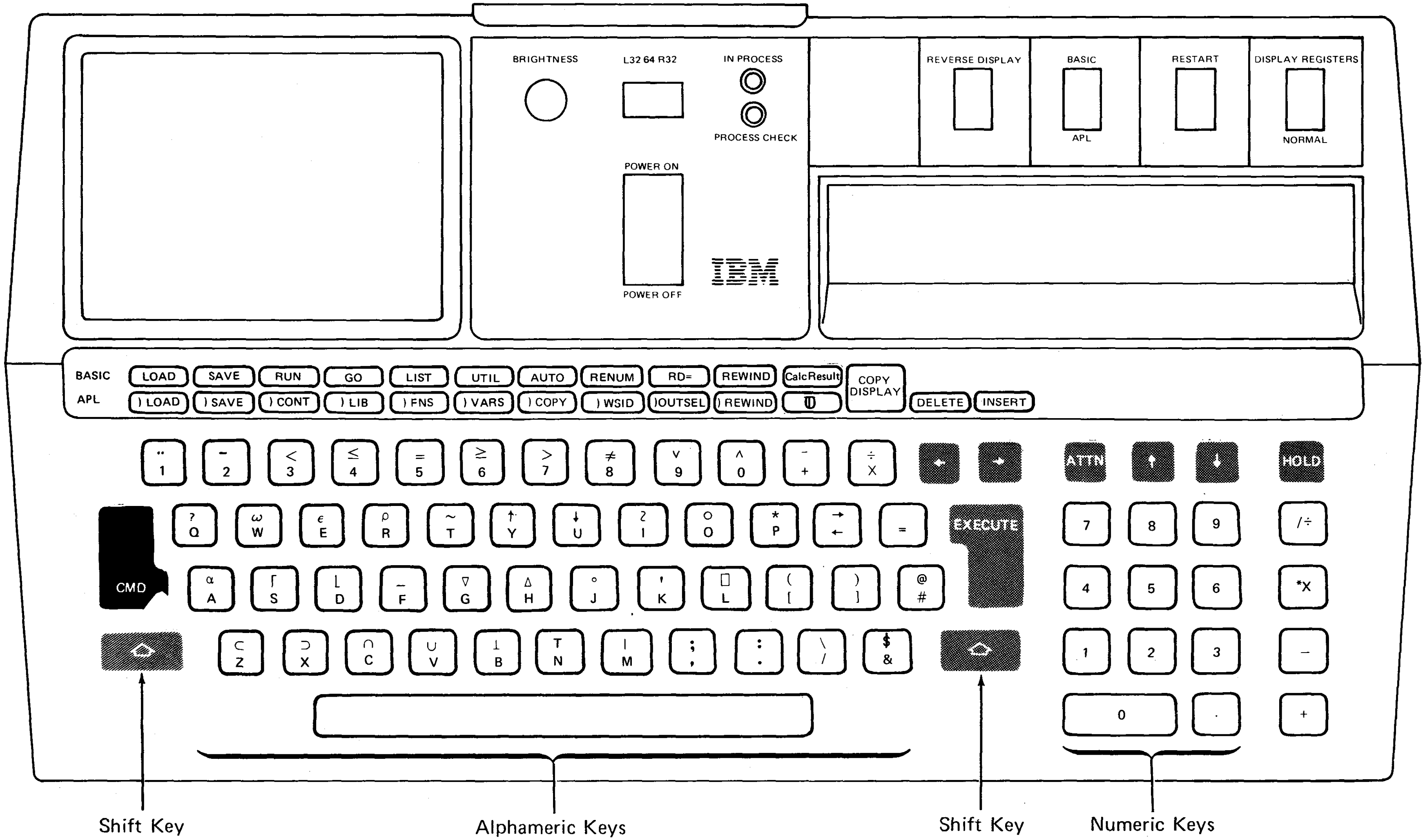


Figure 3. The 5100 Console

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 5100.

Note: When you use your 5100, this switch must be in the NORMAL position.

KEYBOARD

The 5100 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. If either shift key is pressed and held, the upper symbol on the key pressed is entered. The top row of alphameric keys can be used to enter numbers; however, numbers can be conveniently entered using the numeric keys on the right side of the keyboard. The arithmetic symbols (+ - ÷ x) located on the top row of the alphameric keyboard can also be entered using keys to the right of the numeric keys.

The keyboard contains some keys that perform operations in addition to 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 the APL language chapter (Chapter 4) of this manual.

Attention

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

Pressing ATTN during execution of any expression or user-defined function stops system 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 ATTN is pressed twice 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.

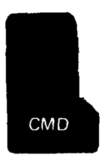
Hold 

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 the display information during an output operation, when the display is changing rapidly. When the hold is in effect (HOLD pressed once), only the COPY DISPLAY key is active.

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), the use of the arithmetic keys (+ - ÷ x) on the right side of the keyboard are restricted to use by service personnel.

Execute 

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

Command 

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. The command keywords are:)LOAD,)SAVE,)CONT,)LIB,)FNS,)VARS,)COPY,)WSID,)OUTSEL, and)REWIND.

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

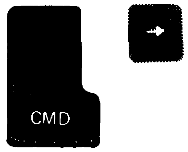
Positioning the Cursor and Information on the Display Screen

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

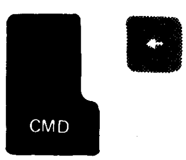
When these keys are both held down, the characters continue to move to the right and blank characters continue to be inserted.

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

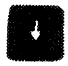
Flashing character

After the delete operation: 123456_

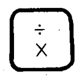
When these keys are both held down, the characters at the cursor position continue to be deleted and all the characters to the right are moved to the left.

Scroll Up 

This key (located above the numeric keys) can be used only in execution mode. When this key 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.

Scroll Down 

This key (located above the numeric keys) can be used only in execution mode. When the key 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.

Copy Display 

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 5100 console (Figure 3) has the following indicator lights:

Process Check

When on, this 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.

In Process

When the system is processing input, generally the display screen is blank and the IN PROCESS light is on. After the input is processed, the light goes off, the output and flashing cursor are displayed, and the system waits for input.

Notes:

1. For some expressions or user-defined functions (see Chapter 5), output is generated before the expression or function has completed execution. In such cases, even though the system is still processing data, the IN PROCESS light goes off and the output is displayed. The flashing cursor is again displayed when the system has finished processing the input (the expression or function has completed execution).
2. If the display screen is blank (no data or cursor is present) and the IN PROCESS light is off, check the brightness control before calling your service representative.

Chapter 2. System Commands

Overview

SYSTEM OVERVIEW

The 5100 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 5100, all the data in the active workspace is lost. However, the contents of the active workspace can be saved on tape (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.

The tape is your library; that is, it is a place where you can store data for later use. Before a tape can be used, it must be formatted. A formatted tape 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.

Commands that Control the Library (Tape)

Command	Meaning
)CONTINUE	Write the contents of the active workspace on tape. The active workspace can contain suspended functions.
)DROP	Drop a tape file.
)MARK	Format the tape.
)SAVE	Write the contents of the active workspace on tape. The active workspace cannot contain suspended functions.

Commands that Provide Information About the System

Command	Meaning
)FNS	Display the names of the user-defined functions.
)LIB	Display workspace file headers.
)SI	Display the state indicator.
)SIV	Display the state indicator and local names.
)SYMBOLS	Display the number of symbols allowed and used in the active workspace.
)VARS	Display the names of the global variables.
)WSID	Display the active workspace ID.

Other Commands that Control the System

Command	Meaning
)MODE	Place the 5100 in communications mode.
)OUTSEL	Select printer output.
)PATCH	Apply IMFs (internal machine fix) to the system or recover data after a tape error.
)REWIND	Rewind the tape.

Notes:
1. Objects refers to both user-defined functions and variables.
2. The system commands)CONTINUE,)COPY,)PCOPY,)DROP,)LOAD,)MARK,)REWIND, and)SAVE will blank the top 8 or 9 lines on the display screen when they are used.

System Overview

Overview

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. System commands cannot be used within APL instructions and cannot be used as part of a function definition (function definition is discussed in Chapter 6).

System commands can be entered two ways:

1. The system command can be entered one character at a time from the keyboard.
2. The system commands)LOAD,)SAVE,)CONT,)LIB,)FNS,)VARS,)COPY,)WSID,)OUTSEL and)REWIND can be entered in one operation by holding the CMD key while pressing the top-row key just below the label of the command you want.

The parameters, if required, must be entered and the EXECUTE key pressed before any operation will take place. Following is an explanation of terms and symbols used as parameters for system commands:

- Device/file number specifies the tape unit and file to be used. The built-in tape unit is tape unit 1 and the auxiliary tape unit is tape unit 2. If the value specified is less than four digits, tape unit 1 is assumed and the value specified represents only the file number. If the value specified is four digits, the rightmost three digits specify the file number and the leftmost digit specifies the tape unit. For example:

Device/File Number	Meaning
1	Tape 1, file 1
02	Tape 1, file 2
2002	Tape 2, file 2

- Workspace ID is any combination of up to 11 alphabetic or numeric characters (with no blanks); however, the first character must be alphabetic. If more than 11 characters are entered, only the first 11 are used.

- Password is any combination of up to eight alphabetic or numeric characters (with no blanks). If more than eight 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.

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 without changing the active workspace. Primarily, this command stores active status, such as suspended functions, so 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 5100 is similar in function and format to the)SAVE command (except as noted below) and is distinguished from the)CONTINUE command on IBM APLSV.

Notes:
1. A clear workspace cannot be written on tape.
2. A workspace with suspended functions can only be written on tape using the)CONTINUE command (it cannot be written to tape using the)SAVE command).
3.)COPY and)PCOPY commands cannot specify stored workspaces that were written on tape using the)CONTINUE command.
4. A stored workspace written to tape using the)CONTINUE command cannot be loaded into a 5100 active workspace that is smaller than the original active workspace.
5. If a stored workspace that was written to tape using the)CONTINUE command is loaded into another 5100 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 tape.
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 using the)CONTINUE command. A subsequent)LOAD allows the user to resume execution if the media is restored to the same condition as when the workspace was stored using)CONTINUE (that is, the tape containing the shared variable cannot be repositioned or placed on a different drive).
8. If a workspace stored with the)CONTINUE command has an open shared variable or a suspended function, the ⎕LX system command will not be executed when the workspace is loaded.
9. Workspaces are stored and loaded into the active workspace faster using the)CONTINUE command than using the)SAVE command.
10. IMFs are not stored by)CONTINUE. If an IMF is required for operation of the stored workspace, it should be reapplied by the)PATCH command (if the IMF is not already in the system) before the workspace is reloaded.

Syntax

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

where:

device/file number (optional) is the number of the tape unit and file on the tape 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 tape, unless the file is marked unused. If the file is marked unused, the active workspace ID and tape 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 eight 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 tape 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.

Syntax

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

where:

device/file number is the number of the tape unit and workspace file the objects are copied from.

workspace ID is the name of the stored workspace on tape.

: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 object(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 tape. When the command is successfully completed, DROPPED device/file number file ID is displayed.

Syntax

)DROP device/file number [file ID]

where:

device/file number is the number of the tape unit and the file on the tape.

file ID (optional) is the name of the stored workspace file to be marked unused. If the file number specified is a data file, any file ID specified is ignored.

The)ERASE Command

The)ERASE command erases the named global objects from the active workspace. There is no message displayed at the successful completion of the command.

Notes:
1. When a pendent function (see Chapter 7) is erased, the response SI DAMAGE is issued.
2. If the object being erased is a shared variable (see Chapter 8), 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).

)LIB

Syntax

)ERASE object name(s)

where:

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

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)LIB Command

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

- File number. The files on tape are numbered sequentially, starting with 1.
- 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 2-digit code; the following chart gives the meaning of each code:

File Type	Description
00	Unused file
01	Exchange data file (see Chapter 8)
02	General exchange data file (see Chapter 8)
03	BASIC source file

File Type	Description
04	BASIC workspace file
05	BASIC keys file
06	APL continued file (see)CONTINUE command in this chapter)
07	APL saved file (see)SAVE command in this chapter)
08	APL internal data format file (see Chapter 8)
16	Patch, tape recovery, and tape copy file
17	Diagnostic file
19	IMF file
72	Storage dump file

- 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.
- 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.

Following is an example of a file header:

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

- 07 — File type
- 010 — Size of the file
- 001 — Available storage
- 0 — Number of defective records
- FILE6 — File ID
- 006 — File number

The)LIB command operation can be interrupted by pressing the ATTN key.

Syntax

)LIB [device/file number]

where:

device/file number (optional) is the number of the tape unit and the starting file number. All file headers from that file to the end of the tape are displayed. If no entry is made, the display begins with the first file following the file you are currently positioned at on tape unit 1. For tape unit 2, the entry 2000 will display the file headers beginning with the first file following the file you are currently positioned at on tape unit 2.

The)LOAD Command

The)LOAD command loads the contents of a stored workspace from the tape 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.

Note: If the ATTN key is pressed during a load operation, the system operation is interrupted and the active workspace is cleared.

Syntax

)LOAD device/file number workspace ID :password

where:

device/file number is the number of the tape unit and the number of the file on the tape.

workspace ID is the name of the stored workspace.

: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 tape 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 using 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 was issued, the specified file already exists on the tape. 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.

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 what size a file should 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 tape storage (number of 1024-byte blocks) that would be required to write the contents of the active workspace to tape.
- 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 tape.

The formula for a data file (data written to tape using an APL shared variable —see Chapter 8) when all of the data is contained in the active workspace is MAXSIZE= ⌈ (WITHOUT-WITH)÷1024, where:

- MAXSIZE is the maximum amount of tape storage (number of 1024-byte blocks) required to write the data to tape.
- 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 tape 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 tape as it is entered from the keyboard. The amount of tape 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 9.

Note: The file header for each marked 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 tape unit that contains the tape to be formatted. An entry of 1 specifies tape unit 1 and 2 specifies tape unit 2. If no entry is made, tape unit 1 is assumed.

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
```

The)MODE Command

The)MODE command is used to load the 5100 communications program or 5100 serial I/O adapter program from a tape mounted in tape drive 1. When the system is in communications mode, APL is no longer available. For more information on the communications feature or the serial I/O adapter feature, see *IBM 5100 Communications Reference Manual,* SA21-9215, or *IBM 5100 Serial I/O Adapter User's Manual,* SA21-9239, respectively.

Syntax

)MODE COM

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)PATCH Command

The following is a list of the uses of this command. This command is used in conjunction with specially devised programs on the customer support cartridge supplied with the 5100. The uses are described in detail, following the list:

- Copy IMFs (internal machine fix), the Copy IMF program, and the Load IMF program onto another tape cartridge.
- Load IMFs for the system program into the active workspace, then make the APL language available again.
- Display the EC version of each interpreter module.
- Recover data on tape when tape read errors (ERROR 007 ddd—see Chapter 11) occur during use of one of the following files:
  1. Exchange (file type 01)
  2. General exchange (file type 02)
  3. BASIC source (file type 03)
  4. APL internal data format (file type 08)
- Copy the contents of one tape cartridge to another tape cartridge.

The customer support cartridge contains the following files:

- File 1. The programs that copy or load IMFs and the program that displays interpreter module EC versions.
- File 2. The IMFs for the 5100.
- File 3. The Tape Recovery program.
- File 4. The Tape Copy program.
- File 5. APL aids. This is a saved workspace file (WSID=APLAIDS) that contains the following four functions:
  1. ∆∆TRACE—Traces all the statements in a specified user-defined function.
  2. ∆∆TRACEALL—Traces the first executable statement of each user-defined function currently in the active workspace.
  3. ∆∆TRACEOFF—Turns off all tracing.
  4. ∆∆SHARED—Displays the shared variable names currently in the active workspace.

The ∆∆TRACE function requires as its right argument the name of the user-defined function to be traced enclosed in single quotes. The other functions do not require any arguments.

This workspace file also contains the following five variables that describe the functions in the workspace:

1. DESCRIBE
2. DESCRIBE∆∆TRACE
3. DESCRIBE∆∆TRACEALL
4. DESCRIBE∆∆TRACEOFF
5. DESCRIBE∆∆SHARED

These functions and variables can be copied into the active workspace using the)COPY command. For example, to copy the ∆∆TRACE function into the active workspace:

```
)COPY 5 APLAIDS ∆∆TRACE
```

Note: The)PATCH command is not required for using the functions in file 5.

When the)PATCH command is used with the tape cartridge inserted in tape drive 1, the following options are displayed:

