Georgia
Institute
of
Technology

RICH ELECTRONIC COMPUTER CENTER /(404) 873-4211/ATLANTA, GEORGIA 30332

A P L

for the

Burroughs B5500

Time Sharing System

August 1971

A P L

for the

Burroughs B5500

Time Sharing System

August 1971

ACKNOWLEDGMENT

Georgia Tech is fortunate and very grateful to have received this implementation of APL/B5500 (A Programming Language) from the Computer Center and the Computer Science Department at the University of Washington. With their permission, we are republishing their manual with only a slight amount of editing. Appendix G was developed at Georgia Tech and provides a useful comparison between APL/B5500 and APL\360.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

1. Introduction

APL/B5500 is an interpreter for a conversational programming language implemented on the Burroughs B5500 computer at the University of Washington. The language is patterned after APL/360(1), an implementation of "Iverson Notation"(2). The APL/B5500 interpreter provides line by line evaluation of APL statements as input by a programmer at a remote teletype station. The interpreter provides both "desk calculator" and "stored program" capabilities. The large number of special purpose operators operate on the basic data elements to allow concise expression of mathematical and manipulative constructs. The basic data elements of APL are constants and identifiers whose values may be characters or numbers. Data elements may take scalar, vector, and array structures.

The APL programmer may define "stored programs" which group APL statements together to be executed at a later time. The stored programs may be either the standard "subroutine" type or the "function" type. The ALGOL 60 notion of local and global quantities is preserved with respect to APL stored programs.

A comprehensive set of commands allows communication with the APL interpreter monitor. These commands allow maintenance of data areas and specification of run parameters.

Appendix A contains a set of Backus-Naur syntax equations which define the APL/B5500 programming language. Each syntax equation is written in terms of metalinguistic variables (enclosed in the broken brackets "<" and ">") and terminal symbols (capital letters or special symbols). The metalinguistic variable to be defined appears on the left side of the symbol "::=" (read as "is defined as"). The right side of each syntax equation contains combinations of metalinguistic variables and terminal symbols, and serves to define the variable on the left. The symbol "|" (read as "or") separates multiple definitions. The braces "{" and "}" enclose an English language definition, used whenever definition in terms of metalinguistic variables and terminal symbols is impossible or unenlightening.

As the elements of the language are explained in this manual, the metalinguistic variables will be enclosed in broken brackets wherever they occur in the text. Broken brackets may be ignored in reading the manual since they serve to indicate that the item appears in the syntax equations of Appendix A.

1.1 Gaining access to APL/B5500

A user logs onto the Time Sharing System by entering his usercode and password when requested. He then can call APL/B5500 by entering the command

CALL APL←

---

(1) Iverson, K. E. and A. D. Falkoff, "APL/360: User's Manual", International Business Machines Corporation, 1968.

(2) Iverson, K. E., A PROGRAMMING LANGUAGE, John Wiley and Sons, Inc., New York, 1962.

1.2 User Communication with APL

The user is provided with input and output buffers which are 200 characters long. On output, APL "folds" the buffer across the terminal in pieces determined by the current WIDTH (Section 3.3), indenting two spaces when a row is continued to the next line.

On input, the user terminates each logical transmission with a left arrow ("←"). In order to transmit more than one teletype line (72 characters), the user depresses the line-feed and carriage-return keys on the teletype and continues typing.

After a line has been sent (indicated by the "←"), the following sequence of events occurs:

a) When APL has queued the input line for processing, it causes the teletype carriage to return.

b) The line is processed and resulting output (if any) is written on the teletype.

c) APL indents six characters on the next line and waits for the next input.

1.2.1 Use of the Break Key

Hitting the BREAK key on the teletype while APL is writing causes termination of the output. Entering a left arrow while APL is executing but is not typing or reading has the effect of stopping the executing process. In either case, if a stored program is executing, it is placed in suspended mode. Throughout the remainder of this manual, a reference to hitting the "break key" means hitting the BREAK key or entering a left arrow, whichever is appropriate.

1.3 The APL/B5500 Character Set

The character set of APL/B5500 is a subset of the characters available on the model 33 teletype. These characters are the <visible string character>s, the ", and the single space. The <visible string character>s consist of the <letter>s: ABCDEFGHIJKLMNOPQRSTUVWXYZ, the <digit>s: 0123456789, and a number of <special symbol>s: ().,:;+-=/$#*@%&[] ←.

1.4 Reserved Words and Symbols

APL reserves several words and symbols to specify the special-purpose operators. They are listed in Appendix D and may not be used as <identifier>s.

1.5 Use of Delimiters

Reserved words and <identifier>s (see Section 4.1) must be preceded and followed by at least one delimiter. Delimiters are the <special symbol>s listed in 1.3, and any number of <single space>s. The start of a line is also considered a delimiter.

The remaining sections serve to describe an <apl program> which begins with a <login> and ends with a <logout>. In between, it may include <monitor command>s (Section 3), and may operate on <data element>s (Section 4) in <basic statement>s (Section 5) using APL operators (Section 6) or user-defined stored programs (Section 7). Section 8 describes some miscellaneous APL constructs such as subscripting, defines the various modes of operation available to the user, and describes the error messages the user may receive.

2. Starting and Ending an APL Run

2.1 The <login>

The user activates APL/B5500 from the terminal according to normal job initiation procedures at his installation. APL responds by typing

```
APL/B5500 UW COMPUTER SCIENCE # (version date)
LOGGED IN (current date and time)
```

2.2 The <logout>

To terminate an APL session, the user types

a))ØFF or
b))ØFF DISCARD

In either case, the user is disconnected from APL and APL types 'END OF RUN". Option a) causes the current active workspace to be saved in "APLIBRY"/<user code> (Section 3.1); in case b), the active workspace is discarded. APL then goes through the normal end of job process.

3. <monitor command>s and the Workspace

The workspace allows the user to save his work from session to session, and the <monitor command>s allow him to interrogate and maintain the workspace, specify <run parameter>s, and create and maintain workspace libraries.

3.1 The Workspace

Defined <identifier>s and their values, definitions of stored programs, and <run parameter>s are kept in a "workspace". The workspace associated with an active run is called the "active workspace". Successful <login> causes activation of the last active workspace or, if there is none, creation of a new workspace. Active workspaces may be saved in libraries. These libraries are inactive, but may be activated using certain <library maintenance><command>s.

3.2 Workspace Interrogation and Maintenance <command>s.

Command	APL Response
)CLEAR	Discards the active workspace except for the <run parameter>s. Not allowed while in suspended mode (Section 8.4.4).
)ERASE <identifier list>	Removes each global variable and its value, or stored program and its definition named in the <identifier list> from the active workspace. The <identifier list> is a list of <identifier>s, separated by one or more blanks. Not allowed while in suspended mode (Section 8.4.4).
)VARS	Lists the names of all the variables and stored programs in the active workspace. Stored program names are followed by the characters "(F)".
)FNS	Lists the names of all stored programs in the active workspace.
)SI	Lists in order the names of stored programs which have been suspended during execution (Section 8.4.4).
)SIV	Lists in order the names of stored programs which have been suspended during execution, as well as the names of variables local to these stored programs (Section 8.4.4).