```
ENTER OPTION NO.
1.    COPY IMF TAPE
2.    LOAD IMF'S
3.    DISP EC VER.
4.    KEY-ENTER IMF
5.    END OF JOB
6.    TAPE RECOVERY
7.    TAPE COPY PGM
_  <---------------------- Flashing Cursor
```

To select an option, enter an option number (the tape cartridge must be inserted in tape drive 1). If an option number other than those displayed is entered, the options will be displayed again. Once the option number has been entered, additional prompting messages might be displayed for the selected option.

)PATCH

Option 1. Copy IMF Tape

The Copy IMF Tape option allows the following files to be copied from the tape:

- File 1, which contains the Copy IMF program, Load IMF program, and Display EC Version program.
- File 2, which contains the IMFs for the 5100. The IMFs can be copied from the file as follows:

  1. Copy all IMFs that apply to APL.
  2. Copy all IMFs for APL that apply to the 5100 being used.
  3. Copy specific IMFs by problem number.
  4. Copy specified IMFs by problem numbers that apply to the 5100 being used. (If a problem number is specified that does not apply to the 5100 being used, it is not copied.)

Note: The tape onto which files 1 and 2 are to be copied must be marked before the copy operation is done. Use the)LIB command to determine what size the files should be marked.

The Copy IMF Tape program will issue prompting messages and wait for the user to respond to each message.

Copying IMFs allows tape cartridges containing only the IMFs that apply to your 5100 to be created.

Option 2. Load IMFs

The Load IMFs option allows IMFs to be loaded into the system program and then makes the APL language available again. IMFs can be loaded as follows:

- Load all IMFs that apply to the 5100 being used.
- Load specified IMFs by problem numbers that apply to the 5100 being used. (If a problem number is specified that does not apply to the 5100 being used, it is not loaded.)

The Load IMFs program will issue prompting messages and wait for the user to respond to each message.

Note: The IMFs occupy storage (space) in the active workspace and can also reduce the performance of your 5100 significantly; therefore, IMFs should not be applied to your 5100 if the problem does not affect your operation or if the problem can be circumvented by an APL statement or command. The IMFs will remain in the active workspace until the power is turned off or RESTART is pressed.

Option 3. Disp EC Ver.

The Disp EC Ver. option is primarily for your service representative's use. This option will display a 4-digit code for each interpreter module. The first two digits are the module identification and the next two digits are the EC version.

The EC Version program will issue prompting messages and wait for the user to respond to each message.

Option 4. Key-Enter IMF

This option allows the service representative to enter IMFs from the keyboard. The IMF is then written to file 2 on the tape containing the IMFs. The IMF can then be loaded or copied from the tape.

Option 5. End of Job

This option causes the APL language to be available again.

Option 6. Tape Recovery

The Tape Recovery option allows the user to recover data from a file or files on which tape read errors (ERROR 007 ddd) are occurring. The Tape Recovery Program can be used on the following files:

- Exchange (file type 01)
- General exchange (file type 02)
- BASIC source (file type 03)
- APL internal data format (file type 08)

The Tape Recovery program will issue prompting messages and wait for the user to respond to each message.

The Tape Recovery program will recover as much data as possible in the file; some of the data in the record where the tape read errors occur is not recoverable; some of the data that precedes and follows that record may also not be recoverable.

Option 7. Tape Copy Program

The Tape Copy option allows you to copy the contents (up to the end of marked tape) of one cartridge to another cartridge. Tape copy can utilize the auxiliary tape drive, if available. Tape copy also marks the tape being copied to.

Tape copy issues prompts and waits for you to respond to each prompt.

Syntax

)PATCH

There are no parameters.

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 tape 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 is the number of the tape unit and the stored workspace file.

workspace ID is the name of the stored workspace on the tape.

: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).

)REWIND
)SAVE

The)REWIND Command

The)REWIND command rewinds the specified tape. There is no message 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 the parameter is omitted, tape 1 is rewound.

The)SAVE Command

The)SAVE command stores the contents of the active workspace onto tape without changing the contents of the active workspace. The stored workspace can be loaded or copied on a machine with a larger or a smaller active workspace. Also, individual global objects can be copied from the stored workspace to the active workspace. When this command is successfully completed, SAVED device/file number workspace ID is displayed. Do not remove the tape until this message is displayed.

Notes:

1. A clear workspace or a workspace with suspended function cannot be written on tape using the)SAVE command; however, a workspace with suspended functions can be written to tape using the)CONTINUE command.
2. The)COPY and)PCOPY commands can specify stored workspaces that were written on tape 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 tape using the)SAVE command can be loaded into another 5100 with a smaller active workspace.
4. If ATTN is pressed during a)SAVE operation, the system operation is interrupted and the file is set to unused.
5. No open shared variables are stored in a)SAVE operation. Open shared variables are stored with the)CONTINUE command.
6. IMFs are not stored by the)SAVE operation. If an IMF is required, it is necessary to reload the IMF by using the)PATCH command (if the IMF is not already in the system) before the stored workspace is reloaded.

Syntax

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

where:

device/file number (optional) is the number of the tape unit and file on the tape 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 tape unless the file is marked unused. If the file is marked unused, the active workspace and tape 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 eight 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 (see *State Indicator* in Chapter 7). 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)SIV Command

The)SIV command displays the names of the suspended and pendent user-defined functions (see *State Indicator* in Chapter 7) 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

)SIV

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 5100. 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 tape.

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 eight 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 writing the workspace to tape with a)SAVE or)CONTINUE command and reloading 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.

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)WSID Command

The)WSID (workspace ID) command is used to change or display the tape device/file number and workspace ID for the file where the active workspace contents will be written if either a)SAVE or a)CONTINUE command is used. The)WSID command is also used to change or assign the security password. 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 only affects the active workspace; it cannot be used to change any information on tape.

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 eight 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 3. Data

VARIABLES

You can store data in the 5100 by assigning it to a variable name. These stored items are called variables. Whenever the variable name is used, APL supplies the data associated with that name. A variable name can be up to 77 characters in length with no blanks; the first character must be alphabetic and the remaining characters can be any combination of alphabetic and numeric characters. Variable names longer than 77 characters can be used, but only the first 77 characters are significant to APL. The ← (assignment arrow) is used to assign data to a variable:

```
LENGTH←6
WIDTH←8
AREA←LENGTH×WIDTH
```

To display the value of a variable, enter just the variable name:

```
      LENGTH
6
      WIDTH
8
      AREA
48
```

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 ten. 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 5100 can range from ¯7.237005577332262E75 to 7.237005577332262E75. The smallest numeric value the 5100 can use is ±5.397604346934028E¯79.

Numeric Value Precision

Numbers in the 5100 are carried internally with a precision of 16 significant digits.

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:
```

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 ($> \geq = < \leq \neq$) 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 ($\wedge\tilde{\wedge}\vee\tilde{\vee}\sim$) 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.

Scalar
Arrays

SCALAR

A single item, whether a single number or single character constant, is called a *scalar.* It has no coordinates; that is, it can be thought of as a geometric point. The following are examples of scalars:

```
      132.3
132.3
      'A'
A
```

Scalars can be used directly in calculations or can be assigned to a variable name. The variable name for the scalar can then be used in the calculations:

```
      2×3
6
      A←2
      B←3
      A+B
5
```

ARRAYS

Array is the general term for a collection of data, and includes scalars (single data items), vectors (strings of data), matrices (tables of data), and arrays of higher dimensions (multiple tables). All primitive (built-in) functions are designed to handle arrays. Some functions are designed specifically to handle arrays rather than scalars. Indexing, for example, can select certain elements from an array for processing.

One of the simplest kinds of arrays, the vector, has only one dimension; it can be thought of as a collection of elements arranged along a horizontal line. The numbers that indicate the positions of elements in an array are called indices. An element can be selected from a vector by a single index, since a vector has only one dimension. The following example shows assigning a numeric and a character vector to two variable names, N and C; the names are then entered to display the values they represent:

```
      N←5 6.2 ¯3 888 95.12
      N
5 6.2 ¯3 888 95.12
      C←'ABCDEFG'
      C
ABCDEFG
```

Generating Arrays

The most common way to generate an array is to specify the following: the shape the array is to have—that is, the length of each coordinate; the values of the elements of the new array. The APL function that forms an array is the reshape function. The symbol for the reshape function is ρ. The format of the function used to generate an array is XρY, where X is the shape of the array and Y represents the values for the elements of the array. For the left argument (X), you enter a number for each coordinate to be generated; this number indicates the length of the coordinate. Each number in the left argument must be separated by at least one blank. The values of the elements of the new array are whatever you enter as the right argument (Y). The instruction 7ρA means that the array to be generated has one dimension (is a vector) seven elements in length, and that seven values are to be supplied from whatever values are found stored under the name A. It does not matter how many elements A has, as long as it has at least one element. If A has fewer than seven elements, its elements are repeated as often as needed to provide seven entries in the new vector. If A has more than seven elements, the first seven are used. The following examples show generation of some vectors:

```
      7ρ1 2 3
1 2 3 1 2 3 1
      2ρ123
123 123
      5ρ1.3
1.3 1.3 1.3 1.3 1.3
```

An array with two coordinates (rows and columns) is called a matrix.

$$\begin{array}{c} \overbrace{\hphantom{1\quad 2\quad 3\quad 4\quad}}^{\text{Columns}} \\ \left.\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 9 \\ 9 & 10 & 11 & 12 \end{array}\right\}\text{Rows} \end{array}$$

To generate a matrix, you specify X (left argument) as two numbers, which are the lengths of the two coordinates. The first number in X is the length of the first coordinate, or number of rows, and the second number is the length of the second coordinate, or number of columns. The following example shows how a matrix is generated:

```
      M←2 3ρ1 2 3 4 5 6
      M
1 2 3
4 5 6
      M←2 4ρ'ABCDEFGH'
      M
ABCD
EFGH
      M1←2 3ρM
      M1
ABC
DEF
```

Note that the values in the right argument are arranged in row order in the arrays. If the right argument has more than one row, the elements are taken from the right argument in row order.

The rank of an array is the number of coordinates it has, or the number of indices required to locate any element within that array. Scalars are rank 0. Vectors have a rank of 1, matrices have a rank of 2, and N-rank arrays have a rank from 3 to 63 (where N is equal to the rank). N-rank arrays, like matrices, are generated by providing as the left argument a number indicating the length for each coordinate (planes, rows, and columns). The following examples show how to generate 3-rank arrays. Note that the elements taken from the right argument are arranged in row order:

```
      A←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      2 3 4ρA
ABCD
EFGH            2-plane, 3-row, 4-column array
IJKL

MNOP
QRST
UVWX
      4 3 2ρA
AB
CD              4-plane, 3-row, 2-column array
EF

GH
IJ
KL

MN
OP
QR

ST
UV
WX
```

Finding the Shape of An Array

Once you have generated an array, you can find its shape (number of elements in each coordinate) by specifying ρ (shape function) with only a right argument which is the name of the array. If A is a vector with six elements and you enter ρA, the result is one number because A is a one-dimensional array. The number is 6, the length (number of elements) of A's one dimension. The result of the shape function is always a vector:

```
      A←111 222 333 444 555 666
      ρA
6
```

The shape of a matrix or N-rank array is found the same way:

```
      M←2 3ρ1 2 3 4 5 6
      M
 1 2 3
 4 5 6
      ρM
2 3
      R←2 3 4ρ1 2 3 4 5 6 7 8
      R
 1 2 3 4
 5 6 7 8
 1 2 3 4

 5 6 7 8
 1 2 3 4
 5 6 7 8
      ρR
2 3 4
```

In some cases, it might be necessary to know just the rank, the number of coordinates (or indices) of an array. The rank can be found by entering ρρ (shape of the shape) and a right argument, which is the name of the array:

```
      A←111 222 333 444 555 666
      B←2 3ρ1 2 3 4 5 6
      C←2 3 4ρ1 2 3 4 5 6 7
      ρA
6
      ρρA
1
      ρB
2 3
      ρρB
2
      ρC
2 3 4
      ρρC
3
```

Array Shape

The following table shows what the shapes and ranks are for the various types of arrays:

Data Type	**Shape** ρX	**Rank** ρρX
Scalar	No dimension (indicated by an empty vector).	0
Vector	Number of elements.	1
Matrix	Number of rows and the number of columns.	2
N-rank arrays	Each number is the length of a coordinate.	N

Empty Arrays

Empty Arrays

Although most arrays have one or more elements, arrays with no elements also exist. An array with no elements is called an *empty array.* Empty arrays are useful when creating lists (see *Catenation* in this chapter) or when branching in a user-defined function (see Chapter 6).

Following are some ways to generate empty arrays:

- Assign ⍳0 to a variable name to generate an empty vector:

```
      EVECTOR←⍳0
      EVECTOR

      ⍴EVECTOR
0
```

An empty array is indicated by a blank display.

The shape of the empty vector is zero (zero elements).

- Use a zero length coordinate when generating a multidimensional array:

```
      EMATRIX1←3 0⍴⍳0
      EMATRIX1

      ⍴EMATRIX1
3 0
```

This matrix has three rows and no (0) columns.

A blank output display

- A function might generate an empty vector as its result; for example, finding the shape of a scalar:

```
      ⍴'A'

```

A blank output display.

CATENATION

You can join together two arrays to make a single array by using the catenation function. The symbol for this function is the comma. When catenating vectors, or scalars and vectors, the variables are joined in the order in which they are specified, as the following examples show:

```
      A←1 2 3 4
      B←4 5 6
      A,B
1 2 3 4 4 5 6
      B,A
4 5 6 1 2 3 4
      A,2
1 2 3 4 2
      3,A
3 1 2 3 4
```

When catenating two matrices or N-rank arrays, the function can take the form A,[I]B, where I defines the coordinate that will be expanded when A and B are joined. If the coordinate is not specified, the last coordinate is used. When A and B are matrices and [I] is [1], the first coordinate (number of rows) is expanded; when [I] is [2], the last coordinate (number of columns) is expanded. The following examples show how to catenate matrices:

Graphic Representation

```
      A←2 3⍴10 20 30 40 50 60
      B←2 3⍴11 22 33 44 55 66
```

A

10	20	30
40	50	60

B

11	22	33
44	55	66

```
      A,B
10 20 30 11 22 33
40 50 60 44 55 66
      A,[2]B
10 20 30 11 22 33
40 50 60 44 55 66
```

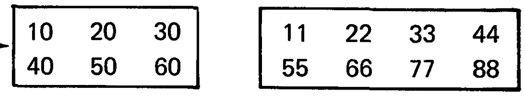
A			B		
10	20	30	11	22	33
40	50	60	44	55	66

```
      A,[1]B
10 20 30
40 50 60
11 22 33
44 55 66
```

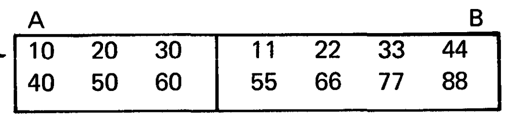
A	10	20	30
	40	50	60
B	11	22	33
	44	55	66

Matrices of unequal sizes can be catenated, providing that the lengths of the coordinates not specified are the same (see the first example following). If the coordinates not specified are not the same, an error results (see the second example following):

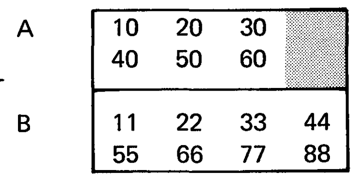
```
    A←2 3⍴10 20 30 40 50 60
    B←2 4⍴11 22 33 44 55 66 77 88
```




```
      A,[2]B
10 20 30 11 22 33 44
40 50 60 55 66 77 88
```




```
      A,[1]B
LENGTH ERROR
      A,[1] B
      ^
```

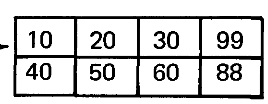


A scalar can also be catenated to an array. In the following example, a scalar is catenated to a matrix. Notice that the scalar is repeated to complete the coordinate:

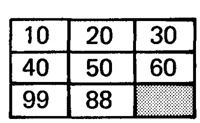
```
      A←2 3⍴10 20 30 40 50 60
      A
10 20 30
40 50 60
      A,[2]99
10 20 30 99
40 50 60 99
      A,[1]99
10 20 30
40 50 60
99 99 99
```

A vector can also be catenated to another array, provided the length of the vector matches the length of the coordinate *not* specified. See the following examples:

```
      A,99 88
10 20 30 99
40 50 60 88
```




```
      A,[1]99 88
LENGTH ERROR
      A,[1] 99 88
      ^
```



The catenate function is useful when creating lists of information. Sometimes it is necessary to use an empty array to start a list. For example, suppose you want to create a matrix named PHONE where each row will represent a 7-digit telephone number. First you want to establish the matrix, then add the telephone numbers at a later time. The following instruction will establish an empty array named PHONE with no (0) rows and seven columns:

```
      PHONE←0 7⍴⍳0
      PHONE
                                    Blank display indicates an
                                    empty array.
      ⍴PHONE
0 7
```

Now, the telephone numbers can be added as follows:

```
      PHONE←PHONE,[1]'5336686'
      PHONE
5336686
      PHONE←PHONE,[1]'4564771'
      PHONE
5336686
4564771
      ⍴PHONE
2 7
                                    The list of telephone numbers
                                    now contains two rows.
```

INDEXING

You may not want to refer to the whole array but just to certain elements. Referring to only certain elements is called indexing. Index numbers must be integers; they are enclosed in brackets and written after the name of the variable to which they apply. Assume that A is assigned a vector as follows: A←11 12 13 14 15 16 17. The result of entering A is the whole vector, and the result of entering A[2] is 12 (assuming the index origin is 1; see Chapter 5 for more information on the index origin).

Here are some more examples of indexing:

```
      A←11 12 13 14 15 16 17
      A[3]
13
      A[5 3 7 1]
15 13 17 11
      B←3 1 4 6
      A[B]                          Blank Character
13 11 14 16
      B←'ABCDEFGHIJKLMNOPQRSTUVWXYZ '
      B[4 1 14 27 1 14 4 27 3 12 1 9 18]
DAN AND CLAIR
      C←22 9 18 7 9 14 9 1
      B[C]
VIRGINIA
```

If you use an index that refers to an element that does not exist in the array, the instruction cannot be executed and INDEX ERROR results:

```
      A
11 12 13 14 15 16 17


      A[8]
INDEX ERROR
      A[8]
      ^
```

You cannot index or do anything else with an array until after the array has been specified. For example, suppose that no value has been assigned to the name Z; then an attempt to store values in certain elements within Z would result in an error, since those elements do not exist:

```
      Z[3 4]←18 46
VALUE ERROR
      Z[3 4]← 18 46
      ^
```

Indices (whatever is inside the brackets) can be expressions, provided that when those expressions are finally evaluated, the results are values that represent valid indices for the array:

```
      B
ABCDEFGHIJKLMNOPQRSTUVWXYZ
      X←1 2 3 4 5
      B[X×2]
BDFHJ
      X
1 2 3 4 5
      B[1+X×3]
DGJMP
```

The array from which elements are selected does not have to be a variable. For example, a vector can be indexed as follows:

```
      2 3 5 7 9 11 13 15 17 19[7 2 4 2]
13 3 7 3
      'ABCDEFGHIJKLMNOPQRSTUVWXYZ '[12 15 15 11 27 16 1]
LOOK PA

      'ABCDEFGHIJKLMNOPQRSTUVWXYZ '[2 4⍴4 15 14 27 13 1 18 25]
DON
MARY
```

The shape of the result is the same as the index.

Indexing a matrix or N-rank array requires an index number for each coordinate. The index numbers for each coordinate are separated by semicolons. Suppose M is a 3 by 4 matrix of consecutive integers:

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

If you ask to see the values of M, they are displayed in the usual matrix form:

```
      M
 1  2  3  4
 5  6  7  8
 9 10 11 12
```

If you want to refer to the element in row 2, column 3, you would enter:

```
      M[2;3]
7
```

If you want to refer to the third and fourth elements in that row, you would enter:

```
      M[2;3 4]
7 8
```

Similarly, to refer to the elements in column 4, rows 1, 2, and 1, you would enter:

```
      M[1 2 1;4]
4 8 4
```

You can use the same procedure to select a matrix within a matrix. If you want the matrix of those elements in rows 2 and.3 and columns 1, 2, and 1 of M, you would enter:

```
      M[2 3;1 2 1]
 5  6  5
 9 10  9
```

If you do not specify the index number for one or more of the coordinates of the array that you are indexing, APL assumes that you want the entire coordinate(s). For instance, to get all of row 2, you would enter:

```
      M[2;]
5 6 7 8
```

Or to get all of columns 4 and 1, you would enter:

```
      M[;4 1]
 4  1
 8  5
12  9
```

Note: You still have to enter the semicolon to make clear which coordinate is which. The number of semicolons required is the rank of the array minus one. If the correct number of semicolons is not specified, RANK ERROR results:

```
      M←3 4⍴⍳12
      ⍴M
3 4
      M[6]←9
RANK ERROR
      M[6]←9
      ^
```

You can change elements within an array by assigning new values for the indexed elements. (The rest of the array remains unchanged.)

```
      A←3 3⍴1 2 3 4 5 6 7 8 9
      A
1 2 3
4 5 6
7 8 9
      A[2;2 3]←10 20
      A
 1  2  3
 4 10 20
 7  8  9
```

APL functions are of two types: user-defined and those that are built into the APL language. 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 (see Chapter 3) 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
```

Conjugate Plus

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 3ρ1 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 3ρ¯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 x Function: Signum, Times

Monadic (One-Argument) Form: Signum xB

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 AxB

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
```

Signum

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 2⍴1 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 2ρ1 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).

Ceiling

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
```

Magnitude Residue

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 scalar, 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

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 = $1/2^3$ = 1/8 = .125

The root of a number can be found by raising 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 2⍴1 2 3 4
      B
1 2
3 4
      B*2
     1              4
     9             16
```

The ⍟ Function: Natural Log, Logarithm

The ⍟symbol is formed 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 non-negative 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
```

Natural Log
Logarithm

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 $\boxed{○}$

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:

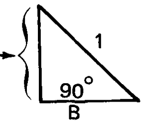
```
      B←2 2⍴1 2 3 4
      B
1 2
3 4
      ○B
   3.1416     6.2832
   9.4248    12.566
```

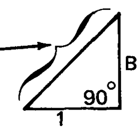
Dyadic (Two-Argument) Form: Circular A○ B

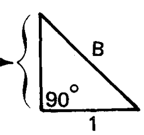
The *circular* function result is the value of the specified trigonometric function (argument A) for the specified radians (argument B). The arguments can be numeric scalars, vectors, or other arrays. Arguments must be the same shape, unless one is a scalar or single-element array. Arguments of the same 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:

Pi Times
Circular

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
0.7854          The left argument specifies
                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 2⍴1 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

The ! symbol is formed 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:

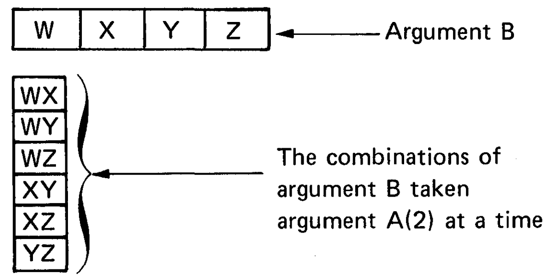
```
        B←2 2⍴0 1 2 3
        B
0 1
2 3
        !B
    1           1
    2           6
```

Factorial
Binomial

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 argument 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 ?

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
 2 17 16
24 13  4
```

Dyadic (Two-Argument) Form

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

Roll

The ∧ Function: And

```
∧
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 v Function: Or [v 9]

Monadic (One-Argument) Form

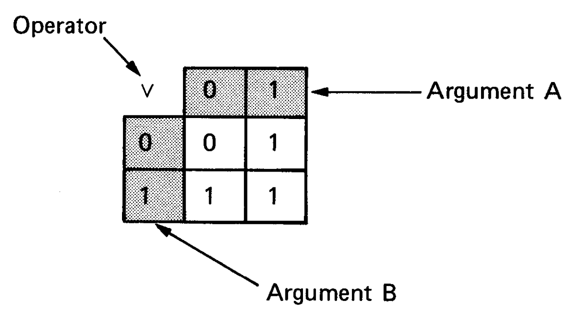
There is no monadic form.

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:

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

Or Table



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]

The ⍲ symbol is formed 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 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

The ⍱ symbol is formed 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 → top row ; Argument B → left column

⍱	0	1
0	1	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:

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

The > Function: Greater Than [> 7]

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 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3>B
1 1 0
0 0 0
```

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

Greater Than

The = Function: Equal To

$\boxed{= \atop 5}$

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 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
```

Note: 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 < Function: Less Than

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
```

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

Less Than

The ≥ Function: Greater Than or Equal To

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 3⍴1 2 3 4 5 6
      B
1 2 3
4 5 6
      3≥B
1 1 1
0 0 0
```

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

The ≤ Function: Less Than or Equal To ≤ 4

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
```

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

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
```

Note: 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

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 ⌽[I]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).

Mixed Functions

Dyadic Mixed Functions	Name	Result
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, without selecting the same number twice.
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 *Operators* later in this chapter) *reduction* and *scan* can be applied to a specific coordinate of an array. This is done by using an index entry [I] which 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 3-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.

Shape
Reshape

The ρ Function: Shape, Reshape (Structure) ρ R

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, since 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) AρB

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 3ρ1 2 3 4 5 6
 1 2 3
 4 5 6
      4 2ρ'ABCDEFGH'
AB
CD
EF
GH
      5ρ'MOUSETRAP'
MOUSE
      3 4ρ123
 123 123 123 123
 123 123 123 123
 123 123 123 123
      A←4 2ρ1 2 3 4 5 6 7 8
      A
 1 2
 3 4
 5 6
 7 8
      2 3ρA
 1 2 3
 4 5 6
      6ρ'..'
............
```

The , Function: Ravel, Catenate, Laminate

Monadic (One-Argument) Form: Ravel ,B

The *ravel* function results in a vector containing the elements 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 2ρ1 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 3ρ'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.

Ravel

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 3ρ1 2 3 4 5 6
      B←2 3ρ7 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
```

Laminate

The / Function: Compress

Monadic (One-Argument) Forn

See Reduction 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;
 1  2  3  4                       the first and third rows, as specified by
 9 10 11 12                       argument A, are 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 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
 1 2 3
 0 0 0
 4 5 6
      1 1 0 1\[2]B
 1 2 0 3
 4 5 0 6
      1 0 1⍀B
 1 2 3
 0 0 0
 4 5 6
```

The first coordinate (rows) is expanded; a row is inserted between the first and second row.

The second coordinate (columns) is expanded; a column is inserted between the second and third columns.

The ⍋ Function: Grade Up

The ⍋symbol is formed by overstriking the ∆ 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

The ⍒ symbol is formed by overstriking the ∇symbol and the | symbol.

Monadic (One-Argument) Form: Grade Down ⍒

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 using the grade up and grade down functions:

```
      A←5 2 8 7 3 4 10 1 2 3
      A
5 2 8 7 3 4 10 1 2 3 ←—— Positions 2 and 9 and 5 and 10 are equal.
      ⍒A
7 3 4 1 6 5 10 2 9 8
      ⍋A
8 2 9 5 10 6 1 4 3 7
```

Because the indices for the equal elements are in the same order (first occurrence first) for both the grade down and grade up function, 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 [↑ Y]

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 4ρ1 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 3ρ1 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

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 argument 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
       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

○ O | M

The ⌽ symbol is formed by overstriking the ○ symbol and the | symbol. A special form of the function symbol is ⊖, formed 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.

Reverse Rotate

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

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

SAVE
MUCH
```

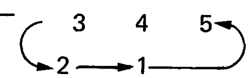
Dyadic (Two-Argument) Form: Rotate A⌽[I] B or A⌽B or A⊖B

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.

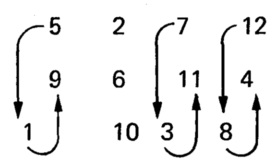
When argument B is a multidimensional array, the index entry [I] can be used to specify the coordinate that is acted on. If the index entry is omitted, the last coordinate (column) is acted on. If the A⊖B form is used, then the first coordinate is acted on.

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 4ρ 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
      1 0 1 2⊖B
 5  2  7 12
 9  6 11  4
 1 10  3  8
      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.

The first coordinate is acted on.

Reverse Rotate

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.

Argument A

1	0	0
0	2	0
0	0	0

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

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

The ⍉ symbol is formed 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
 10 22

  3 15                 The coordinates are reversed.
  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 1 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 3-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
                 An array with two planes, three rows, and four columns.
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         The second and third coordinates have been interchanged,
                 forming an array with two planes, four rows, and three
13 17 21         columns.
14 18 22
15 19 23
16 20 24
```

Transpose

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
```

Deal

The ⊥ Function: Decode (Base Value) ⊥ B

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 1776 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 equals 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	The units position always represents itself.
7 x10 =	70	The value in the next position is multiplied by the rightmost value in argument A.
7 x10x10 =	700	The value in the next position is multiplied by the two rightmost values in argument A, and so on.
1 x10x10x10 =	1000	
	1776	

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 calculating the result. 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:

```
      10 ⊥ 3 2 5
325
      10 10 10 ⊥ 7
777
```

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
      10 10 ⊥ B
17 59 24
```

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 0 10 5
      A
10 10 10                                    3 =     3
 0 10  5                                    2 x5 =  10
      A⊥1 2 3                               1 x10x5 = 50
123 63 ←————————————————————————————————————————— 63
      A⊥2 3 4
234 119
```

Decode

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⍴10 10 10 20 10 5
      B←3 2⍴1 2 2 4 3 3
      A
10 10 10
20 10  5
      B
1 2
2 4
3 3
      A⊥B
123 243
 63 123
```


The following examples convert hours, minutes, and second 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) equals one unit of the next position to the left.

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

24 units (hours) equals 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 Form

There is no monadic form.

Dyadic Form: Encode A⊤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 equals one unit of the next position to the left.

Argument B has this value:

1776

Encode

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:

```
         10 10 ⊤17 59 24
 1 5 2
 7 9 4
         10 5 ⊤ 17 36 24
 3 7 4
 2 1 4
```

```
   3
5 ┌17
   15
    2
```

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 2ρ10 20 10 10 10 5
      A
10 20
10 10
10  5
      B←123
      A⊤B
1 2
2 4
3 3
      ⍉A⊤B
1 2 3
2 4 3
```

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

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←10 10 10
      B←2 2ρ123 456 789 000
      A
10 10 10
      B
 123 456
 789   0
      A⊤B
1 4
7 0

2 5
8 0

3 6
9 0
```

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 2ρ10 20
      A
10 20
10 20
      B←2 2ρ 99 88 77 66
      B
99 88
77 66
      A⊤B
 9  8
 7  6

 4  4     Result of the number system in
 3  3     column 1 of argument A

 9  8     Result of the number system in
 7  6     column 2 of argument A

19  8
17  6
```

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 (seconds): 60 units (seconds) equals one unit of the next position to the left.
- 60 (minutes): 60 units (minutes) equals one unit of the next position to the left.
- 24 (hours): 24 units (hours) equals one unit of the next position to the left.

Argument B

5415 (seconds)

The result was obtained as follows:

$$60 \times 60 = 3600 \quad \begin{array}{r} \boxed{1} \\ \overline{)5415} \\ \underline{3600} \\ 1815 \end{array}$$

$$60 \quad \begin{array}{r} \boxed{30} \\ \overline{)1815} \\ \underline{1800} \\ \boxed{15} \end{array}$$

The ∊ Function: Membership [∊ E]

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
      '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

The ⌹ symbol is formed by overstriking the ⎕ and the ÷ symbols.

Monadic (One-Argument) Form: Matrix Inverse ⌹

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 2ρ1 2 3 ¯1
      A⌹B
2 3
      B←2 2ρ3 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 2ρ26 16 9 6
      B←2 2ρ3 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 3ρ11 14 ¯2 4 7 ¯1
      B←3 2ρ3 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

The ⍎ symbol is formed 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
3
```

The character vector in the variable A is executed.

```
      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)/'A+B'
4
```

A + B is executed only when A equals B.

Dyadic (Two-Argument) Form

There is no dyadic form.

The ⍕ Function: Format [T/N] [∘/J]

The ⍕ symbol is formed by overstriking the ⊤ and the ∘ symbols.

Monadic (One-Argument) Forms: 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 5100 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.

Each column of an array can be formatted 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. If the format function is used with a shared variable during input operations (see Chapter 8), alternate records are skipped if the input data is already in character form.

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: ⓕ/[I] B or ⓕ/B or ⓕ⌿B, where ⓕ can be any primitive dyadic scalar function that is applied between each of 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 each of 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. The identity elements are listed in the following table:

Identity Element Table

Dyadic Function		**Identity Element**
Times	×	1
Plus	+	0
Divide	÷	1
Minus	-	0
Power	⋆	1
Logarithm	⍟	
Maximum	⌈	¯7.237 . . . E75
Minimum	⌊	7.237 . . . E75
Residue	∣	0
Circular	○	
Binominal	!	1
Or	∨	0
And	∧	1
Nor	⍱	
Nand	⍲	
Equal to	=	1
Not equal to	≠	0 Apply for
Greater than	>	0 logical
Not less than	≥	1 arguments
Less than	<	0 only
Not greater than	≤	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
4	5	6
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 identity element for the function. 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                      (1x5) + (2x7) = 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                                 (1x5) + (2x5) = 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
  27   30   33                   (3x8) + (4x11) = 68
  61   68   75
  95  106  117
```

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
```

Inner Product

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
```

Outer Product

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
 2  3  4        1+1=2
 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
```

Outer Product

Scan Operator (\)

The symbol for the *scan* operator is \. The forms of scan are: f\[I] B, f\B or f⍀B, where f 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 each of 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 f⍀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
```

+\[1]B: The first coordinate (rows) is specified; therefore, the scan is between rows.

+\[2]B: 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
 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
 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
 1  3  6 10
 5 11 18 26
 9 19 30 42

13 27 42 58
17 35 54 74
21 43 66 90
```

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

The second coordinate (rows) is specified; therefore, the scan operation is between rows for each plane.

The third coordinate (columns) is specified; therefore, the scan operation is between columns for each plane.

Scan

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
```

Special Symbols

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 (see *Suspension* in Chapter 7).
- To clear the state indicator (see *State Indicator* in Chapter 7).

Quad ⎕

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
```

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]

The quad quote symbol is formed 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.

Special Symbols

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:

```
      ∇PLUS[⎕]∇
    ∇ 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
```

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:

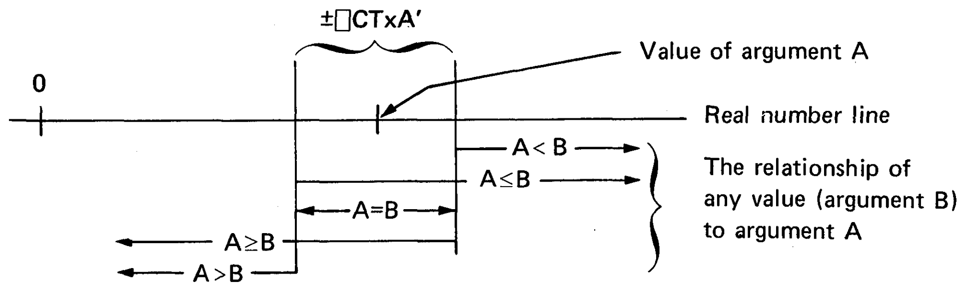
Variable Name	Meaning
⎕CT	Comparison tolerance
⎕IO	Index origin
⎕PP	Printing precision
⎕PW	Printing width
⎕RL	Random link
⎕LC	Line counter
⎕WA	Workspace available
⎕LX	Latent expression
⎕AV	Atomic vector

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 using 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.

Comparison Tolerance: ⎕CT

The value of this variable determines the maximum tolerance (how different the two numbers must be to be considered unequal) when using any relational function 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  ←———The first five integer powers of 16.
1 16 256 4096 65536
      ⎕CT←.1
      14=15.
0
```

Next lower integer power (A') is 1; therefore, the difference between the arguments exceeds ±⎕CTxA' (.1x1 = .1).

```
      15=16.
1
```

Next lower integer power (A') is 16; therefore, the difference between the arguments does not exceed ±⎕CTxA' (.1x16 = 1.6).

```
      253=255.
0
```

Next lower integer power (A') is 16; therefore, the difference between the arguments exceeds ±⎕CTxA' (.1x16 = 1.6).

```
      255=257.
1
```

Next lower integer power (A') is 256; therefore, the difference between the arguments does not exceed ±⎕CTxA' (.1x256 = 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 (?), grade up (⍋), and grade down (⍒).

```
      ⎕IO←0
      1 2 3 4⍳1 2 3 4
0 1 2 3
      ⍋6 3 4 5
1 2 3 0
      ⍳4
0 1 2 3
      ⎕IO←1
```

The index values represented by the result start from 0 rather than 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 precision 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
      12345.67
12346
      123456.7
1.2346E5
```

Five digits are displayed and the least significant digit is rounded off.