)ABØRT — Terminates all suspended stored programs and returns directly to execution mode (Section 8.4).

)STØRE — When in suspended mode (Section 8.4.4), causes APL to store into the active workspace the values of global variables which have been changed (they are not stored until a stored program is terminated normally).

When the <command> calls for a list (VARS, FNS, SI, SIV) and no elements exist in the requested category, APL types "NULL".

3.3 <run parameter> <command>s

These <command>s allow the user to select values for certain parameters or to ascertain the current values of them. To change the value, the user types a ")", followed by the parameter name, followed by a number (except for SYN and NØSYN). If no number is given, APL responds with the current value of the parameter. Where an integer is required, the number given is rounded to an integer. The following table gives the <command>, the value the parameter is given when a new workspace is initialized, any restrictions on the value of the number to be given, and the use of the parameter.

Command	Initial Value	Restrictions	Usage
)ØRIGIN <integer>	1		The starting index of arrays. Also the starting point for index and random number generation.
)WIDTH <integer>	72	between 10 and 72, inclusive	The width of a teletype line for output. Lines longer than WIDTH will be "folded over".
)DIGITS <integer>	9	between 0 and 12, inclusive	The maximum number of digits written after the decimal point on output. (APL removes non-significant zeroes.)
)SEED <integer>	59823125	absolute value is taken; for best results, should take previously-held values.	The "seed" for the random number generator; value changes with each call to the random number generator.
)FUZZ <number>	10^{-11}	absolute value is taken	To counter truncation error and use in comparisons. A is considered equal to B if $\lvert A-B \rvert \leq$ FUZZ $\times \lvert B \rvert$.

Command	Initial Value	Restrictions	Usage
)SYN	on		Causes APL to check the syntax of each line of a stored program as it is being defined (Sec. 7). This is the default state.
)NØSYN	off		Turns off the syntax checking option during stored program definition.

3.4 <library maintenance> <command>s

These <command>s allow the user to save the active workspace or to activate previously saved workspaces, or parts thereof. Library files are saved under the user's B5500 TSS <USERCODE>. The library name used takes the form of an <identifier> not more than six characters long. For)LOAD,)COPY, and)CLEAR, the message: "FILE NOT ON DISK" may occur with obvious meaning.

Note: For APL/B5500 under B5500 TSS, all APL disk files have a file-ID (i.e., <file-ID>/<multi-file-ID>) that starts with a "/". This is a convention used to distinguish APL files from CANDE files on other types of files. This may be easily changed and the use of this convention is really installation dependent.

a))SAVE <library name> <lock option> saves the current active workspace on disk under the generated <library name> provided that a file with the same name is not already on disk. Otherwise, the message "FILE ALREADY ØN DISK" is typed. The <lock option> may be "LØCK" or <empty>. If "LØCK" is used, no user may access the file unless he is logged in to APL under the same <user code> as that generated for <library name>.

b))LØAD <library name> loads the previously SAVEd workspace referenced by <library name> into the active workspace (subject to restrictions on LØCKed files). When a name in the library workspace matches a name in the active workspace, the corresponding item is not LØADed unless the name in the active workspace is for a variable and the name in the library file is for a stored program. When an item is not LØADed, APL responds with an appropriate message.

c))CØPY <library name> <copy name> copies the value of the variable or definition of the stored program named by the <copy name> (an <identifier>) from the library file referenced by <library name> into the active workspace (subject to the restrictions on LØCKed files.) The rule for replacing matching variables is the same as for LØADing.

d))CLEAR <library name> removes the library file referenced by the generated <library name> from the disk.

e))FILES lists the <library name>s of the user's APL workspace library files which have been saved and are present on disk.

3.5 The <buffer edit>

APL retains each user's last input. This <command> allows the user to change the last line of input without retyping the entire line. It is given by typing

)" <line edit> where <line edit> takes the form
"<search string>
<search string> "<insert string> or <empty> . APL responds as follows:

a) The last input line is edited according to the specifications given in line edit --the rules for which are discussed in Section 7.2.3.

b) The resulting line is typed on the teletype.

c) The new line is processed as if the user had just typed it in.

4. <data element>s and Data Structures

The basic <data element>s of APL are <identifier>s and <constant>s. These in turn may take on various structures as given below. Exact syntax for <data element>s is given at the end of Appendix A.

4.1 <identifier>s

An APL <identifier> is a combination of <letter>s and <digit>s, beginning with a letter (except for the reserved words listed in Appendix D). Only the first seven characters of an <identifier> are significant. <identifier>s may be assigned values via the <assignment statement> (Section 5.1.3). <identifier>s which have not been assigned values are "null" (Section 4.2.4).

4.2 <constant>s

A <constant> may be a <number>, a character <string>, or "null".

4.2.1 <number>s

For input, a <number> takes the following form:
<integer> <decimal fraction> <exponent part> where all but one of the three entities may be omitted. An <integer> is a sequence of decimal

digits which may be preceded by a sign ("+" or <empty> for a positive number, "#" for a negative number--sorry, the "-" has another use). A decimal fraction is a "." followed by a sequence of decimal digits. The exponent part is an "E" or "@" followed by an <integer> (except that here a "-" may be used instead of "#" if desired). The <exponent part> must begin with "@" if it stands alone since the number would otherwise be interpreted as an <identifier>. The "E" or "@" means "times ten to the power given by the following integer"--e.g. 2.3@-2 means .023, @-2 means .01. Output of <number>s is governed by the value of DIGITS (Section 3.3). Extraneous zeros are omitted, and "-" is used instead of "#".

Examples: 47 47.2 47.2@3 .341 .341@#11 #47 #47.2E+3 #.341@-11 #@45

There is no distinction in APL between integers and real numbers. They are both carried in B5500 "floating point form". Numbers whose significant digits (disregarding sign, decimal point, and exponent) do not exceed 549755813887 are carried to full precision. Absolute values of numbers must be less than about 4.314@68 to stay within the limits of the B5500. Numbers with absolute value less than 10@-47 are converted to zero.

4.2.2 Boolean Elements

A <constant> is sometimes used in a "logical" context and is considered to be "true" (if it is 1) or "false" (if it is 0). However, these Boolean elements may be used in arithmetic computations regardless of previous usage. Using <number>s other than 0 or 1 in Boolean operations yields an error.

4.2.3 Character <string>s

For input, a character <string> is a sequence of <letter>s, <digit>s, <special symbol>s, and <space>s enclosed in quotation marks. A quote within a string is indicated by a double quote--"". A null string is indicated by enclosing nothing in quotes--that is "". On output, the string is typed without the enclosing quotes. For example, if the user types

```
"HE SAID, ""2+2=4""", APL would interpret the <string> as
 HE SAID, "2+2=4".
```

4.2.4 The Null Element

<identifier>s which have not been assigned values and the results of certain operations have the special value "null"--that is, they have no value. Usually, attempting to apply an operator to a null element yields an error. However, it has a special meaning for certain operators (see Section 6).