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 or printer, the output will overflow onto the next line.

Note: During function definition mode (see Chapter 6), the print width variable is automatically set to 390. The variable returns to its original value when the function is closed.

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.

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. For more information on ⎕LC, see Chapter 7.

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 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 C 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
	⎕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 ⎕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:

```
      ∇ 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
```

System Functions

To format the function INTG into a character matrix and assign the matrix to a variable named VAR, the following instruction would be entered:

```
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 matrix VAR can be changed by simply indexing the elements:

```
      VAR[4;12]←'I'◄────The element in row 4, column 12
      VAR               is changed to I.
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 using 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, missing single quote, etc) 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, the following instruction would be entered:

```
      ⎕FX VAR      APL responds with the name of
INTG ◄──────────── the user-defined function.
```

The ⎕FX function produces an explicit result (the array 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
```

System Functions

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 change (the local
  [2]    Y←⎕FX M           version will not exist after the execution of
  [3]    INTG 4∇           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
```

System Functions

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 out 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 8), it will be retracted.

System Functions

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 character A or 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)

System

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 5100 is in execution mode; that is, after a line has been entered and the EXECUTE key pressed, the 5100 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. The mode is changed 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
```

Note: During function definition mode, the print width (see ⎕PW system variable in Chapter 5) is automatically set to 390. The print width returns to its original value when the function is closed. This prevents problems that occur when editing statements that exceed the print width. Editing statements are discussed later in this chapter. 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 tape. (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 S∆ or T∆, because S∆ and T∆ 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	Niladac	∇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 assigning a label 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 only be used 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
```

Branching Labels

```
      ∇ S←SUM2 N
[1]     S←0
[2]     I←1
[3]    CHECK:→(I>N)/0 ←—— Branch to 0 (terminate the function)
                          or fall through.
[4]     S←S+I
[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 →(X=Y)⍴LABEL	Branches to LABEL or falls through

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 →I←3xA means I is assigned the value of 3 times the value of A, and the value of I 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 variables 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
VALUE ERROR                 has executed.
        X
        ^
        LOC                 The global value associated with this
100                         name was unchanged by the function.
        GLOB                The global value was changed by the
10                          function, since GLOB was not made
                            local name to the function.
```

Since 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 (see Chapter 7) 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)SIV command causes the SIV list (state indicator with local variables and local user-defined functions listing) to be displayed. The SIV 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 SIV 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 SIV display, variable P has referents as follows:

```
        )SIV
G[7]    *Z X I
F[4]     P J ←——— Active referent of P is local to function F.
Q[3]    *C X T
R[2]     P ←———— First latent referent of P
G[3]     Z X I    is local to function R.
```

As the state indicator is cleared (see Chapter 7), latent referents become active.

Local and

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 5100 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 →.

Requesting

ARRANGING THE OUTPUT FROM A USER-DEFINED FUNCTION

The output from user-defined functions can be arranged by using 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
```

Since 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:

```
      ∇ OUTPUTAINPUT
[1]     ⍞←'THIS IS BARE OUTPUT!!!'
[2]     ⍝THE NEXT STATEMENT REQUEST ⍞ INPUT
[3]     IN←⍞
[4]     ⍝NOW DISPLAY THE INPUT∇
        OUTPUTAINPUT
  THIS IS BARE OUTPUT!!!_
```

The cursor appears here. Now enter THIS IS ⍞ INPUT.

```
  THIS IS BARE OUTPUT!!!THIS IS ⍞ INPUT
```

After EXECUTE is pressed, the output line looks like this.

Output

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.
```

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 can be locked, or protected, by opening or closing the function definition 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 5100 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 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 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 in function definition mode:

Entry	Result
[n⎕]	Displays statement n
[⎕n]	Displays all statements from n onward
[⎕]	Displays all statements

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 4 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].

  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.

  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 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 ATTN to delete everything from the cursor position to the end of the line.

  3. Press EXECUTE. 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 EXECUTE 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, the function definition for function R is reopened 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 5100 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 5100 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 5100 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, since 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 5100 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 EXECUTE.

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 traced are specified by a trace vector. The format of the trace control function is T∆ 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 T∆ STEVE←2 3 5 is entered, the statements 2, 3, and 5 are traced each time function STEVE is executed. T∆ STEVE←⍳0 must be entered to discontinue the tracing of function STEVE. To trace each statement of the function, enter T∆ STEVE←⍳N, where N is the number of statements in the function:

```
        ∇STEVE[⎕]∇
      ∇ 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
        T∆STEVE←⍳4
        STEVE 2
STEVE[1] 2              Trace the first four statements
STEVE[2] 4              in function STEVE.
STEVE[3] 6
STEVE[4] 8
20
        T∆STEVE←2 ←——Trace statement 2 in function STEVE.
        STEVE 2
STEVE[2] 4
20
        T∆STEVE←⍳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]     TΔSTEVE←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 will establish trace control for the first statement of each user-defined function in the active workspace:

```
⍎¯1↓,'0','ρ','T','Δ',(((⎕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 will establish a trace vector for each statement in a specified user-defined function:

```
        ∇TRACE[⎕]∇
      ∇ TRACE NAME
[1]     ⍎'TΔ',NAME,'←⍳1↑ρ⎕CR''',NAME,''''
      ∇
```

When executing the function TRACE, 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
        TΔSTEVE←⍳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 SΔ 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 (see Chapter 7); execution is resumed by entering →⎕LC (see Chapter 5). SΔSTEVE←⍳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, SΔ STEVE←4xN>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
        SΔF←1 2 3
SYNTAX ERROR
        SΔF← 1 2 3
        ^
        )VARS
F
        F
1 2 3
```

Stop

SUSPENSION

The execution of a user-defined function can be interrupted (suspended) in a variety of ways: by an error message (see Chapter 11), by pressing ATTN (see Chapter 1), or by using the stop control vector (see Chapter 6). In any case, the suspended function is still considered active, since its execution can be resumed. Whatever the reason for the suspension, when it occurs, the statement number of the next statement to be executed is displayed. A branch to the statement number that was displayed or a branch to ⎕LC (→⎕LC, see Chapter 5) causes normal continuation of the function, and a branch out (→0) exits the function.

When a function is suspended, the 5100 will:

- Continue to execute system commands except)SAVE,)COPY, and)PCOPY.
- Resume execution of the function at statement n when →n is entered.
- Reopen the definition of any function that is not pendent. A pendent function is a function that called the suspended function. If a function called a function that called a suspended function (and so on), it is also pendent (see *State Indicator* in this chapter).
- Execute other functions or expressions.

Note: The display of output generated by previous statements might have been interrupted when the suspension occurred. This would be caused by the delay between execution of the statement and the display of the output.

STATE INDICATOR

The state indicator identifies which functions are suspended (*) and at what point normal execution can be resumed. Entering)SI causes a display of the state indicator. Such a display might have the following form:

```
      )SI
H[7]  *
G[2]
F[3]
```

This display indicates that execution was halted just before statement 7 of function H, that the current use of function H was invoked in statement 2 of function G, and that the use of function G was invoked in statement 3 of F. The * appearing to the right of H[7] indicates that function H is suspended; the functions G and F are said to be pendent.

Suspension

During the suspension of one function, another function can be executed. Thus, if a further suspension occurred in statement 5 of function Q, which was invoked in statement 8 of G, a display of the state indicator would be as follows:

```
      )SI
Q[5]  *
G[8]
H[7]  *
G[2]
F[3]
```

An SI DAMAGE error (see Chapter 11) indicates that a suspended function has been edited or a pendent function has been erased and the normal execution of the suspended function can no longer be resumed. Therefore, when an SI DAMAGE error occurs, the state indicator display will not include the damaged function name (however, the asterisk is still displayed). For example, if function Q is edited and the modification causes an SI DAMAGE error, the display of the state indicator would be as follows:

```
      )SI
      *
G[8]         No suspended function name is displayed.
H[7]  *
G[2]
F[3]
```

State Indicator

A suspension can be cleared by entering a branch with no argument (that is, →). One suspended function is cleared at a time, along with any pendent functions for that suspended function. The first branch clears the most recently suspended function, as the following example shows:

```
      →
      )SI
H[7]  *
G[2]
F[3]
```

It is a good practice to clear suspended functions, because suspended functions use available storage in the active workspace. Repeated use of → clears all the suspended functions; as the functions are cleared, they are removed (cleared) from the state indicator. When the state indicator is completely cleared, the state indicator display is a blank line.

Note: To display the state indicator with local names, enter the)SIV command (see *Local and Global Names* in Chapter 6 for more information on the SIV list).

State Indicator

Chapter 8. Tape and Printer Input and Output

Input and output involving the tape or printer can be done with an APL shared variable, which is a specific variable shared between the active workspace and the tape or printer. During output operations, the data assigned to the shared variable is printed, or is written on tape. During input operations, data is read from tape and assigned to the shared variable; the shared variable can then be used in an expression in the active workspace. To do tape or printer input or output, the following steps must be performed:

1. Establish a variable to be shared.
2. Open a data file on tape or specify printer output.
3. Transfer the data.
4. Close the data file or terminate the printer output.
5. Retract the variable being shared.

ESTABLISHING A VARIABLE TO BE SHARED

The ⎕SVO function is used to establish the variable name(s) to be shared. The ⎕SVO function is dyadic (requires two arguments) and is entered as follows:

```
1 ⎕SVO 'NAME(S)'
```

The left argument must be a 1.

The right argument NAME(S) can be up to eight variables to be shared. If more than one name is required, the names must be entered as a character matrix with each row representing a name. For example:

```
      SHARE←3 3ρ'ONETWOTHR'
      SHARE
ONE
TWO
THR
```

Each row represents a separate variable name.

Following are three examples of how the ⎕SVO function can be entered:

- ```
  1 ⎕SVO 'DATA'
  ```

- ```
  A←'DATA'
  1 ⎕SVO A
  ```

- ```
  SHARE←3 1⍴'ABC'
  1 ⎕SVO SHARE
  ```
  Establishes three names (A, B, and C) to be shared.

The 5100 will respond with a 2 for each shared variable that is successfully established and a 0 or 1 for each variable that is not. If a 1 is displayed, a value other than 1 was specified as the left argument for the ⎕SVO function. In this case, the variable name must be retracted (see *Retracting the Variable Name being Shared* later in this chapter) and reestablished as a shared variable before it can be used for input/output. If a 0 is displayed, an error message (see Chapter 11) will also be displayed.

Note: The instruction +/0≠⎕SVO ⎕NL 2 will display the existing number of shared variables in the system, and the instruction (0≠⎕SVO ⎕NL 2)/[1] ⎕NL 2 will display the existing shared variable names.

OPENING A DATA FILE OR SPECIFYING PRINTER OUTPUT

The first value assigned to the shared variable must be information required to open a data file on tape or to specify printer output. When opening a data file, this information specifies the following:

- Data to be transferred to tape or from tape
- Device/file number
- File ID
- Data format to be used

Note: If this information has already been assigned to a variable name that is being used as the right argument for the ⎕SVO function, the 5100 will establish the variable name to be shared, then open the data file or specify printer output. The return codes are described later in this chapter.
  ```

This information must be character data (enclosed in single quotes) and must be entered with a blank between each parameter, as follows:

		IN				A
		or				or
		OUT				I
name ←	'	or	device/file number	[ID=(file ID)]	[MSG=OFF]	[TYPE=or
		ADD				I1
		or				or
		PRT				I2]

where:

name is the name of the variable being shared.

IN specifies that the data is to be transferred from tape into the active workspace.

OUT specifies that the data is to be transferred to a tape file.

ADD specifies that the data is to be transferred to an existing tape file, following the last record in that data file.

PRT specifies that the data is to be printed.

Note: When PRT is specified, the only other information that can be specified is MSG=OFF (which is defined later).

device/file number specifies the tape unit and file number. For example:

1003

File Number 3

Tape Unit 1

Note: If fewer than four digits are used, tape unit 1 is assumed, and the value entered represents only the file number.

ID=(file ID) (optional) specifies from 1 to 17 characters enclosed in parentheses:

- For an IN or ADD operation, the entry (file ID) is compared to the file ID in the file header; the open fails if they do not match.

- For an OUT file, the entry (file ID) is put in the file ID field of the file header (see the)LIB command in Chapter 2). If the ID=(file ID) parameter is not specified, the characters DATA are put in the file ID field.

  It is a good practice to give the data files meaningful names; for example, a file that contains sales data could be named SALES. Also, any blanks within the 17 characters become part of the file ID.

  Note: To do an OUT operation to an existing data file (write new data over the existing data), the file ID specified must match the existing file ID for the data file or the data file must be dropped using the)DROP command (see Chapter 2).

MSG=OFF (optional) specifies that no error message is to be displayed for nonzero return codes (see *Return Codes* in this chapter).

A
or
I
TYPE=or (optional) can only be specified for OUT operations. It specifies the
I1
or
I2

data format to be used when writing data to tape:

- When TYPE=A is specified, the APL internal data format is used; that is, the data is written on tape in the same format that it is stored in, in the active workspace.

- When TYPE=I or TYPE=I1 is specified, the exchange data format is used. When the exchange data format is used, only character scalars or vectors can be assigned to the variable being shared. Therefore, when storing numeric data or arrays on tape using the exchange data format, the data must first be changed to a character scalar or vector (see the ⍕ function in Chapter 4).

  The following items apply to an exchange data file that is used by both the 5100 APL and BASIC languages:

  1. All data items must be separated by commas. For example, the numeric vector 1 3 5 6 must be changed to character data, then commas placed in the blank positions.

  2. Negative signs must be replaced by minus signs.

  3. Enclosing single quotes must be part of any data that represents character constants. Also, any embedded quotes in the character constant must be represented by double quotes.

     Note: The 5100 BASIC language accepts only the first 18 characters in each character constant.

  4. The 5100 BASIC language creates a logical record for each PUT statement or each row of an array with a MAT PUT statement.

- When TYPE=I2 is specified, the general exchange data format is used; it is the same as TYPE=I (and TYPE I1) except that the data file can also be used as a BASIC language source file.

Note: The data format can be specified only for an OUT operation. For IN or ADD operations, the data format is specified by the data file type (see)LIB command in Chapter 2). If the data format is not specified for an OUT operation, the APL internal data format (TYPE=A) is used.

CAUTION
If the tape cartridge is removed from the 5100 when an OUT or ADD file is open, the file will be unusable. If another tape is inserted at this point, one of its files may be destroyed. See *Closing a Data File or Terminating the Printer Output* in this chapter for information on how to close a data file.

The following four examples, using an APL shared variable named EXAMPLE, show how the information required to open a data file or specify printer output can be entered:

1. ```
EXAMPLE←'IN 1001 ID=(TEST)
```

   - IN — Data is to be transferred from tape into the active workspace.
   - 1001 — File 1 on tape unit 1 is to be opened.
   - TEST — The value TEST will be compared to the file header field.

2. ```
EXAMPLE←'OUT 003 ID=(TEST2) TYPE=I'
```

   - OUT — Data is to be transferred to tape from the active workspace.
   - 003 — File 3 on tape unit 1 (assumed) is to be opened.
   - TEST2 — The value TEST2 will be placed in the device header field.
   - I — Exchange data format is to be used.

3. ```
EXAMPLE←'ADD 3'
```

   - ADD — Data is to be transferred into an existing file on tape.
   - 3 — File 3 on tape unit 1 (assumed) is to be opened.

   *Note:* Since the file ID was not specified, no value is compared to the file header field.

4. ```
EXAMPLE←'PRT MSG=OFF'
```

   - PRT — Data is to be printed.
   - OFF — No error messages will be displayed for nonzero return codes.

After the information has been entered, a code (2-element vector) that indicates whether the operation was successful or not is assigned to the shared variable. A return code of 0 0 indicates the operation was successful, and a nonzero return code indicates that the operation failed. See *Return Codes* in this chapter for a description of each return code.

Opening File

TRANSFERRING DATA

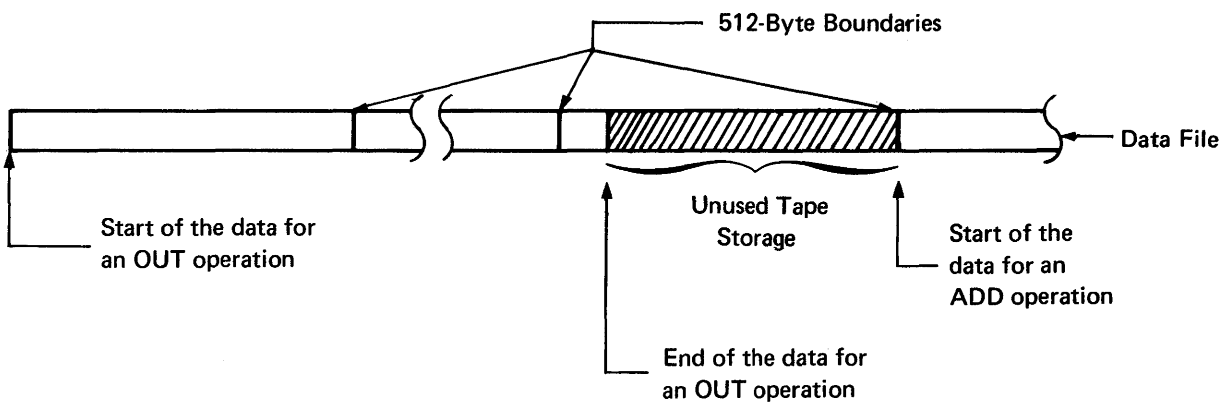
After the data file has been opened or printer output specified, data can be transferred using the shared variable. (An example using tape and printer input/output is shown later in this chapter.)

Transferring Data to Tape (OUT or ADD Operation)

When data is assigned to the shared variable, the data is written on tape and a return code is assigned to the shared variable. A 0 0 return code means the data was transferred successfully and a nonzero return code means the transfer of data 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), any existing data following the new data cannot be used again.

For ADD operations, the new data is written to the data file starting at the 512-byte boundary following the last record in the file. This might cause some tape storage to be unused, for example:



The unused tape storage that results from an ADD operation is unavailable for use. However, you can make all the unused tape storage available for use (compress the data), as follows:

1. Transfer the data from the data file.

2. Perform an OUT operation to write all the data back on tape.

Transferring Data

Transferring Data from Tape (IN Operation)

When data is transferred from tape, the data is read from tape and is assigned to the shared variable in the same sequence as it was written to tape. New data is read from the tape file and assigned to the shared variable each time the shared variable is used. (There is no return code assigned to the shared variable after an IN operation.)

Using the format function (⍕) directly on a character shared variable when doing input operations causes alternate records to be skipped.

When doing an IN operation with an exchange data file, the following conditions occur if a cursor return character (X'9C') or end-of-block character (X'FF') was embedded in a character vector that was written to tape:

- If a cursor return character was embedded in the character vector, the data will be read from tape in a different sequence than it was written to tape. This condition occurs because as the interchange data is written to tape, the system writes an end-of-record character (X'9C') after each character vector (record) that was written to tape. The end-of-record character and the cursor return character are the same. When used on tape, this character separates the data (records) so that it can be read from tape in the same sequence as it was written to tape. However, if a cursor return character is embedded in the data that was written to tape, the system will recognize it as an end-of-record character when the data is read from tape.

- If an end-of-block character was embedded in the character vector, any data from the embedded end-of-block character to the next physical record is not read from tape. This condition occurs because the system looks at the tape in 512-byte segments (one physical record). A physical record can be terminated by an end-of-block character (X'FF'). When the system is reading data from the tape and an end-of-block character is encountered, the system skips to the next physical record and continues reading data. Therefore, if an X'FF' character is embedded in the data that was written to tape, the system recognizes it as an end-of-block character when the data is read from tape and skips ahead to the next physical record.

Transferring Data to the Printer (PRT Operation)

When data (character scalars or vectors only) is assigned to the shared variable, it is printed and a return code is assigned to the shared variable. A 0 0 return code indicates the data was printed successfully and a nonzero return code indicates the operation failed. See *Return Codes* in this chapter for a description of each return code.

Note: The)OUTSEL OFF command is automatically issued by the system when doing PRT operations. The)OUTSEL option will return to its previous setting after the PRT operation has been terminated (PRT termination is discussed next).

CLOSING A DATA FILE OR TERMINATING THE PRINTER OUTPUT

Transferring an empty vector will close the data files or terminate the printer output and a final return code will be issued. A 0 0 return code indicates the file was closed or printer output was terminated successfully. See *Return Codes* in this chapter for a description of each return code. Also, for an IN operation, the file is closed and a return code is issued if an error occurs due to the device or if an end-of-file empty vector is returned.

CAUTION
For OUT and ADD operations, if the tape cartridge is removed from the 5100 before a data file is closed, the data in the file will be unusable.

After a data file has been closed, another data file can be opened by assigning the information required to open a file to the shared variable. Once the tape and printer input and output operations are done and the data files are closed or printing is terminated, the variable name being shared should be retracted. How to retract the variable name is discussed next.

RETRACTING THE VARIABLE NAME BEING SHARED

The ⎕SVR function is used to retract a variable name being shared. That is, once the ⎕SVR function has been used successfully, the variable name still exists as an APL variable, but it cannot be used to transfer data to tape or printer, unless it is reestablished as a shared variable. The ⎕SVR function is monadic (takes one argument) and is entered as follows:

```
⎕SVR 'NAME(S)'
```

where NAME(S) can be the names of up to eight variables. If more than one name is required, the names must be in a character matrix with each row representing a name (see *Establishing a Shared Variable* earlier in this chapter).

The 5100 will respond with a 2 (or a 1 if the left argument for the ⎕SVO function was not a 1–see *Establishing a Variable to be Shared* in this chapter) for each variable name that is successfully retracted and a 0 for each variable name that is not successfully retracted. Normally, if a variable name cannot be successfully retracted, it was never properly established as a shared variable.

Note: If the ⎕SVR function is used before a file is closed, the system will automatically close the file.

RETURN CODES

Return codes assigned to the shared variable when doing input/output operations indicate whether or not the operation was successful. If the return code is nonzero and MSG=OFF was not specified, an error message is also displayed.

Operation of the system does not stop when a nonzero return code is assigned. Therefore, if you have a user-defined function that is doing input/output operations, the user-defined function should check the return code that was assigned to the shared variable to make sure each operation is successful. When you are checking the return code, the shared variable cannot be referred to more than once.

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 0		Operation successful.
1 eee		Device error; the second element (eee) is the error code (see ERROR eee ddd in Chapter 11).
2 0	INVALID FILE	The specified file cannot be used for input/output operations.
3 0	INVALID DEVICE or INVALID DEVICE NUMBER	Enter the information required to open the file again, using device number 1 or 2.
4 0	INVALID FILE NUMBER	Enter the information required to open the file again, using a valid file number.
5 0	NOT WITH OPEN DEVICE	The specified device is already being used for input/output operations; the existing open file must be closed before another file can be opened.
6 0	INVALID PARAMETER	The information required to open the file was entered incorrectly; enter it again, correcting any keying errors.
7 0	WS FULL	Use the)ERASE command to erase any unwanted objects; then enter the information required to open the file again.
8 0	DEVICE NOT OPEN	Open the file.

Codes

Code	Error Message	Description and/or User's Response
9 0		This return code is only a warning; an empty vector was read from tape, but the empty vector is not the end-of-file empty vector.
10 0	EXCEEDED MAXIMUM RECORD LENGTH	This 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	INVALID DATA TYPE	The wrong type of data was used; for example, noncharacter data was sent to an exchange file, noncharacter data was used as the information required to open a file, or non-character data was sent to the printer.

AN EXAMPLE USING TAPE AND PRINTER INPUT/OUTPUT

In this example, file number 11 on tape unit 1 will be used as a data file. First, a variable name must be established to be shared and the data file opened so that data can be written to the file (OUT operation):

```
      )LIB 11
011                                00 016
      1 ⎕SVO 'SHARE'
2
      SHARE←'OUT 1011 ID=(INVENTORY)'
      SHARE
0 0
```

File 11 is an unused file.

Establish a variable name to be shared.

Open the data file.

Check the return code that was assigned to the shared variable.

The file was opened successfully.

Example

Now, as data is assigned to the shared variable, it is transferred (written) to the data file:

```
        SHARE←'2456300 SCREW     5000'
        SHARE
0 0
        SHARE←'2456400 NUT       7000'
        SHARE
0 0
        SHARE←'2456550 WASHER      50'
        SHARE
0 0
        SHARE←'5357800 CIRC BD     10'
        SHARE
0 0
        SHARE←⍳0 ←———————After all the data has been transferred,
        SHARE              the file must be closed.
0 0
```

If more data is to be added to an existing data file but the file is closed, a variable name must be established to be shared and the data file opened again:

Note: In this example, the variable name SHARE has not been retracted and can still be shared.

```
                                        Open the data file again.
      SHARE←'ADD 1011 ID=(INVENTORY)'
      SHARE
0 0
      SHARE←'5357950 BOARD          5'
      SHARE                              These records are added
0 0                                      following the existing
      SHARE←'5357951 A/W            1'   records in the file.
      SHARE
0 0
      SHARE←⍳0 ——————————The file is closed.
      SHARE
0 0
```

Since no more data is to be written on tape, the shared variable should now be retracted:

```
        ⎕SVR 'SHARE'
  2
```

Now, assume that at a later time you want to read the data from file 11 and print it on the printer, using the following user-defined function:

```
      ∇PRINT[⎕]∇
    ∇ PRINT;WORK
[1]   ⍝READ DATA FROM THE DATA FILE AND ASSIGN IT TO WORK
[2]   LOOP:WORK←DATA
[3]   ⍝CHECK FOR AN EMPTY VECTOR (INDICATING AN END OF FILE OR
[4]   ⍝TAPE ERROR)--AN EMPTY VECTOR HAS A SHAPE OF 0 (NO ELEMENTS)
[5]    →(0=⍴WORK)/DONE
[6]   ⍝DISPLAY AND PRINT THE VALUE ASSIGNED TO WORK
[7]    PRNT←⎕←WORK
[8]   ⍝CHECK THE RETURN CODE FOR THE PRINT OPERATION
[9]    →(0 0 ≠+/WORK←PRNT)/PRINTΔERROR
[10]   →LOOP
[11]  PRINTΔERROR: PRINT ERROR--THE RETURN CODE IS:
[12]   WORK
[13]   →0
[14]  ⍝ TERMINATE THE PRINTER OUTPUT
[15]  DONE:PRNT←⍳0
[16]  ⍝CHECK THE RETURN CODES TO MAKE SURE ALL TAPE INPUT
[17]  ⍝OPERATIONS WERE SUCCESSFUL AND THE SHARED VARIABLE
[18]  ⍝PRINTER OUTPUT IS TERMINATED
[19]   →(0 0 ≠+/WORK←DATA)/TAPEΔERROR
[20]   →(0 0 ≠+/WORK←PRNT)/PRINTΔERROR
[21]   →0
[22]  TAPEΔERROR:'TAPE ERROR--THE RETURN CODE IS:
[23]   WORK
    ∇
```

The variable names to be shared must be established again and the data file opened. Also, printer output must be specified:

```
      NAMES←2 4⍴'DATAPRNT'
      NAMES
DATA
PRNT
      1 ⎕SVO NAMES        ← Establish the variable names
2 2                         to be shared.
                          Open the data file for input.
      DATA←'IN 1011 ID=(INVENTORY)'
      DATA
0 0
      PRNT←'PRT'          ← Specify printer output.
      PRNT
0 0
```

Example

Now, when the function PRINT is executed, the data file is read, displayed, and printed:

```
      PRINT
2456300 SCREW      5000
2456400 NUT        7000
2456550 WASHER       50
5357800 CIRC BD      10
5357950 BOARD         5
5357951 A/W           1
```

After the operation is complete, the shared variable names should be retracted:

```
      ⎕SVR NAMES
2 2
```

DATA SECURITY

You are primarily responsible for the security of any sensitive data. After you are through using the 5100, the data in the active workspace can be removed by one of the following:

- Using the)CLEAR command to clear the active workspace
- Pressing the RESTART switch
- Turning the POWER ON/OFF switch to off

There are several methods available for protecting or removing sensitive data on a tape. These methods are:

- Assigning a password to the workspace when writing the active workspace on tape.
- Rewriting a tape file, which makes the old data inaccessible.
- Filling a data file with meaningless data. For example, the following user-defined function fills file 4, a data file named DATA on tape 1, with zeros:

```
          ∇SECURITY[⎕]∇
        ∇ SECURITY
[1]       1 ⎕SVO 'A'
[2]       A←'OUT 4 ID=(DATA)'
[3]       B← 10 1000 ⍴0
[4]     WR:A←B
[5]       →(A[1]=0)/WR
        ∇
```

Note: **ERROR 010 ddd will be displayed after the data file has been filled with zeros. When this error is displayed, enter**

```
        A←⍳0
        ⎕SVR 'A'
  2
```

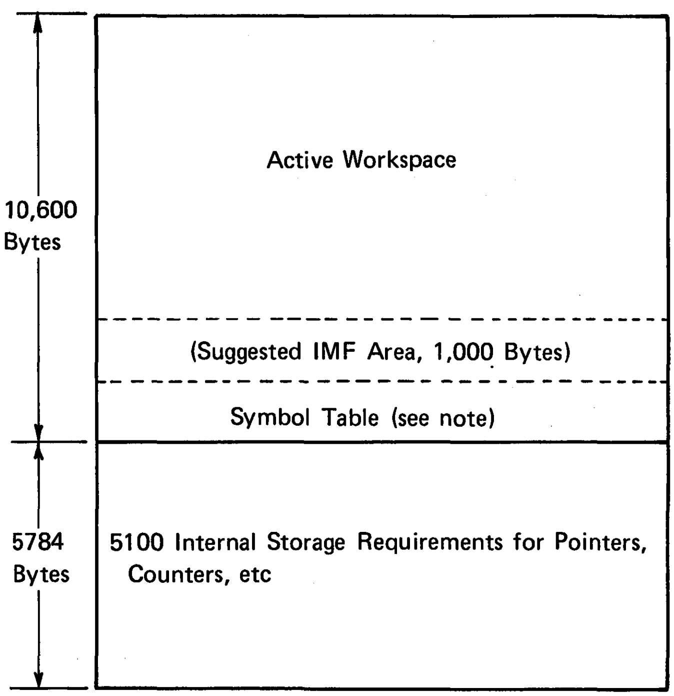
Data Security

5100 STORAGE CAPACITY

The base 5100 (Model A1) has a storage capacity of 16K (K = 1024 bytes). Figure 4 shows how this storage is allocated for various requirements. Notice that the workspace available to the user (active workspace) is 10,600 bytes, while the remaining bytes are used for internal purposes. The storage capacity is increased in the following models of the 5100:

Model A2 is 32K
Model A3 is 48K
Model A4 is 64K

In these models, all additional storage is allocated to the active workspace. For example, on the Model A4, the active workspace is approximately 60,000 bytes.



Note: The symbol table requires eight bytes of storage for each symbol allowed in the active workspace (see)SYMBOLS in Chapter 2).

Figure 4. Storage Allocation for a Model A1 5100

Storage Considerations

The following list shows how many bytes of storage are required for each data type that can be in the active workspace:

Data Type	Number of Bytes Required
Character constant or variable name	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)

Because the 5100 active workspace contains a fixed amount of storage, it is good practice to conserve as much storage as possible. Following are some considerations that can be used to conserve storage:

- Make all objects (variables and user-defined functions) not required for use outside of a user-defined function local to the function.
- Store data in data files on the tape, and use an APL shared variable (see Chapter 8) to transfer the data into the active workspace when required.
- Clear suspended functions (see Chapter 7) from the active workspace.
- Group user-defined functions by related operations and store each group into a workspace file on tape. Then when a certain group of related functions is required to process data in the active workspace, the stored workspace containing these functions can be copied into the active workspace. When the processing is done, the functions can be expunged (see Chapter 5) and another group of functions (one workspace) can be copied into the active workspace.
- If a value consists of all 1's and 0's, store the value as logical data. For example, you have the following vector:

```
      VECTOR←10⍴(2-1)
      VECTOR
1 1 1 1 1 1 1 1 1 1
```

  The result is a vector of ten 1's, and each 1 requires four bytes of storage. However, the vector can be changed to a logical vector as follows:

```
      VECTOR←1∧VECTOR
      VECTOR
1 1 1 1 1 1 1 1 1 1
```

  The result looks just like the previous result; however, only 2 bytes of storage was required.

- Since each data item requires at least 12 bytes of overhead, an array of six elements would require approximately 60 bytes less storage than six scalars.

- Names of 3 characters or less require 8 bytes of storage in the symbol table (the symbol table is part of the active workspace where the names of all the symbols, including variables, user-defined functions, and labels, are stored). Names of 4 characters or more require an additional 8 bytes plus 1 byte for each character in the name.

  Note: Even if an object is erased from the active workspace, the storage used for its name will not be available for use unless the contents of the active workspace are written to tape with a)SAVE command and then loaded or copied back into the active workspace.

- Identical names that are local to more than one user-defined function do not require additional symbol-table space for each function.

When the contents of the active workspace are written to tape using the)CONTINUE command, then the stored workspace is loaded into a 5100 with a larger active workspace, the amount of available workspace (see ⎕WA system variable in Chapter 5) remains the same as it was when the contents of the active workspace were originally written to tape. To take advantage of the additional storage in the larger active workspace, write the contents of the active workspace to tape using the)SAVE command, then load the stored workspace back into the 5100.

The following formula shows how much storage in the active workspace is required to perform an input or output operation to tape using an APL shared variable (see Chapter 8):

REQUIRED∆STORAGE = BUFFER + SHARED∆VARIABLE

where:

- REQUIRED∆STORAGE is the amount of storage that must be available in the active workspace (see ⎕WA in Chapter 5) before an input or output operation to tape can be performed. If there is not enough available storage, a WS FULL error occurs.

- BUFFER is the amount of storage in the active workspace reserved by the 5100 for input and output operations. This storage is reserved when the data file is opened. For all output operations and input operations using an internal data file (file type 8), BUFFER is about 650 bytes. For input operations using an exchange data file (file types 2 and 3), BUFFER is about 650 bytes plus the storage required for the largest record in the data file.

- SHARED∆VARIABLE is the amount of storage required for the data assigned to the shared variable.

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) or sunlight.

- Do not touch or clean the tape surface.

- If a data cartridge has been exposed to a temperature drop exceeding 30° F 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 200 1 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

Chapter 10. 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 will be 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 will be 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 mm) wide for individual forms and from 3 to 15 inches (76.2 to 381 mm) wide for continuous forms.

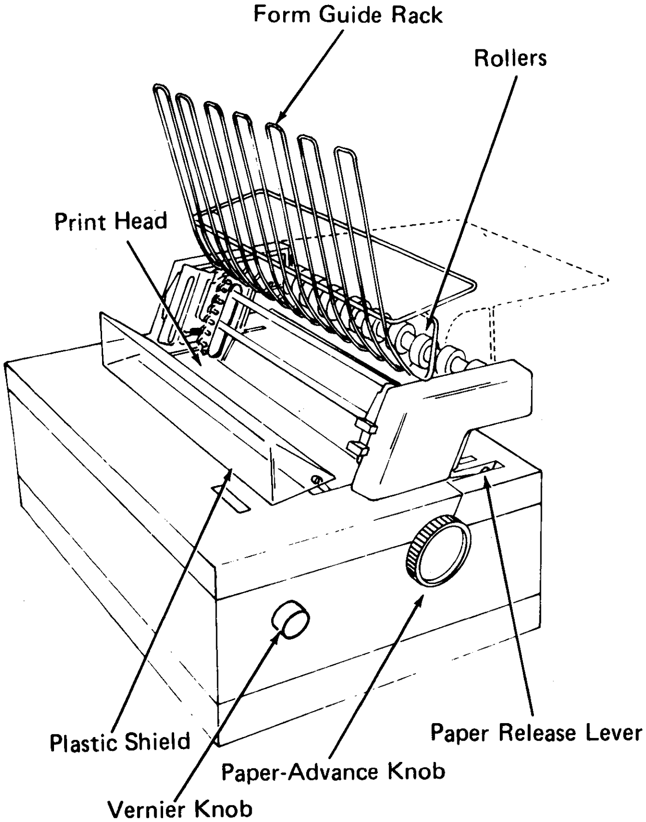
- Print position spacing of 10 characters per inch and line spacing of six lines per inch.