4.3 Scalars, Vectors, and Arrays

A scalar is a single <number>. A vector is a one-dimensional list of <number>s or a character <string>. Arrays are an extension

of vectors with up to 31 dimensions. The number of dimensions is called the rank (e.g. an array of rank 2 is a matrix; an array of rank 1 is a vector).

Every array must consist entirely of numbers or entirely of characters. In the former case, each number is one element of the array; in the latter case, each character is an element of the array (e.g. "45A2-*" is a vector of 6 elements, each of which may be accessed). There are no null elements in a non-null array. With one exception (Section 5.1.4.2 (b)), one-element arrays are treated as scalars.

4.3.1 Vectors

Vectors may be input directly, provided they do not exceed 200 characters. A numeric vector is input by typing a list of <number>s, separated by one or more blanks, and is treated as a <constant>--e.g.

5 7 #42 @-7

Character vectors are also <constant>s and are input by typing a character <string>. A <string>with one element is stored as a one-element vector rather than as a scalar. Elements may be added to vectors using the catenate operator (Section 6.5).

4.3.2 Arrays

Arrays of rank greater than 1 may not be input directly, but may be created using the restructure operator RHØ (Section 6.5) or may result from other operations. Subsets of named arrays (that is, arrays which have been assigned to <identifier>s via <assignment statement>s--Section 5.1.3) may be accessed by subscripting (Section 8.3). The exact structure of an array is given by its "dimension vector" which has as many elements as the rank of the array. In other words, if the dimension vector is 2 3, the array is a matrix with 2 rows and 3 columns; if the dimension vector is 4 5 6, it is an array of 4 submatrices, each of which has 5 rows and 6 columns.

5. The <basic statement>

The <basic statement> does all the computing in APL. It is a sequence of <constant>s, special symbols, <identifier>s, and <function name>s in some syntactically correct order.

Formation of <constant>s and <identifier>s is reviewed at the end of Appendix A, and they have the usual programming significance. The special symbols are listed and indexed in Appendix D. <function name>s are <identifier>s which have been associated with stored programs (via <function definition>).

There are three types of <basic statement>s in APL:

a) the <expression>, which causes APL to respond with the value of the <expression> or to associate its value with an <identifier>;

b) the <subroutine call>, which causes the appropriate subroutine to be executed;

c) the <transfer statement>, which causes a transfer to occur.

5.1 The <expression>

The <expression> may be an <operand>, a combination of operators, <expression>s, and <operand>s, or an <assignment statement>.

5.1.1 The <expression> as an <operand>

In its simplest form, the <expression> is an <operand>. The <operand> may be a <constant>, an <identifier> (possibly subscripted--see Section 8.3), an <expression> enclosed in parentheses, a <niladic function name>, [], or ["].

When an <operand> is typed in alone, APL responds according to the table below.

<operand>	APL Response
<constant>	Types the <constant>.
<identifier> <subscript option>	Types the value of the data item associated with <identifier> <subscript option>. If it is null, APL types nothing.
(<expression>)	Evaluates the <expression> and types the result.
<niladic function name>	Executes the corresponding function and types its value.
[]	*Types []: and waits for input (input must be an APL <expression>).
["]	*Returns to next line and waits for character input without indenting.
	*See Section 8.1 for further explanation.

5.1.2 The <expression> as a Combination of Operators, <operand>s, and <expression>s.

The next simplest form of the <expression> introduces APL operators. If the operator is monadic, the <expression> takes the form <monadic operator> <expression>; if it is dyadic, the <expression> takes the form <operand> <dyadic operator> <expression>. In either case, the <expression> to the right of the operator (which may also include operators) is evaluated and then the <monadic operator> is applied to the result or the <dyadic operator> is applied between the <operand> and the value of the <expression> to the right of the operator. Thus, there is no operator precedence in APL--evaluation of <expression>s proceeds directly from right to left. The order of evaluation may be changed by the use of parentheses--which is equivalent to using (<expression>) as an <operand>. Using this form of the <expression> causes the <expression> to be evaluated and the result typed out.

5.1.3 The <expression> as an <assignment statement>

The value of an <expression> may be assigned to an <identifier> or to a subset of a previously defined array <identifier> (see <subscript option>) via the <assignment statement>. It takes the form <assign operand> := <expression>. The <assign operand> may be an <identifier> followed by <subscript option>, or a []. Since this form is itself an <expression>, intermediate results may be saved or displayed during the execution of the <basic statement>.

If the <assign operand> is a non-subscripted <identifier>, the value of the <expression> is associated with that <identifier> and any previous value associated with it, regardless of its structure, is lost. If the <assign operand> is subscripted, the value of <expression> is assigned to the portion of the array <identifier> specified by <subscript option>. Permanent assignment is not made in the user's workspace until the entire <basic statement> has been evaluated successfully. However, a temporary assignment is made to allow using an <assign operand> "later" in the basic statement. If any error occurs, the evaluation was unsuccessful.

If the <assign operand> is [], the result of the <expression> is displayed immediately.

5.1.4 Outline of APL Operators
(See Section 6 for detailed descriptions.)

APL operators provide several built-in functions for the user's convenience. They may be monadic or dyadic, depending on whether they have one or two "arguments" or operands. Many APL symbols represent operators having both forms. Context decides whether the interpretation should be monadic or dyadic. A few APL symbols (CEIL, FLR, ABS, FACT and MAX, MIN, RESD, CØMB) are syntactically equivalent although they are visibly distinct. Since no special symbols were available, the names were made different for mnemonic clarity. The equivalents are shown in Appendix D.

5.1.4.1 The <monadic operator>s

There are four types of <monadic operator>s:

a) <monadic function name>s are user-defined APL functions which return a value and have one <formal parameter>. Mention of a <monadic function name> causes the corresponding function to be executed and its value returned.

b) The <monadic scalar operator>s are defined in Section 6.1. They are defined for scalars and extended to vectors and arrays by applying the operator to each element of the vector or array. The arguments must be numeric.

```
Example:  % 5 is .2  (% stands for multiplicative inverse)
          % 5 2 4 8  is  .2 .5 .25 .125
```

c) The <monadic mixed operator>s are defined in Section 6.2.

d) The <monadic suboperator>s are operators which are, in effect, subscripted. The value of the subscript determines the exact action of the operator. If no subscript is given, the highest valid subscript is used. <monadic suboperator>s also include the <reduction type operator>s where the preceding <dyadic scalar operator> also determines the action to be taken. They are defined in Section 6.3.

5.1.4.2 The <dyadic operator>s

There are five types of <dyadic operator>s:

a) <dyadic function name>s are user-defined functions which have two <formal parameter>s. Use of a <dyadic function name> causes the corresponding function to be executed with the appropriate values substituted for its <formal parameter>s and its value to be returned.

b) The <dyadic scalar operator>s are defined in Section 6.4.

They are defined for scalars and extended to vectors as follows.

For clarity, consider the basic form arg1 d arg2 where arg1 and arg2 are operands and d is a dyadic scalar operator. If arg1 (arg2) is a scalar or one-element array, the operator is applied between arg1 (arg2) and each element of arg2 (arg1) and the result has the same structure as arg2 (arg1). If arg1 and arg2 are both arrays, their ranks and dimension vectors (that is, their structures) must match and the operator is applied between corresponding elements of arg1 and arg2--thus the result has the same structure as arg1 and arg2. However, if arg1 and arg2 are one-element arrays of different ranks, an error results unless one of them is a character array of rank 1. In this case, the result has the same structure as the one-element array having rank greater than 1.

Examples:

2 + 2 is 4

1 + 2 3 4 is 3 4 5

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 2 & 1 \\ 4 & 7 \end{pmatrix} \text{ is } \begin{pmatrix} 3 & 3 \\ 7 & 11 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix} + 3 \text{ is } \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

1 2 3 4 5 + 2 3 1 4 2 is 3 5 4 8 7

c) The <dyadic mixed operator>s are defined in Section 6.5.

d) The <dot operator>s are defined in Section 6.6.

e) The <dyadic suboperator>s are operators which are, in effect, subscripted. The value of the subscript determines the action taken. If no subscript is specified, the highest valid subscript of the right-hand argument is taken. They are defined in Section 6.7.

5.2 The <subroutine call>

User-defined stored programs which do not return values are called subroutines. Following the usual conventions, the call takes the following forms:

a) <niladic subroutine name> if the subroutine has no <formal parameter>s;

b) <monadic subroutine name> <expression> if the subroutine has one <formal parameter>;

c) <operand> <dyadic subroutine name> <expression> if the subroutine has two <formal parameter>s.

In each case, the corresponding subroutine is executed. In b) and c), the <expression> is evaluated and its value substituted for the right-hand <formal parameter>. Notice that in c) the left-hand argument must be an <operand> as its value must be obtained before execution of the subroutine so it can be substituted for the left-hand <formal parameter>.

5.3 The <transfer statement>

The <transfer statement> takes the form =: <expression>. "GØ" may be used instead of "=:". Its main use is in providing transfer capabilities in functions and subroutines. It is not valid outside of stored programs, except that it is used to provide instructions for recovering from suspension of a stored program (see Section 8.4.4). The <transfer statement> may cause three different actions, depending on the value of the <expression>. (See table below.)

Value	Action
An <expression> whose first element is a statement number which occurs in the stored program being executed.	The statement specified by the statement number is executed next, and execution proceeds from there.
An <expression> which specifies a line which is not in the stored program being executed.	Normal exit (actually, a return transfer) from the stored program--local variables are released and the value, if any, is returned to the <basic statement> which invoked the stored program.
Null.	No transfer occurs. The next statement in sequence is executed.

6. Descriptions of APL Operators

APL operators differ as to the structure of the operands to which they may be applied and as to the types of values the operands may have. Definitions will be given for the operators as if they were applied to identifiers. Restrictions on structure will be indicated by the identifiers used for arguments as follows:

X, Y	mean scalar or one-element array
V, W	mean vector or scalar
L, M	mean matrix or vector
A, B	mean any structure

Notice that, while monadic and dyadic scalar operators are defined for scalars, they are extended to arrays as discussed in 5.1.4.1 b) and 5.1.4.2 b).

Restrictions on types of values of the arguments will be given in a special column as follows:

R	means the set of real numbers
N	means the set of integers
B	means the set of Boolean elements ($\{0,1\}$)
C	means the set of characters
null	means the null element

If no restrictions are given, there are none. Restrictions on actual values will be given using the standard relational operators.

Example: V RHØ A $V \in N \cup null, V \geqslant 0, A \neq null$

means that for RHØ (restructuring), the right-hand argument can have any structure and any values except null, while the left-hand argument must be either null or a vector (or scalar) composed of non-negative integers.

Since APL does not formally distinguish between integers and real numbers, errors will not result from using real numbers where integers are required. In such cases, the real numbers will be rounded to integers (.5 is rounded up).

All examples given in this section assume an ØRIGIN of 1. For further examples of the more complicated APL operators, see Appendix E.

6.1 <monadic scalar operators>

APL Symbol	Name of Operator	Form Used	Restrictions	Result
+	identity	+X	X ε R	X
-	additive inverse	-X	X ε R	negative 1 times X
&	sign	&X	X ε R	1 if X > 0 0 if X = 0 -1 if X < 0
%	multiplicative inverse	%X	X ε R X ≠ 0	1 divided by X
*	exponential	*X	X ε R	e to the power X e=2.71828...
LØG	natural logarithm	LØG X	X ε R X > 0	natural logarithm of X
CEIL	ceiling	CEIL X	X ε R	smallest integer ≥ X (FUZZ is used)
FLR	floor	FLR X	X ε R	largest integer ≤ X (FUZZ is used)
ABS	absolute value	ABS X	X ε R	absolute value of X
FACT	factorial	FACT X	X ε R X ≥ 0	X factorial if X is an integer. Γ(X) to 7 significant digits if X is not an integer
RNDM	random number	RNDM X	X ε N X ≥ ØRIGIN	an integer selected randomly between ØRIGIN and X, inclusive. (A pseudo-random number generator is used with SEED changing with each use of RNDM
NØT	negation	NØT X	X ε B	1 if X=0, 0 if X=1
CIRCLE	circular	CIRCLE X	X ε R	3.14159265 times X

6.2 Monadic Mixed Operators

APL Symbol	Name of Operator	Form Used	Restrictions	Result
IØTA	index generator	IØTA X	X N	A vector containing the first X integers starting at ØRIGIN. If X < ØRIGIN, the null vector results. (e.g. IØTA 5 is 1 2 3 4 5)
RHØ	dimension vector	RHØ A	A ≠ null	The dimension vector of A. If A is a scalar, result is null. (e.g. RHØ3 5 6 2 is 4; If B is the matrix $\begin{pmatrix}3&4&7&2&3\\2&1&6&8&9\end{pmatrix}$, RHØ B is 2 5)
,	ravel	,A		A vector containing the elements of A taken in row order (rightmost subscript varying most rapidly). If A is a scalar, result is a vector. If A is null, result is null. (e.g. ,B is 3 4 7 2 3 2 1 6 8 9 where B is defined as in RHØ example.)
TRANS	transpose	TRANS A	A ≠ null	An array with the last two coordinates of A transposed. If A is a scalar or vector, result is A. (e.g. if C is the matrix $\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}$, TRANS C is $\begin{pmatrix}1&4\\2&5\\3&6\end{pmatrix}$; if C is the array $\begin{pmatrix}1&2&3\\4&5&6\\\\7&8&9\\2&3&4\end{pmatrix}$, TRANS C is $\begin{pmatrix}1&4\\2&5\\3&6\\\\7&2\\8&3\\9&4\end{pmatrix}$.
BASVAL	base-2 value	BASVAL V	V R	Result is 2 BASVAL V(see <dyadic mixed operator>s
EPS or XEQ	execution of a character string	EPS V	V C	Result is the value of the APL <expression> giver by V. (e.g. EPS "2+3%4" is 2.75)