- Stapled forms or continuous card stock cannot be used.

- The character printing rate is 80 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.

How to Insert Forms



1. Pivot the plastic shield forward.

2. Push the print head to the extreme left position.

3. For single part forms, pivot the form guide rack up and forward to a vertical position. For multipart forms, leave the form 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.

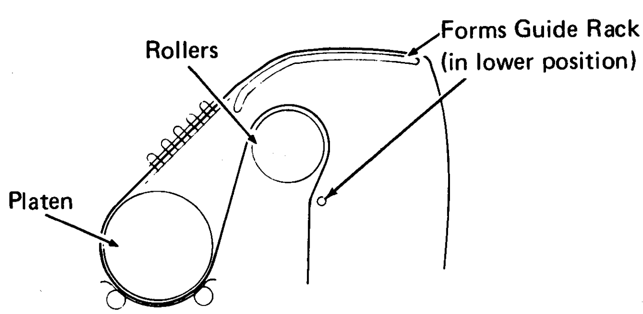
5. Place the forms on the table behind the printer.

   Note: The forms must be positioned behind the printer so that the forms feed squarely into the printer.

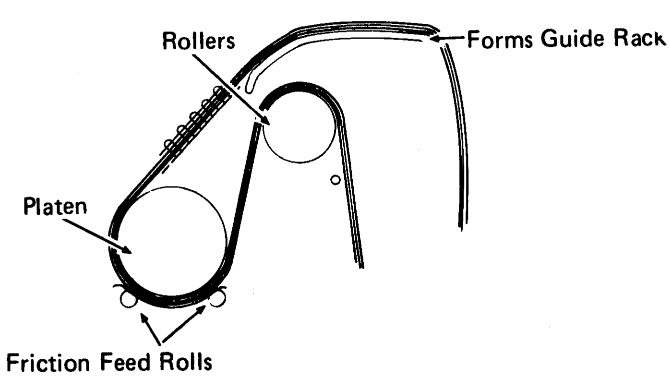
6. Thread the paper down, over the rollers, behind the tractors, and behind the platen.

7. Turn the paper-advance knob to move the paper around the platen until you can grasp it with your fingers.

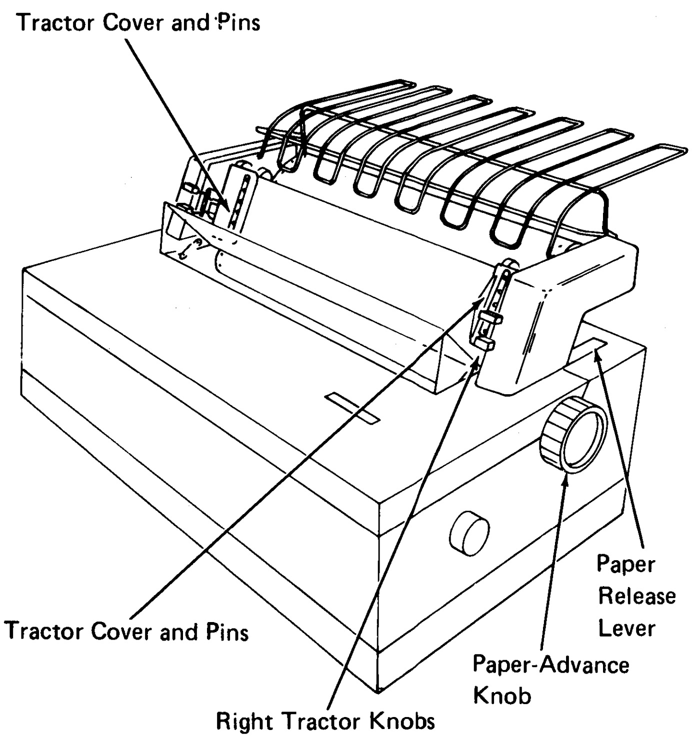
Forms Path for Singlepart Forms



Forms Path for Multipart Forms



Printer



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 form 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 form 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.

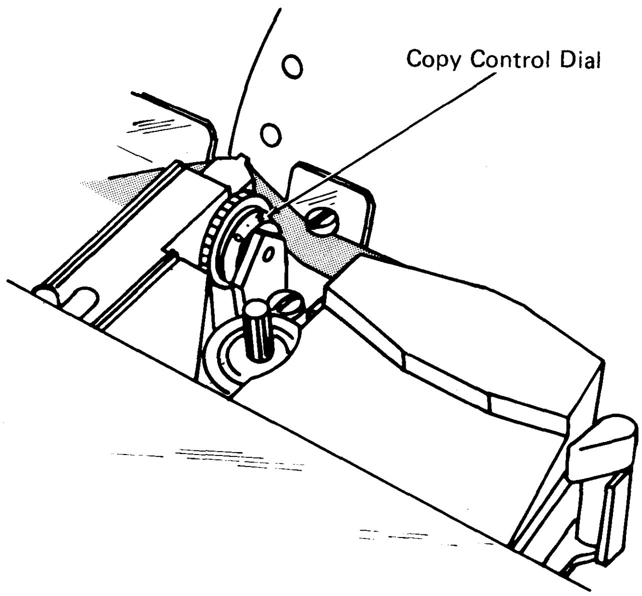
17. Close the plastic shield.

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. If you are installing the printer, return to step 7 of the *Printer Installation Procedure.*

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

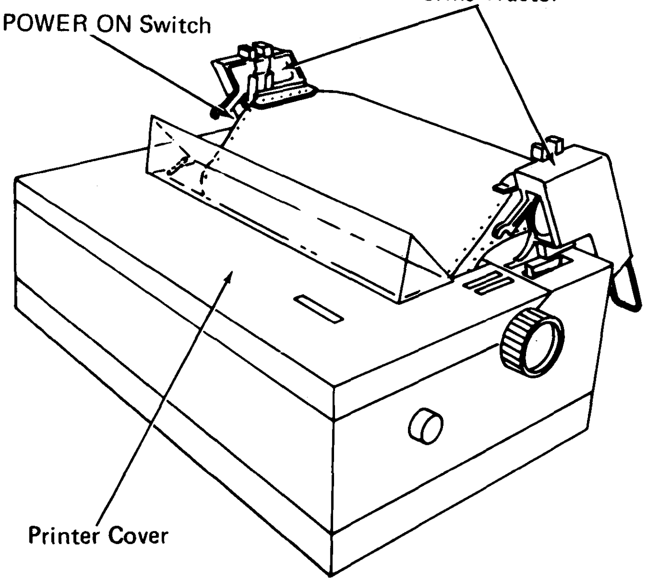


1. If you are using singlepart forms, set the copy control dial on 0.

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.

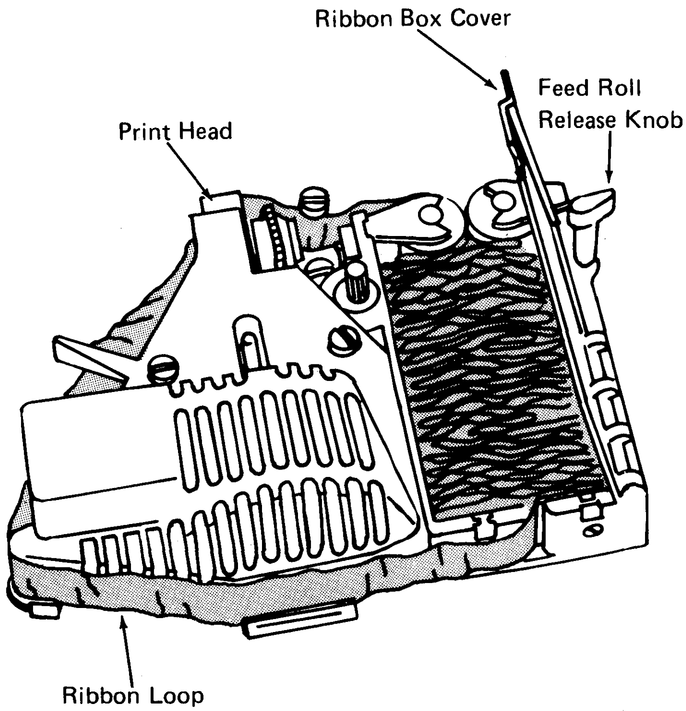
How to Replace a Ribbon (Part Number 1136653)



1. Turn off power to the printer.

2. Tilt the forms tractor back by lifting both sides at the front.

3. Raise the printer cover.



4. Be sure that the print head is to the extreme left.

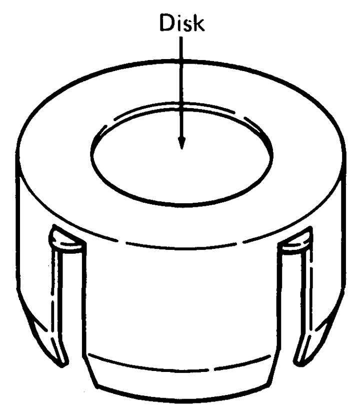
5. Turn the feed roll release knob counterclockwise until it points to the right.

6. Open the ribbon box cover.

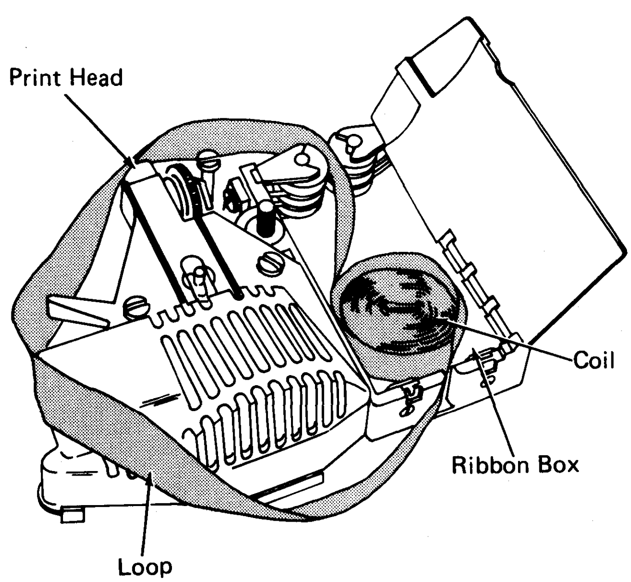
7. Put on the gloves supplied with the new ribbon.

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



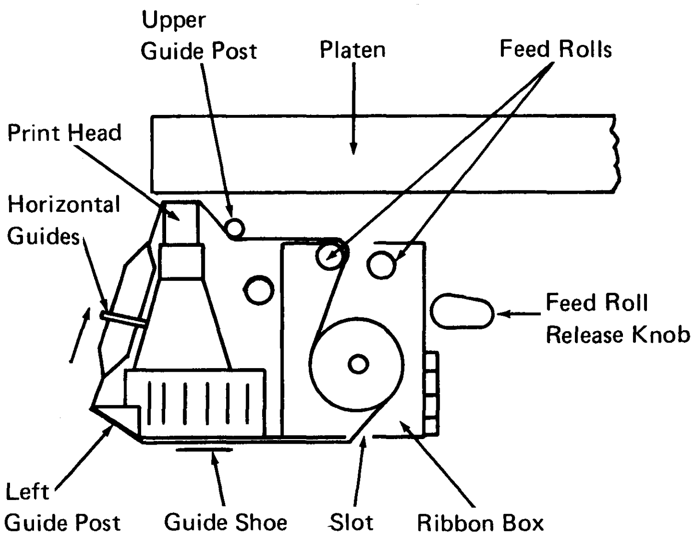
10. Eject the new ribbon from its holder into the ribbon box by pressing on the disk.

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.

Printer



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.

Printer

Chapter 11. Error Messages

Error messages can result when using 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 is to be remarked, enter GO. *Note:* Any existing data in the files following the last re-marked file will no longer be available.
CHARACTER ERROR	An invalid character was entered.	Enter a corrected statement.
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 (see Chapter 7), then edit the function.
DEVICE NOT OPEN	An attempt was made to read a data file and the file is not open.	Assign the information required to open the file to the shared variable.
DEVICE TABLE FULL	An attempt was made to establish more than eight variable names to be shared for tape or printer input/output.	Retract any unused shared variable names.

Error Message	Cause	User's Response
DOMAIN ERROR	The function indicated by the caret (∧) cannot operate on the arguments given: • The result exceeds the capacity of the 5100 (<¯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 joined 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.
ERROR eee ddd	eee is the error code for an input/output device operation and ddd is the device number. The device numbers are: 500—printer; 001—built-in tape unit; 002—auxiliary tape unit. Following is a list, cause, and user's response for the input/output device error messages:	
ERROR 002 ddd	Command error.	Performing tape operations with an unMARKed cartridge will cause error 004. Otherwise, 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.	
	Uneven winding of the tape.	Move the tape to its limits using the procedure described under *Tape Data Cartridge Handling and Care,* in Chapter 9.
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, turn the SAFE switch on the tape cartridge off of the SAFE position.
ERROR 007 ddd	Tape read error.	Use the)PATCH command and Tape Recovery program (see Chapter 2) to recover as much data as possible.
ERROR 008 ddd	The tape cartridge was probably removed from the tape unit when data or a workspace was being written to tape. The data in the file cannot be used.	Try the operation again. If the error occurs again, copy the files following the file that caused the errors onto another tape. Then use the)MARK command and re-mark the tape from the file that caused the error.

Error Message	Cause	User's Response
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.
ERROR 011 ddd	A file number was specified that has not been marked.	Specify the correct file number 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 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 050 ddd	The printer has run out of forms.	Insert forms in the printer (see Chapter 10).
ERROR 051 ddd	The printer POWER ON/OFF switch is turned off.	Turn the POWER ON/OFF switch on.
ERROR 052-059 ddd	Printer errors.	Try the operation again. If the error occurs a second time, call your service representative.
EXCEEDED MAXIMUM RECORD LENGTH	The tape was removed before the data file was closed during a tape input/output operation.	The data in the file cannot be used.
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 function. *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).
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.

Error Message	Cause	User's Response
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 (ρA) against the index values. If a primitive function or operator is being indexed, determine the rank(s) (ρρA) 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 eight variable names to be shared for tape or printer input/output.	Retract any unused shared variable names.
INTERRUPT	Attention was pressed twice when the 5100 was processing data or an invalid tape input/output operation was attempted.	If an invalid tape input/output operation was attempted, check the file open information to make sure the file was opened correctly.
INVALID DATA TYPE	Only exchange data can be used, but there was an attempt made to use data that is not a character scalar or vector.	Change the data to a character scalar or vector.
	An attempt was made to open a data file with other than character data.	Enclose the information required to open the data in single quotes.
INVALID DEVICE	A device was specified that does not exist or is incorrect for the operation to be performed.	Specify the correct device number.
INVALID DEVICE NUMBER	A device number that does not exist was specified.	Specify the correct device number.
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 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.

Error Message	Cause	User's Response
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 open 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 open the data file.
INVALID OPERATION	An invalid tape input/output operation was attempted. This message is followed by an INTERRUPT error message.	Check the file open information to make sure the data file was opened correctly or make sure you are using the shared variable correctly.
INVALID PARAMETER	A keying error was made or an incorrect parameter was specified when entering the information required to open a data file or specifying printer output.	Enter the file open information or system command again, correcting the keying errors.
	A keying error was made when entering the parameters for a system command.	
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 the statement up 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 5100 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 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.
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.

Error Message	Cause	User's Response
NOT FOUND: names	A)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 the 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 open a file on a tape unit that is already being used for input/output operations.	Close the data file or wait until the input/output operation is complete before issuing the command or the file open information again.
	A)OUTSEL command was issued, but printer output has been specified for a shared variable.	Retract the printer shared variable.
NOT WITH SUSPENDED FUNCTION	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 (ρρA).
	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 Messages

Error Messages

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. • A suspended function's header was changed. • A label was removed or changed on the suspended statement. • Statements were added to or erased from a suspended function.	Use the)SI or)SIV command to display the state indicator. Clear out the state indicator by entering → repeatedly.
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.	)SAVE the workspace,)CLEAR the active workspace, increase the number of symbols allowed by using the)SYMBOLS command, then)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.
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:* 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.
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.

Error Message	Cause	User's Response
WS FULL	One of the following conditions occurred:	Erase unnecessary objects. If there is still not enough space: – Partition the workspace into two or more workspaces with related functions. – Store data in a separate workspace or in a shared variable file. – Reprogram using smaller intermediate results. – Clear the state indicator with → if suspended functions exist. – Reduce the size of the symbol table. See note under)SYMBOLS.
	• A)COPY or)PCOPY command was issued, but the active workspace could not contain all the objects requested.	
	• The active workspace could not contain all the information required to build a defined function.	
	• The active workspace could not contain the intermediate results of an APL expression.	
	• The active workspace could not contain the final results of an APL expression.	
	• The active workspace could not contain the information required to do input/output operations.	
	• A workspace was written to tape with a)SAVE command, but the extra storage required when loading 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 fewer symbols allowed 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.

Error Messages

Error Message	Cause	User's Response
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.
WS TOO BIG	One of the following conditions occurred:	
	• An attempt was made to load a workspace stored with the)CONTINUE command into a 5100 with less internal storage.	Use a 5100 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.

Environment

ENVIRONMENT

The 5100 Portable Computer and associated units are designed for these environments:

	Operating Environment	Nonoperating Environment
Dry bulb temperature	60°-90° F (15°-32° C)	50°-105° F (10°-43° C)
Relative humidity	8%-80%	8%-80%
Maximum wet bulb temperature	73° F (23° C)	80° F (27° C)

You should not expose the machine to extreme temperatures for an extended time. For example, do not store the machine in the car trunk when the weather is very warm or very cold. If you must expose the machine to extreme temperatures, the machine should be acclimated to the operating environment before turning it on:

- If the machine was exposed to heat, allow it to cool enough so that you can place your hand on the surface without discomfort before turning it on.
- If the machine was exposed to cold and there is no frost or moisture on the external surfaces or visible on the internal parts, the machine can be turned on.
- If the machine was exposed to cold and there is frost or moisture on the external surfaces or visible on the internal parts, acclimate the machine for 8 hours after the frost and moisture disappears. This is to make sure all internal moisture evaporates before turning the machine on; otherwise, the machine may be damaged.

5100 SETUP PROCEDURE

After you have placed the 5100 where you intend to use it, make sure the red POWER ON/OFF switch (located on the front panel) is in the OFF position. Plug the power line into a *grounded* electrical outlet.

Note: For proper operation, the 5100 must be plugged into a grounded outlet.

Set the POWER switch to ON, and be sure that the fan is operating:

- If your machine location is not too noisy, you should hear the fan motor operating.
- If you are not sure, hold a light piece of paper near the air intake on the back of the machine. The loose end of the paper should be pulled toward the machine.

If the fan does not appear to be operating, check your power outlet. If it is OK, set the POWER switch to OFF and call for service. Do not continue with these instructions.

If the fan is operating, wait for about 20 seconds and your 5100 will be ready for operation.

APL Checkout Procedure

☐ 1. After power has been on 20 seconds, the display screen should show:

```
CLEAR WS
      _
```

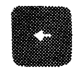
The underline (cursor) flashes on and off.

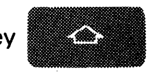
If the display screen does not show the above information, check the following top panel switches:

a. Turn the BRIGHTNESS control to get the best character definition.
b. Set the DISPLAY REGISTERS switch to the NORMAL position.
c. Set the L32 64 R32 switch to the center (64) position.
d. Set the BASIC/APL switch (combined machines only) to the APL position.
e. If information displayed is not as shown above, press the RESTART switch. This recycles a portion of the power-on sequence. If the information displayed is still not as shown above (after the 20-second delay), call for service.

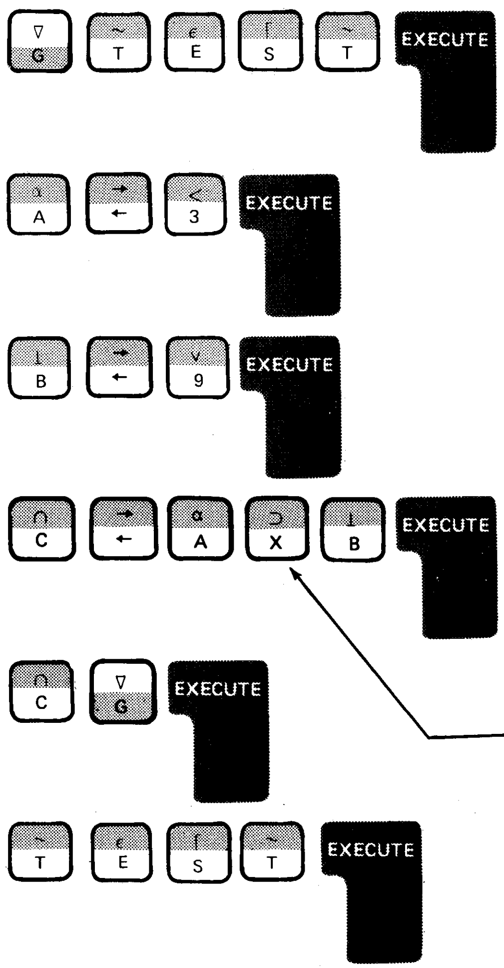
☐ 2. If the display screen does not show the correct results in the remaining steps of this procedure, press RESTART once, go back to step 1 and try again. If the correct result is still not shown, call for service.

Enter the data shown by the key drawings below. The data will be displayed as the keys are pressed.

If you make a keying error, you can press the backspace key  (above EXECUTE) to backspace the cursor, then press the correct key.

Where the bottom portion of the key is shown shaded, hold the shift key  down while you press the character key. (Enter the unshaded character.)

Press the following keys in sequence line by line:



The display now shows:

```
CLEAR WS
      ∇TEST
[1]   _
```

Be sure to use the multiply key and not the alphabetic X.

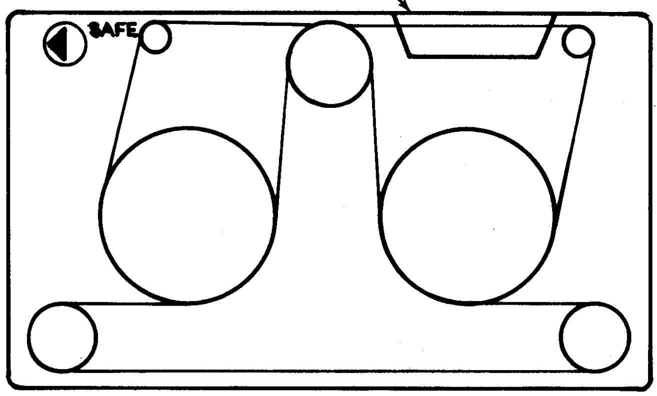
☐ 3. Below the lines of the test program that you just entered, the answer of 27 will be displayed (the program multiplies 3 times 9):

```
      ∇TEST
[1]  A←3
[2]  B←9
[3]  C←A×B
[4]  C∇
      TEST
27
      _
```

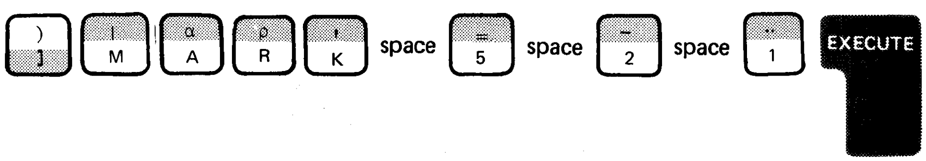
☐ 4. If you checked out the tape operation under the BASIC checkout procedures, insert the tape cartridge into the 5100 and go to step 6. Remove an unused or scratch tape cartridge from its package. Check that the arrow is pointing away from the word SAFE as shown in the illustration. Insert a coin or screwdriver into the slot if you must turn the triangular arrow away from the word SAFE.

Note: Do not use any prerecorded tape cartridges that were shipped with your machine.

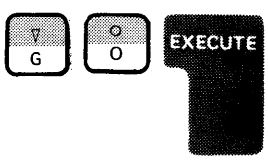
This edge goes into machine first.



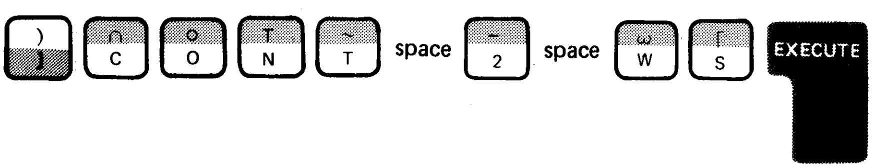
☐ 5A. Insert the tape cartridge into the 5100 (metal bottom down), and press it in until it seats firmly. Then press the following keys (you must leave a space before each number):



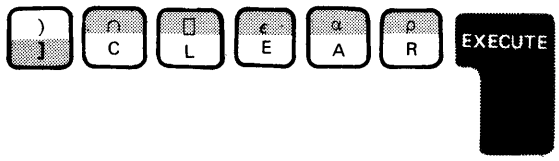
5B. The previous step initialized the tape to hold information. If a message of MARKED is displayed, go to step 6. If a message of ALREADY MARKED is displayed, the tape is already marked. To re-mark the tape, press:



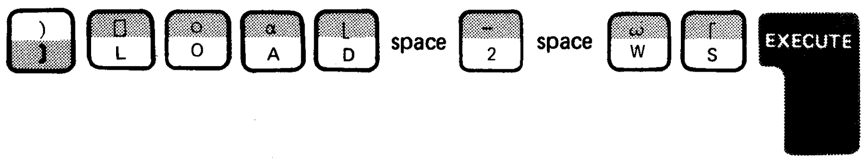
☐ 6. Press the following keys:



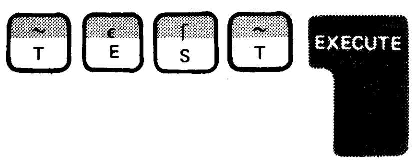
☐ 7. The last step wrote the program onto tape, but it is still recorded in the storage workspace. To prove the program can be read from tape, the program must be erased from the workspace. To do this, press the following keys:



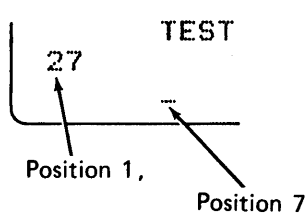
☐ 8. To read the program from tape into the 5100, press the following keys:



☐ 9. When LOADED 1002 WS is displayed, press these keys:



The display screen should again show:



This completes the APL checkout procedure.

☐ 10. Check to see that you received all the items on the Contents Checklist.

If the words above the top row of numeric keys are labeled on the left with:

APL , you have an APL machine without the communications feature.

BASIC , you have a BASIC machine without the communications feature.

BASIC
APL } , you have a combined machine.

COMM
BASIC
APL } , you have a combined machine with the communications feature.

COMM
APL } , you have an APL machine with the communications feature.

COMM
BASIC } , you have a BASIC machine with the communications feature.

If you have not checked out BASIC on a combined machine, set the BASIC/APL switch to the BASIC position, press RESTART, and go to the *BASIC Checkout Procedures* in Appendix C of the *IBM 5100 BASIC Reference Manual,* SA21-9217. If you already did the BASIC checkout procedures, continue with step 11.

☐ 11. If the auxiliary tape unit is to be installed, unpack the tape unit and proceed to the *Auxiliary Tape Unit Setup Procedure* which follows. After installing the auxiliary tape unit, return to step 12.

☐ 12. If the printer is to be installed, unpack the printer and proceed to *Printer Setup Procedure,* which comes later in this appendix. After installing the printer, return to step 13.

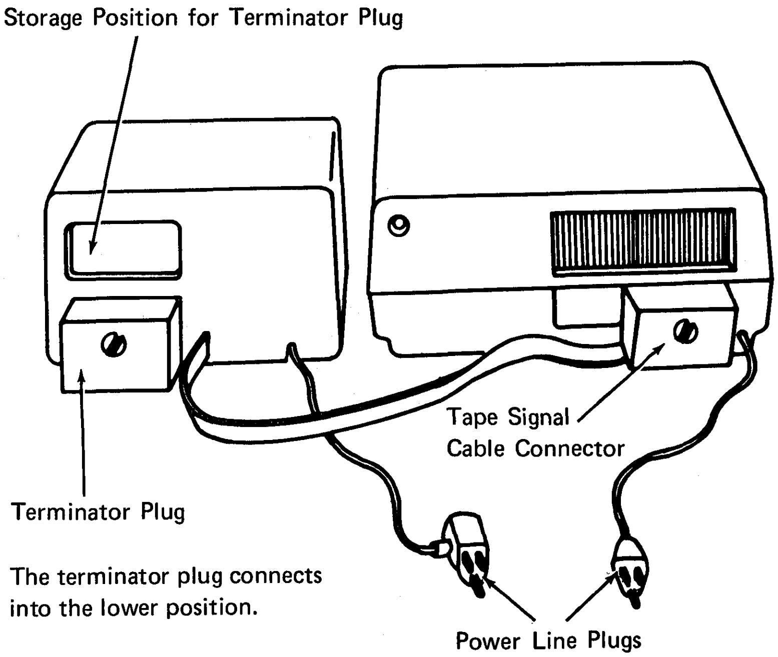
☐ 13. If your 5100 is equipped with any other feature, use the manual supplied with that feature to set up and check out the feature, then return to step 14 in this manual.

☐ 14. When the preceding devices or features are installed, or if none are, begin reading the *IBM 5100 APL Introduction* to learn how to operate your 5100.

AUXILIARY TAPE UNIT SETUP PROCEDURE

☐ 1. Set the 5100 and auxiliary tape unit power switches to OFF.

☐ 2. Remove the shipping tape from the signal cable (flat cable) and connect the signal cable into the back of the 5100. Make sure the connector fits squarely. Turn the knob in a clockwise direction until the connectors fit together firmly:



☐ 3. Check that the terminator plug is in place on the rear panel as shown in the preceding diagram.

☐ 4. Remove the shipping tape from the power line and plug the power line into a grounded electrical outlet.

☐ 5. Set the auxiliary tape unit POWER switch to ON, and be sure that the fan is operating.

a. If your location is not too noisy, you should hear the fan motor operating.
b. If you are not sure, hold a light piece of paper near the air intake on the left side of the tape unit. The loose end of the paper should be pulled toward the tape unit.

If the fan does not appear to be operating, check your power outlet. If it is OK, set the POWER switch to OFF and call for service. Do not continue with these instructions.

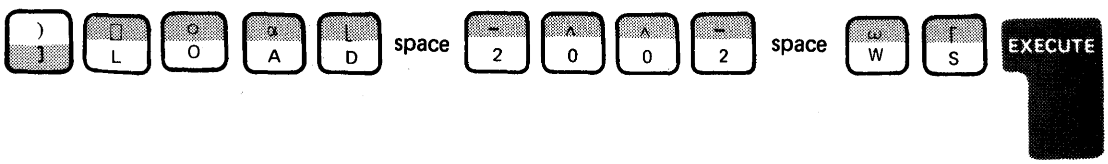
☐ 6. Set the 5100 POWER switch to ON and continue to the checkout procedure.

Tape Unit Checkout Procedure

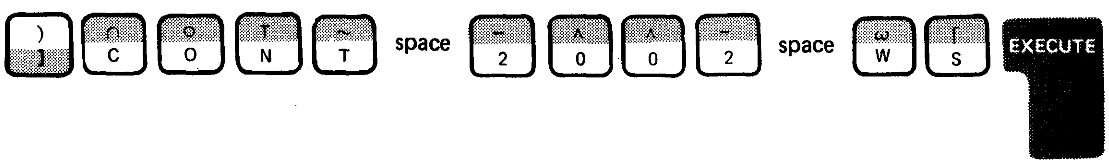
Note: The following steps assume you are using the same cartridge that you used to check the 5100. If you are not, write any program onto the cartridge in the auxiliary tape unit and read it back.

☐ 1. Insert a tape cartridge into the auxiliary tape unit after checking that the arrow is pointing away from the word SAFE.

☐ 2. Press the following keys to read in the program that was stored on tape during the 5100 checkout procedure:



☐ 3. After the message LOADED 2002 WS appears on the display screen, press the following keys:



☐ 4. The message CONTINUED 2002 WS appears on the display to verify that the program was written back to tape and was checked by the 5100.

This completes the checkout procedure for the auxiliary tape unit.

Return to step 12 of the 5100 checkout procedure.

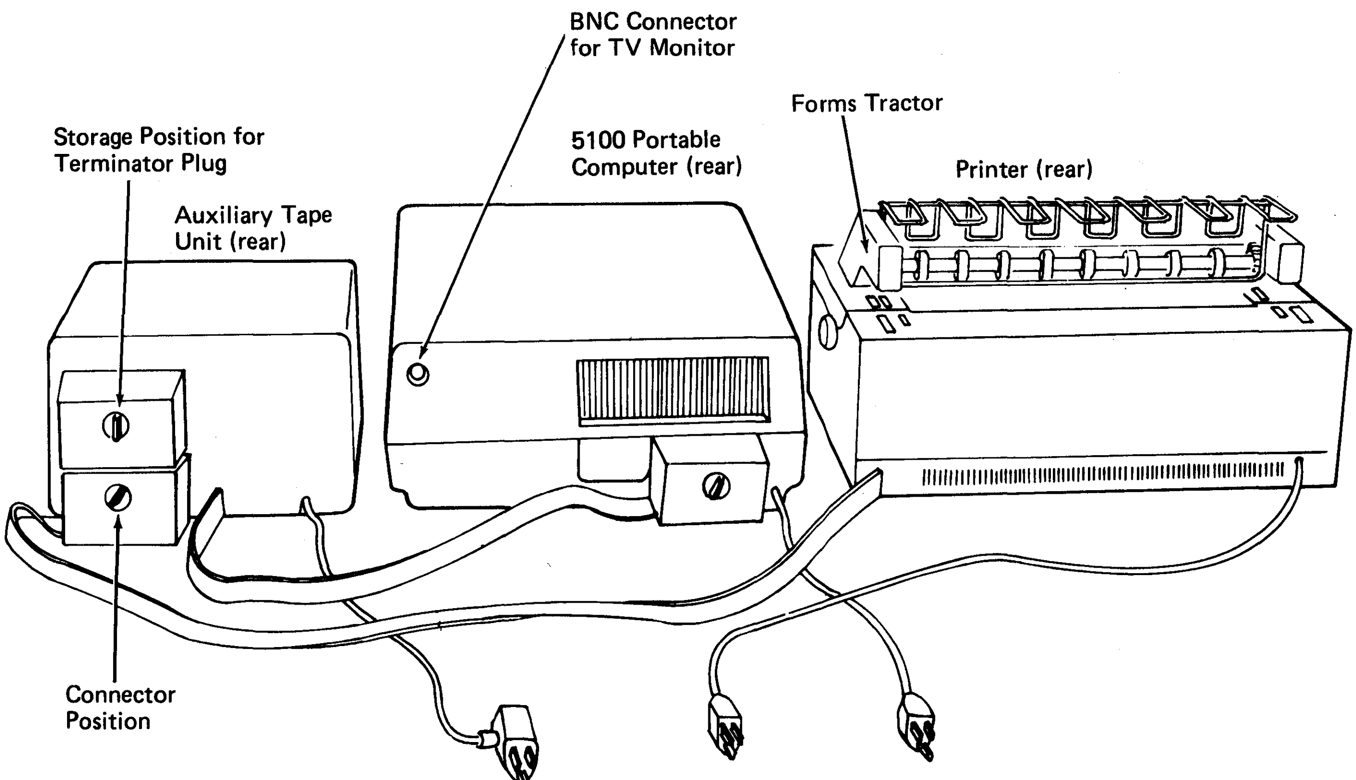
Setup

PRINTER SETUP PROCEDURE

☐ 1. Set the 5100, 5103, and 5106 (if attached) POWER switches to OFF.