6.3 Monadic Suboperators

Name of Operator	Form Used	Restrictions	Result
reduction	d/[X]A or d/A where d is a dyadic scalar operator	A ε R ∪ null X ε N ORIGIN≤X≤(rank of A)+ ORIGIN-1 If d is relational or or Boolean, A ε B. More generally, elements of A and partial results must be in domain of d unless A is null.	If A is scalar, result is A. If A is null, result is the identity of the operator d if one exists, (see insert below), null otherwise. If A is a vector, result is formed by inserting d between each pair of elements of A and evaluating right to left as usual; e.g. +/1 2 3 4 5 is 15; -/5 4 3 2 1 is 3. If A is an array of rank 2, the reduction proceeds along the Xth coordinate (with respect to ORIGIN). When X is not given, it is taken to be (rank of A)+ORIGIN-1. For A a matrix and X=1, d is applied between corresponding row elements; for A a matrix and X=2, d is applied between corresponding column elements. Where right-to-left evaluation does not apply, evaluation is bottom to top. The rank of the result is one less than the rank of A. (e.g. If A is $\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}$, +/[1]A is 5 7 9 +/[2]A is +/A is 6 15 .)

d	+	&	-	%	*	RESD		MIN		MAX		CØMB
identity	0	1	0	1	1	0		4.314@68		-4.314@68		1
d	LSS	=	GEQ	GTR	NEQ	LEQ	AND	ØR	NAND	NØR	LØG	CIRCLE
identity	0	1	1	0	0	1	1	0	null	null	null	null

scan	d\[X]A or d\A where d is a dyadic scalar operator	X ε N ORIGIN≤X≤(rank of A)+ ORIGIN-1 A ε R Partial results must be in correct domain for d (as in reduction).	If A is scalar, result is A. Otherwise, scan proceeds along the Xth coordinate with respect to ORIGIN where X=(rank of A)+ ORIGIN-1 if X is not given. Result has same structure as A, where each element is d applied between the last-obtained element and the corresponding element of A. Scan goes from left to right or top to bottom, depending on X, and the topmost or leftmost element of the result is the corresponding element of A. (e.g. + \1 2 3 4 5 is 1 3 6 10 15. If A is $\begin{pmatrix}1&2&3&4\\5&6&7&8\\0&3&4&7\end{pmatrix}$ + \A is +\[2]A is $\begin{pmatrix}1&3&6&10\\5&11&18&26\\0&3&7&14\end{pmatrix}$ and +\[1]A is $\begin{pmatrix}1&2&3&4\\6&8&10&12\\6&11&14&19\end{pmatrix}$

Monadic Suboperators, continued

Name of Operator	Form Used	Restrictions	Result
reversal	PHI[X]A or PHI A	X ε N ØRIGIN$\leq$X$\leq$(rank of A)+ØRIGIN-1 A $\neq$ null	An array with the same structure as A where elements of the result are elements of A in reverse order along the Xth coordinate with respect to ØRIGIN. If X is not specified, (rank of A)+ØRIGIN-1 is used. (e.g. PHI 1 2 3 4 5 is 5 4 3 2 1; If A is $\begin{pmatrix}1&2&3&4\\5&6&7&8\end{pmatrix}$, PHI[1]A is $\begin{pmatrix}5&6&7&8\\1&2&3&4\end{pmatrix}$ and PHI[2]A is PHI A is $\begin{pmatrix}4&3&2&1\\8&7&6&5\end{pmatrix}$).
sorting up	SØRTUP[X]A or SØRTUP A	same as reversal	If A is scalar, result is ØRIGIN. Otherwise, the permutation of indices which would order A along the Xth coordinate with respect to ØRIGIN. If X is not specified, (rank of A)+ ØRIGIN-1 is used. Order is ascending, with topmost and leftmost elements given first if they are equal. For a vector V, V[sortup V] does the actual ordering. (See subscripts -- Section 8.3.) (e.g. SØRTUP 5 3 7 9 3 2 is 6 2 5 1 3 4; if A is $\begin{pmatrix}1&6&3&4\\5&2&7&8\end{pmatrix}$, SØRTUP[1]A is $\begin{pmatrix}1&2&1&1\\2&1&2&2\end{pmatrix}$ SØRTUP A is SØRTUP[2]A is $\begin{pmatrix}1&3&4&2\\2&1&3&4\end{pmatrix}$).
sorting down	SØRTDN[X]A or SØRTDN A	same as reversal	Same as sorting up, except the permutation is for descending order. (e.g. SØRTDN 5 3 7 9 3 2 is 4 3 1 2 5 6).

6.4 Dyadic Scalar Operators

APL Symbol	Name of Operator	Form Used	Restrictions	Result
+	addition	X + Y	X,YϵR	sum of X and Y
-	subtraction	X - Y	X,YϵR	X minus Y
&	multiplication	X & Y	X,YϵR$\cup$null	if X or Y is null, result is null; otherwise X times Y
%	division	X % Y	X,YϵR, Y$\neq$0	X divided by Y
*	exponentiation	X * Y	X<0 and YϵN or X>0 or X=0 and Y>0 X, YϵR	X raised to the power Y
LØG	logarithm	X LØG Y	X,YϵR X>1, Y>0	logarithm of Y to the base X ((LØG Y) % LØG X)
MAX	maximum	X MAX Y	X,YϵR	X or Y, whichever is greater
MIN	minimum	X MIN Y	X,YϵR	X or Y, whichever is smaller
RESD	residue	X RESD Y	X,YϵR see result	the smallest non-negative element of the set {Y-I&X} where I is any positive integer. Y-(ABS X)&FLR Y%ABS X if X$\neq$0 Y if X=0 and Y$\geq$0 domain error if X=0 and Y<0
CØMB	combinatorial	X CØMB Y	X,YϵR	(FACT Y) % (FACT X) & FACT Y-X; if X,YϵN and X>Y, result is 0. (recall that FACT may give the gamma function)
LSS	less than	X LSS Y	X,YϵR or X,YϵC	*1 if X<Y, 0 otherwise (FUZZ is used)
LEQ	less than or equal	X LEQ Y	X,YϵR or X,YϵC	*1 if X$\leq$Y, 0 otherwise (FUZZ is used)
=	equals	X = Y	X,YϵR$\cup$C	*1 if X=Y, 0 otherwise (FUZZ is used)

*When characters are compared, their octal equivalents, listed in Appendix F, are used.

Dyadic Scalar Operators, continued

APL Symbol	Name of Operator	Form Used	Restrictions	Result
NEQ	not equal	X NEQ Y	X,YϵR$\cup$C	*1 if X$\neq$Y, 0 otherwise (FUZZ is used)
GEQ	greater than or equal	X GEQ Y	X,YϵR or X,YϵC	*1 if X$\geq$Y, 0 otherwise (FUZZ is used)
GTR	greater than	X GTR Y	X,YϵR or X,YϵC	*1 if X>Y, 0 otherwise (FUZZ is used)
AND	and	X AND Y	X,YϵB	1 if X and Y are 1, 0 otherwise
ØR	or	X ØR Y	X,YϵB	0 if X and Y are 1, 1 otherwise
NAND	nand	X NAND Y	X,YϵB	0 if X and Y are 1, 1 otherwise
NØR	nor	X NØR Y	X,YϵB	1 if X and Y are 0, 0 otherwise
CIRCLE	circular	X CIRCLE Y	XϵN, YϵR (ABS X) LEQ 7	See table below. Angles are in radians

(-X) CIRCLE Y	Restrictions	X	X CIRCLE Y	Restrictions
(1-Y*2)*.5	(ABS Y) LEQ 1	0	(1-Y*2)*.5	(ABS Y) LEQ 1
arcsine of Y	(ABS Y) LEQ 1	1	sine of Y	
arccosine of Y	(ABS Y) LEQ 1	2	cosine of Y	
arctangent of Y		3	tangent of Y	0 $\neq$ cosine of Y
(#1+Y*2)*.5	(ABS Y) GEQ 1	4	(1+Y*2)*.5	
arcsinh of Y		5	sinh of Y	
arccosh of Y	Y GEQ 1	6	cosh of Y	
arctanh of Y	(ABS Y) LSS 1	7	tanh of Y	

*When characters are compared, their octal equivalents, listed in Appendix F, are used.

6.5 Dyadic Mixed Operators

APL Symbol	Name of Operator	Form Used	Restrictions	Result
IØTA	indexing	V IØTA A	A∈R∪C V∈R∪C∪null	If V is null, result is ØRIGIN; otherwise, result has same structure as A where for each element X of A, the corresponding element of the result is the least index I of V such that V[I]=X. If V[I]≠X for any valid I, then I is ØRIGIN + (length of V). (e.g. 1 8 7 IØTA 3 6 7 8 1 4 6 is 4 4 3 2 1 4 4).
RHØ	restructuring	V RHØ A	V∈N∪null V≥0 A≠null	If V is null, result is a scalar; if any element of V is 0, result is null. Otherwise, result is an array whose dimension vector is given by V with elements taken from ,A (A ravelled), repeating ,A as many times as necessary. (e.g. 3 3 RHØ 1 0 0 0 is $\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$).
,	catenation	V , W	V,W∈R∪null or V,W∈C∪null	A vector composed of the elements of V followed by the elements of W. If V (W) is null, result is the same as W (V). (e.g. 1 2 3 , 4 2 6 is 1 2 3 4 2 6.)
RNDM	random deal	X RNDM Y	X,Y∈N X≤Y-ØRIGIN+1	A vector composed of X elements taken at random (using a pseudo-random number generator as in monadic RNDM), without replacement, from IØTA Y. If X≤0, result is null. (e.g. 8 RNDM 8 might be 4 7 5 3 8 2 6 1.) X calls to the random number generator are made.
BASVAL	base value	V BASVAL W	V,W∈R V≠0	The decimal value of W with respect to the base given by V. Notice that V may be of mixed radix variety. V is effectively modified to make the lengths of V and W equal by extending V to the left (copying it over) as many times as necessary. If V is longer than W, the rightmost elements of V are used. Where N is the length of W, the formula is $(\sum_{I=2}^{N} W[I-1] \times \prod_{J=I}^{N} V[J]) + W[N]$. Note that if V is a scalar, it is extended to a vector of length N with each element equal to V, and the formula becomes $\sum_{I=1}^{N} W[I] \& V * (N-I)$. (e.g. the number of pennies in 3 half-dollars, 4 quarters, 6 nickels, and 8 pennies is 2 2 5 5 BASVAL 3 4 6 8 is 288. 2 BASVAL 1 1 0 is 6 2 3 BASVAL 4 1 2 is the same as 3 2 3 BASVAL 4 1 2.

Dyadic Mixed Operators, continued

APL Symbol	Name of Operator	Form Used	Restrictions	Result
REP	representation	V REP X	V,XϵR V$\geq$0	The digits of the base V representation of the decimal integer X in vector form. Length of result is same as length of V. (e.g. 2 2 2 REP 6 is 1 1 0; 2 2 5 5 REP 288 is 5 1 2 3; 60 60 REP 3721 is 62 1.)
EPS	membership	A EPS B	A,BϵR$\cup$C	A Boolean array with the same structure as A where for each element X of A, the corresponding element in the result is 1 if X = Y for some element Y of B, 0 otherwise. (e.g. 3 4 9 7 EPS $\begin{pmatrix}1&2&3&4\\5&6&7&8\end{pmatrix}$ is 1 1 0 1.)
TAKE	take	V TAKE A	VϵN rank of A$\geq$1 length of V = rank of A	V[I] determines the number of elements to be taken from the Ith coordinate of A. If V[I]>0, the first V[I] elements of the Ith coordinate of A are taken. If V[I]<0, the last ABS V[I] elements of the Ith coordinate of A are taken. If V[I]=0 for any I, result is null. If V[I] is greater than the size of the Ith coordinate, a domain error occurs. If V is a scalar, it is extended to a vector of the appropriate length. (e.g. if A is $\begin{pmatrix}1&2&3&4\\5&6&7&8\\3&1&3&5\\4&2&6&7\end{pmatrix}$, 2 TAKE A is $\begin{pmatrix}1&2\\5&6\end{pmatrix}$ and #2 3 TAKE A is $\begin{pmatrix}3&1&3\\4&2&6\end{pmatrix}$.)
DRØP	drop	V DRØP A	same as take	Result is like TAKE except that the indicated elements are dropped rather than taken. Thus, if V[I] = the size of the Ith coordinate, result is null. (e.g. with A as in TAKE example, 2 DRØP A is $\begin{pmatrix}3&5\\6&7\end{pmatrix}$, 1 #2 DRØP A is $\begin{pmatrix}5&6\\3&1\\4&2\end{pmatrix}$.)
;	heterogenous output	V ; W	see result. Note that if ';' appears between '[' and ']' it will be interpreted as a subscript delimiter.	V is an operand and W is an expression. V and W when evaluated must have rank not greater than 1. Result is a character vector which is the concatenation of the strings which would be typed for V and W. (e.g. if X is 1 2 3 and Y is 3, "Y PLUS X = " ; X + Y is "Y PLUS X = 4 5 6"; "Y PLUS X = " ; A:=X+Y gives the same result.)

Dyadic Mixed Operators, continued

APL Symbol	Name of Operator	Form Used	Restrictions	Result
TRANS	dyadic transpose	V TRANS A	V ε N, V exhausts IOTA MAX/V, RHO V = RHO RHO A unless V is scalar, A ≠ null	V is a permutation vector with respect to ORIGIN. Rank of the result is (MAX/V)-ORIGIN +1 and is less than or equal to rank of A. The dimension vector of the result is found as follows: let R = RHO A and NEWRHO be the dimension vector of the result. Then for each I ε IOTA MAX/V, NEWRHO [I] = least element of R [J] such that V [J] = I. Each element of the result is found by permuting the subscripts of the result according to V to find the correct element of A. That is, if V = 3 1 2, then RESULT [I;J;K] = A [K;I;J] . If V = 1 1 2, RESULT [I;J] = A [I;I;J] . If V is scalar and A is vector or scalar, then V must be equal to ORIGIN and the result is A.

e.g. if A = 2 3 4 RHO IOTA 24

then

```
      1 3 2 TRANS A = 1  5  9
                      2  6 10
                      3  7 11
                      4  8 12

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

      3 1 2 TRANS A = 1 13
                      2 14
                      3 15
                      4 16

                      5 17
                      6 18
                      7 19
                      8 20

                      9 21
                     10 22
                     11 23
                     12 24

      2 1 3 TRANS A = 1  2  3  4
                     13 14 15 16

                      5  6  7  8
                     17 18 19 20

                      9 10 11 12
                     21 22 23 24

      1 1 2 TRANS A = 1  6  11
                     13 18  23
```