☐ 2. If you have an auxiliary tape unit, remove the terminator plug from the bottom position and insert it into the top position (storage position).

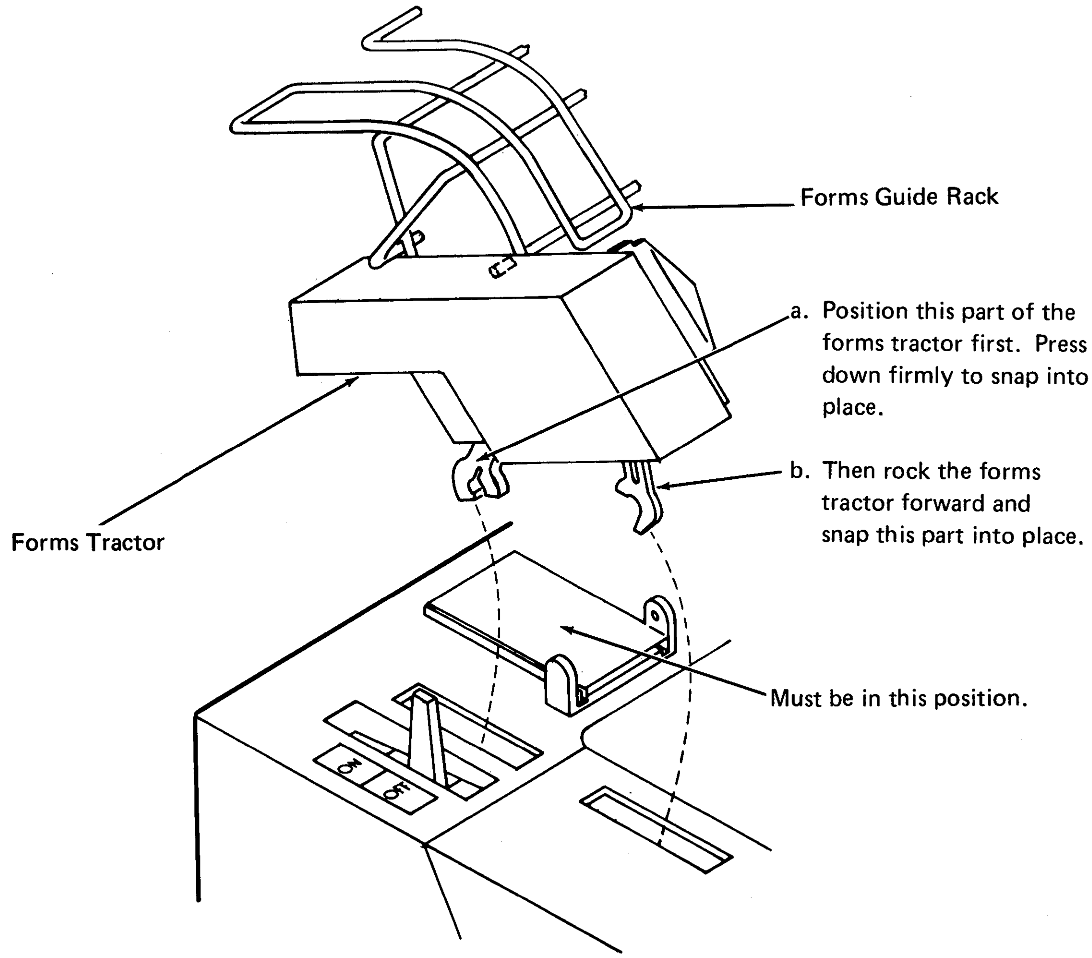
☐ 3. Remove the shipping tape from the printer signal cable (flat cable) and connect the signal cable to the back of the auxiliary tape unit, if it is attached, or to the back of the 5100. Make sure the connector fits squarely. Turn the knob in a clockwise direction until the connectors fit together firmly:



The printer signal connects to the back of the 5100 if the auxiliary tape unit is not attached.

☐ 4. Remove the shipping tape from the printer power line and plug the power line into the back of the auxiliary tape power plug or into a grounded electrical outlet.

☐ 5. Unpack the forms tractor and set it in place on top of the printer as shown in the drawing.

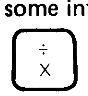


☐ 6. Insert paper in the printer. Use the printer information in this manual if you need help in inserting the paper (see Chapter 10).

☐ 7. Set both the printer and 5100 POWER switches to ON and continue on to the checkout procedure.

Printer Checkout Procedure

Press several alphameric keys to display some information. Then, hold down the CMD key 

and press the key 

below Copy Display on the command word strip. The printer will provide a copy of the information on the display screen.

Return to step 13 of the 5100 checkout procedure.

Appendix B. 5100 APL Character Set and Overstruck Characters

Overstruck characters are formed by entering one character, backspacing, and entering the other character. The 5100 APL character set consists of all the characters represented on the 5100 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	⍋	∆ 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 C. Atomic Vector

The following chart shows the character, the character name, and the index of that character in the 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
[	LEFT BRACKET.	15
]	RIGHT BRACKET	16
(	LEFT PARENTHESIS.	17
)	RIGHT PARENTHESIS	18
;	SEMICOLON	19
/	SLASH	20
\	BACK SLASH.	21
←	LEFT ARROW.	22
→	RIGHT ARROW	23
	RESERVED.	24
	RESERVED.	25
¨	DIERESIS (UPPERSHIFT 1)	26
+	PLUS.	27
−	MINUS	28
×	TIMES	29
÷	DIVIDE.	30
*	STAR.	31
⌈	MAXIMUM	32
⌊	MINIMUM	33
∣	RESIDUE	34
∧	AND	35
∨	OR.	36
<	LESS THAN	37
≤	LESS THAN OR EQUAL.	38
=	EQUAL	39
≥	GREATER THAN OR EQUAL	40
>	GREATER THAN.	41
≠	NOT EQUAL	42

Character	Character Name	Index (⎕IO←1)
⍺	ALPHA	43
∊	EPSILON	44
⍳	IOTA.	45
⍴	RHO	46
⍵	OMEGA	47
,	COMMA	48
!	SHRIEK (EXCLAMATION).	49
⌽	REVERSAL.	50
⊥	ENCODE (BASE)	51
⊤	DECODE (REPRESENTATION)	52
○	CIRCLE.	53
?	QUERY	54
∼	NOT	55
↑	UP ARROW.	56
↓	DOWN ARROW.	57
⊂	SUBSET.	58
⊃	RIGHT SUBSET.	59
∩	CAP	60
∪	CUP	61
_	UNDERSCORE.	62
⍉	TRANSPOSE	63
⌶	⌶-BEAM.	64
∘	NULL (SMALL CIRCLE)	65
⎕	QUAD.	66
⍞	QUAD QUOTE.	67
⍟	LOG	68
⍲	NAND.	69
⍱	NOR	70
⍝	LAMP-COMMENT.	71
⍋	GRADE UP.	72
⍒	GRADE DOWN.	73
⊖	OVERSTRUCK CIRCLE-HYPHEN.	74
⌿	OVERSTRUCK SLASH-HYPHEN	75
⍀	OVERSTRUCK BACKSLASH-HYPHEN . . .	76
⌹	MATRIX DIVIDE	77
⍕	FORMAT.	78
⍎	EXECUTE	79
&	AMPERSAND	80
@	AT.	81
#	POUND	82
$	DOLLAR.	83
	UNUSED.	84
T∆	TRACE (T DELTA)	85
S∆	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

Character	Character Name	Index (⎕IO←1)
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
∆	DELTA	113
$\underline{A}$	A-UNDERSCORE	114
$\underline{B}$	B-UNDERSCORE	115
$\underline{C}$	C-UNDERSCORE	116
$\underline{D}$	D-UNDERSCORE	117
$\underline{E}$	E-UNDERSCORE	118
$\underline{F}$	F-UNDERSCORE	119
$\underline{G}$	G-UNDERSCORE	120
$\underline{H}$	H-UNDERSCORE	121
$\underline{I}$	I-UNDERSCORE	122
$\underline{J}$	J-UNDERSCORE	123
$\underline{K}$	K-UNDERSCORE	124
$\underline{L}$	L-UNDERSCORE	125
$\underline{M}$	M-UNDERSCORE	126
$\underline{N}$	N-UNDERSCORE	127
$\underline{O}$	O-UNDERSCORE	128
$\underline{P}$	P-UNDERSCORE	129
$\underline{Q}$	Q-UNDERSCORE	130
$\underline{R}$	R-UNDERSCORE	131
$\underline{S}$	S-UNDERSCORE	132
$\underline{T}$	T-UNDERSCORE	133
$\underline{U}$	U-UNDERSCORE	134
$\underline{V}$	V-UNDERSCORE	135
$\underline{W}$	W-UNDERSCORE	136
$\underline{X}$	X-UNDERSCORE	137
$\underline{Y}$	Y-UNDERSCORE	138
$\underline{Z}$	Z-UNDERSCORE	139
$\underline{\Delta}$	DELTA-UNDERSCORE	140
0	0	141
1	1	142
2	2	143
3	3	144

Character	Character Name	Index (⎕IO←1)
4	4	145
5	5	146
6	6	147
7	7	148
8	8	149
9	9	150
.	PERIOD.	151
¯	OVERBAR	152
	BLANK	153
'	QUOTE	154
:	COLON	155
∇	DEL (FN DEF CHAR)	156
	CURSOR RETURN	157
	END OF BLOCK(CANNOT BE DISPLAYED)	158
	BACKSPACE	159
	LINEFEED.	160
⍫	PROTECTED DEL	161
	UNUSED.	162
	UNUSED.	163
	UNUSED.	164
	UNUSED.	165
	UNUSED.	166
	UNUSED.	167
	UNUSED.	168
	LENGTH OF Z-SYMBOL TABLE.	169
⌹	O-U-T FOR COMMUNICATION TAPE. . .	170
¬	LOGICAL NOT	171
"	DOUBLE QUOTE.	172
%	PERCENT	173
⍙	PROTECTED DELTA	174
⊙	BULLS EYE	175
Ä	A UMLAUT.	176
Ö	O UMLAUT.	177
Ü	U UMLAUT.	178
Å	ANGSTROM.	179
Æ	AE DIAGRAPH	180
₧	P SUB T	181
Ñ	N TILDE	182
£	POUND STERLING.	183
¢	CENT.	184
Õ	O TILDE	185
Ã	A TILDE	186

Note: The remaining elements (187-256) are unused.

Atomic Vector

Appendix D. 5100 APL Compatibility with IBM APLSV

The 5100 APL system differs from the IBM APLSV system primarily because the 5100 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 5100 active workspace is generally smaller than APLSV active workspace. It is further limited by the shared variable processor which uses 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 1- to 5-digit number that specifies the device and file number where a workspace is to be)SAVE'd or)LOAD'ed. If the number is less than 4 digits, it is only the file number; device 1 is assumed; otherwise, the high-order 1 or 2 digits is the device number.

- The)LOAD,)COPY,)PCOPY commands require the library (device/file) number and workspace ID parameters. The)DROP command requires the library (device/file) number and if the specified file is a stored workspace file, the workspace ID parameters. These requirements protect the user from inadvertently destroying his or her saved workspaces.

- The following commands are not supported because they apply only to multi-terminal 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 5100 processor and its input/output devices:

)MARK	–	To format tape files
)OUTSEL	–	To specify which transactions are to be printed
)REWIND	–	To rewind the tape unit
)MODE	–	To select communications mode
)PATCH	–	To load an IMF or Tape Recovery program into storage from an IBM-supplied tape

- 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)COPY'ed, or)LOAD'ed into a 5100 with a smaller active workspace.

- Since the 5100 system is not in a communications environment, the RESEND message will not occur.

-)SAVE and)LOAD have to be 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 attempting anything other than)CLEAR after a system error:

  NOT WITH SYSTEM ERROR

- Saved workspaces are not time-stamped and dated because that 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

Compatibility

- 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 in Chapter 2).

- The following system messages have been added for the new system commands and input/output operations:

  ALREADY MARKED
  DEVICE NOT OPEN
  DEVICE TABLE FULL
  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 DEVICE

- The shared variable processor on the 5100 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.

  Since 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 not supported:

  ⎕TS – Time stamp
  ⎕AI – Accounting information
  ⎕TT – Terminal type
  ⎕UL – User list
  ⎕DL – Delay

- The I-beam functions have been replaced with system variables or system functions and are not supported.

- Catenation using semicolons has been replaced by format, but it is still supported on the 5100.

- 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 for ⎕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 5100 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 execute 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 390 when the 5100 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 5100 will insert a quote if an uneven number of quotes is entered.

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: Using a shared variable to add information to an existing data file.

alphameric keys: The keys on the left side of the keyboard that are arranged similar to 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: To display output without the cursor returning to the next line.

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 one byte of storage.

character constant: 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 (file type 01, 02, or 08) where data was stored using a shared variable.

defective record: A 512-byte block of storage on tape that cannot be read.

device/file number: Specifies the tape unit and file to be used when doing tape input or output operations.

dual-language machine: A 5100 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.

exchange data file: The data in the file is in the exchange data format.

exchange data format: The data consists of all character scalars or vectors.

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 tape. The tape is formatted into files by using the)MARK command.

file ID: The name of a file on tape. If the file contains a stored workspace, the file ID is the same as the stored workspace ID.

file number: The files on tape are sequentially numbered starting from one.

file type: Identifies the type of data stored in a file.

function body: Consists of the statements within a user-defined function. These statements determine the operation(s) performed by the function.

function definition: Defining a new function (a user-defined function) to solve a problem.

function definition mode: The mode that is used when defining or editing user-defined functions. The ∇ symbol is used to change the mode of operation. Contrast with execution mode.

function header: Defines the function name, number of arguments, local names, and whether or not the function will have an explicit result.

general exchange data file: The data in the file is in the general exchange format.

general exchange data format: The data consists of all character scalars or vectors.

global names: The value or function associated with these names can be used within or outside of 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 names.

identity element: The value that generates a result equal to the other argument of a function.

IN 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 is the lowest value of an index. The index origin is set to 1 in a clear workspace and can be changed by using the ⎕IO system variable.

input: Information entered from the keyboard or read from tape using a shared variable.

input line: Consists 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: The data in the file is in the internal data format.

internal data format: The format in which the data is stored in the 5100.

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 where data is stored for future use.

local name: A name that is contained in the function header and has a value only 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.

mixed function: The results of mixed functions 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 that are arranged similar to a calculator keyboard.

object: A user-defined function or variable name.

operators: Have as their arguments dyadic primitive scalar functions. These arguments are applied to arrays in a specified way.

OUT operation: Using a shared variable to write information into a data file.

output: The results of statements processed by the 5100.

overstruck character: A character formed by entering one character, backspacing, and entering another character. Only certain combinations of characters can form overstruck characters.

parameter: (1) Information needed by a system command (such as device/file number). (2) Information required to open a data file or 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.

plane: The coordinates of an n-rank array other than the rows and columns.

primitive function: The functions that are part of the APL language (such as , + – ÷ x).

PRT 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: Assigned to a shared variable after a PRT, OUT, or ADD operation. This code indicates whether or not the operation was successful.

scalar: A single data item that does not have a dimension (ρ ρ = 0).

scalar function: The results of the scalar functions are the same shape as the arguments. The function is applied to corresponding elements in the arguments.

scale: An integer representing the power of ten in scaled representation.

scaled representation: Stating a value in a convenient range and multiplying it by the appropriate power of ten.

scroll: Moving the information on the display screen up or down.

shape: The length of each coordinate of an array.

shared variables: A variable shared by the active workspace and the tape or printer. Used to transfer data during IN, OUT, ADD, or PRT 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: 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 (S∆): Stopping execution of a user-defined function before the execution of a specified statement.

stop vector: Specifies the statements when using stop control.

stored workspace: The contents of the active workspace stored on tape.

suspended: See suspended function.

suspended execution: See suspended function.

suspended function: Execution has stopped because of an error condition, ATTN being pressed, or stop control being used.

system commands: Are used to manage the active workspace and tape or printer operations.

system functions: Are used to change or provide information about the system.

system operation: Processing input data.

system variable: Provides controls for the system and information about the system to the user.

trace control (TΔ): Displaying the results of specified statements during the execution of a user-defined function.

trace vector: Specifies the statements when using 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 defined using the primitive functions. See function definition mode.

variable name: A name associated with the value of a variable.

variables: Data stored in the 5100.

vector: An array with one dimension ($\rho\rho = 1$).

workspace: See active workspace.

workspace available: The amount of unused storage (number of unused bytes) in the active workspace.

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

IBM 5100 **SA21-9213-2**
APL Reference Manual

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page *Comment*

I would like a reply. ☐

Name ______________________________

Address ______________________________

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

SA21-9213-2

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

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 5100 APL Reference Manual Printed in U.S.A. SA21-9213-2

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

SA21-9213-2

IBM 5100 APL Reference Manual Printed in U.S.A. SA21-9213-2

IBM

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)