6.6 The Dot Operators

Name of Operator	Form Used	Restrictions	Results
inner product	I dl . d2 M where dl and d2 are dyadic scalar operators and dl is not CIRCLE	L,M ≠ null. Size of last coordinate of L must equal size of first coordinate of M unless one or both is scalar. dl and d2 must be defined on the elements to which they are applied.	An array of rank (rank of L)+(rank of M)-2 or a scalar if that rank is negative, whose dimensions are given by the size of the first coordinate of L and the size of the last coordinate of M. (Note that the first coordinate of L and the last coordinate of M are null if they are vectors.) When L and M both have rank 2, the [I;J]th element of the result is formed by applying d2 between the Ith row of L and the Jth column of M and then reducing this new vector over dl (see reduction--dl is effectively inserted between pairs of elements of the vector and this new entity is evaluated). If L (M) is a vector, it is treated as a row vector--a 1 by RHØ L matrix--(a column vector--a RHØ M by 1 matrix--) and the result is evaluated as if it were a matrix. If L (or M) is a scalar, it is treated as a row (or column) vector of appropriate length, each element of which has the value L (or M). +.& is standard matrix multiplication. (e.g. if A is $\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}$ and B is $\begin{pmatrix}1&2\\3&4\\5&6\end{pmatrix}$, 2+.&A is 2 2; +.& A is 10 14 18; A+.&B is $\begin{pmatrix}22&28\\49&64\end{pmatrix}$; 3 +.& 5 is 15).
outer product	A CIRCLE . d B where d is a dyadic scalar operator	A,B ≠ null. d must be applicable between each pair of elements of A and B.	An array of dimension (RHØ A) , RHØ B which is formed by applying d between each element of A and all of B. (e.g. if B is $\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}$, 2 CIRCLE.&B is $\begin{pmatrix}2&4&6\\8&10&12\end{pmatrix}$; 1 2 3 CIRCLE.& 1 2 is $\begin{pmatrix}1&2\\2&4\\3&6\end{pmatrix}$).

6.7 The Dyadic Suboperators

Name of Operator	Form Used	Restrictions	Result
rotation	A PHI[X] B or A PHI B	A,XϵN ØRIGIN$\leq$X$\leq$(rank of B)+ØRIGIN-1. Dimension(s) of A must equal the dimension(s) of B with the Xth coordinate removed unless A is scalar. B $\neq$ null.	An array with same structure as B where the elements of B have been rotated according to the specifications of A. Rotation is along the Xth coordinate with respect to ØRIGIN and X is taken to be (rank of B)+ØRIGIN-1 if not specified. Corresponding elements of A (see examples) give the number of positions to rotate--if the element is positive, rotation is to the left or top; if the element of A is negative, rotation is its absolute value to the right or down. If A is a scalar, it is treated as an array of appropriate structure, each element of which has the value A. (e.g. 0 1 2 0 PHI[1] $\begin{pmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\end{pmatrix}$ is $\begin{pmatrix}1&6&11&4\\5&10&3&8\\9&2&7&12\end{pmatrix}$; $\begin{pmatrix}0&1&2&0\\1&0&\#1&2\end{pmatrix}$PHI[2] $\begin{pmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\\13&14&15&16\\17&18&19&20\\21&22&23&24\end{pmatrix}$ is $\begin{pmatrix}1&6&11&4\\5&10&3&8\\9&2&7&12\\17&14&23&24\\21&18&15&16\\13&22&19&20\end{pmatrix}$).
compression	V /[X] A or V / A	XϵN; VϵB. ØRIGIN$\leq$X$\leq$(rank of A)+ØRIGIN-1. Length of V must equal the size of the Xth coordinate of A unless V is scalar.	If V is scalar, result is A if V=1, null if V=0. If A is null, result is null. If A is scalar, result is A if V is 1, null if V is 0. Otherwise, A is compressed along the Xth coordinate with respect to ØRIGIN where X is (rank of A)+ØRIGIN-1 if not specified. The coordinate is deleted if the corresponding element of V (see example) is 0, included in the result if the corresponding element of V is 1. (e.g. 1 0 0 1 / 1 2 3 4 is 21 4 ; if A is $\begin{pmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\end{pmatrix}$, 1 0 1 1/A is $\begin{pmatrix}1&3&4\\5&7&8\\9&11&12\end{pmatrix}$ and 1 0 1 /[1]A is $\begin{pmatrix}1&2&3&4\\9&10&11&12\end{pmatrix}$

Dyadic suboperators, continued.

Name of Operator	Form Used	Restrictions	Result
expansion	V\[X]A or V\A	V∈B ; X∈N. A ≠ null. ØRIGIN≤X≤(rank of A)+ØRIGIN-1. The number of 1's in V must equal the size of the Xth coordinate of A.	An array formed by expanding A along the Xth coordinate with respect to ØRIGIN where X is (rank of A)+ØRIGIN-1 if it is not specified. Where an element of V is 0, 0's (blanks if A is a character array) are inserted for the corresponding "row" or "column" of the result. (e.g. if A is $\begin{pmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\end{pmatrix}$, 1 0 1 0 1 1 \ A is $\begin{pmatrix}1&0&2&0&3&4\\5&0&6&0&7&8\\9&0&10&0&11&12\end{pmatrix}$; and 1 0 1 1 \ [1] A is $\begin{pmatrix}1&2&3&4\\0&0&0&0\\5&6&7&8\\9&10&11&12\end{pmatrix}$; 1 0 1 0 1 \ "ABC" is "A B C"

7. APL Stored Programs

As mentioned earlier, APL statements can be grouped together to form stored programs for later execution. There are two basic types of stored programs, those which return a value (called "functions"), and those which do not (called "subroutines"). These classes are further subdivided by the number of <formal parameter>s in their definitions. This section discusses <stored program definition> and <edit> capabilities, and the execution of stored programs.

7.1 The Mechanics of <stored program definition>

The <stored program definition> consists of two parts, the <definition entry> and the <stored program body>, enclosed in dollar signs ("$"). The <definition entry> may be a) the <stored program name> of a previously defined stored program, in which case the specified stored program is reopened for further definition, or b) a <header>, in which case a new stored program is being defined. The <stored program body> is a set of APL <basic statement>s which may be labelled, ordered by statement numbers. <edit> commands (which alter, display and delete lines) are also considered part of the <stored program body> although they do not appear in the completed definition.

7.1.1 The <header>

The <header> is always followed by a "←". It specifies the name, number of <formal parameter>s and their names, type and <local variable>s of the stored program. The <header> may not be changed without removing the entire stored program. It takes the form <stored program option>s <local variable>s. The <stored program options> may take any of the six forms given in the table below. In the table, Z stands for the <variable name> in the <function specifier>, PRG stands for the name of the stored program, and A and B stand for the names of the <formal parameter>s. All of these are simply <identifier>s.

Subclass:	Niladic	Monadic	Dyadic	type
	Z:=PRG	Z:=PRG A	Z:=A PRG B	function
	PRG	PRG A	A PRG B	subroutine

7.1.1.1 The <function specifier>

The presence of a <function specifier> (of the form <variable name> :=) in the <header> indicates that the stored program is to be a function. The value of the function becomes the value last assigned to the <variable name> given in the <function specifier> during execution. The value of the function is initialized to null at each call.

7.1.1.2 The <formal parameter>s

The <formal parameter>s are <identifier>s which serve as names for the arguments that will actually be used when the stored program

is invoked. They are positioned about the name of the stored program in the same way the actual arguments are placed when it is invoked. They are "called by value"--that is, changing the value of a <formal parameter> during execution of the stored program will not affect the corresponding actual argument.

7.1.1.3 The Name of the Stored Program

The name of the stored program becomes permanently associated with the <header> and definition given by the <stored program body>. It is specified by type and subclass as indicated in the table of 7.1.1 (e.g. <monadic function name>, <dyadic subroutine name>, etc.). It is, in effect, a user-defined operator and as such has at most two arguments.

7.1.1.4 The Local Variables

The user may specify <identifier>s which are "local" to the stored program by following the stored program options with a ";", followed by a list of <identifier>s separated by semicolons. For instance, the <header> for a dyadic function with three local variables might look like Z:= A FUNC B; X; Y; Z←. The local variables X, Y, and Z are initialized to null at each call of the stored program and have no relation to variables of the same name in the user's workspace or in other stored programs. They cease to exist when execution of the stored programs has been completed.

7.1.2 The<stored program body>

Once <definition entry> has been accomplished and a "←" supplied, the following sequence of events occurs repeatedly until the user types in another "$".

1) APL provides a statement number (the last statement number used plus an increment) inclosed in brackets and waits for user input. The increment between statement numbers is usually 1, but may be altered through <edit>ing (Section 7.2). For defining a new stored program, APL provides [1].

2) The user types in a line, which may be
   a) a <compound statement>, or
   b) an <edit> command.

3) a) If the line is a <compound statement>, APL stores it with the rest of the definition. If the SYN option is in effect (Section 3.3), syntax errors are noted where possible. If the line is not a <compound statement>, but does not start with a "[" (which distinguishes the <edit>), the line will be stored anyway. Thus a "stored program" might be used to document other stored programs.

   b) If the line is an <edit>, the appropriate action is taken (Section 7.2).

If the <definition entry> re-opened the definition of a stored program, the <stored program body> may start on the same line as the <definition entry>. The "$" concluding the <stored program definition> need not appear on a separate line.

7.1.2.1 The <compound statement>

The <compound statement> is either a <basic statement> (Section 5) or a <basic statement> preceded by one or more <label>s. A <label> is an <identifier> followed by a ":". A label is normally used when a forward transfer is desired.

7.1.2.2 The Use of <label>s

When a stored program is invoked, each <identifier> used in a <label> is initialized to the statement number of the line it appears on. However, it is otherwise treated like a local variable. Thus, the value of the "label" may change, and it may be used in computations.

7.1.2.3 The Use of "Global Variables"

"Global variables" are the <identifier>s associated with <data element>s and definitions of stored programs which are stored in the user's workspace. Global variables may be used or changed in stored programs, except that values are not changed permanently until the stored program has been executed successfully. (Section 3.2). Global variables are superseded by local variables of the same name during the execution of a stored program. However, a variable which is local to stored program P1 is not global to stored program P2 when P1 invokes P2.

7.1.3 Example of a <stored program definition>
(See Appendix E for further examples.)

```
     $ S := STDEV X←
[1]  AVE := (+/X) ⌹ N:=RHØ X←
[2]  S := ((+/(X-AVE)*2) ⌹ N-1)*.5←
[3]  $←
```

This is a definition for a function, STDEV, with one <formal parameter>, X, whose value is the standard deviation of a vector X. As a side effect, it also sets the global variable AVE to the average value of the vector, and the global variable N to the number of elements of X.

If the above side effects were not wanted, the <header> line should have been $ S := STDEV X; N; AVE←, which makes N and AVE into local variables.

A subroutine to perform the same operation, storing the answers in S, AVE, and N would have the following <header> line: $ STDEV X←.

7.2 The <edit>ing of a <stored program defintion>

The APL user is provided with a number of <edit>ing capabilities. They are <display> , <insertion> , <change>ing, <delete>ing, and <resequence>ing.

The concept of the <line reference> is basic to all of the <edit> commands. The simplest sort of <line reference> is the statement number of the line which is being referred to. Another type of <line reference> is a label which occurs on the line in question. The most complex <line reference> is the form <identifier> + <number>. This simple expression is evaluated to determine the <line reference>. This construct allows the user to refer to a line relative to a line which has a label. A <line reference> may not reference statement number zero, which is reserved for the <header>. Examples of the <line reference>, where "LAB" is a <label>: 23 LAB LAB+5.1 LAB-2 LAB+@2 14 .

7.2.1 The <display> Command

The <display> command causes APL to list a set of lines of the stored program currently being defined. It takes the form [<line option>[]<line option>]. The "[]" is read as a single symbol, "quad", and may not have blanks in it. The <line option> may be a <line reference> or <empty>, except that the first <line option> may not be <empty> unless both are. The following table gives the meaning of the various types of <display> commands:

Display Command	Meaning
[[]]	The entire stored program is displayed.
[2[]]	Line 2 of the stored program is displayed.
[2[]6]	Lines 2 through 6 are displayed.
[LAB[]]	The line with the <label> "LAB" is displayed.
[LAB-3[]LAB+5]	Lines from number LAB-3 through number LAB+5 are displayed.

7.2.2 The <insertion> Command

The <insertion> command takes the form <line reference> <compound statement>. It causes the specified <compound statement> to be inserted at the statement number position specified by <line reference>. If the <line reference> specified is a <number>, APL will change the increment between statement numbers to 10^{-k}, where k is the number of digits after the decimal point (if any) in <number>. (There may be as many as four digits ahead of the decimal point, and four digits after the decimal point, making an implicit restriction of 99,999,999 lines per stored program. The practical limit is considerably smaller.) If the <line reference> is a <number> which already appears as a statement number, or a <label>, the specified line is overwritten.

Insertion Command	APL Action
[2]A:=3 4 + 6 2	Line is inserted at 2, increment becomes 1.
[LAB]B:=C+D	Line is inserted over the line labelled "LAB", increment is unchanged.
[9.001] 2+2	Line is inserted at 9.001, increment becomes .001.

7.2.3 The <change> Command

This command allows alteration of the text of a stored program without retyping the lines. It takes the form [<line option>["]<line option>]<line edit>. The symbol "["]" is read as a single symbol, "quote quad" and may not have blanks in it. The <line option> has the same meaning as in the <display> command.

The <line edit> takes the form <search string>"<insert string>{"<search string> / or <empty> .

Each of the strings may be either a <proper string> (a character string without a quotation mark), or <empty>. The action of the APL interpreter when the command is issued is the following:

1. Search the line for the first occurence of the character string specified by the first <search string> (the null string is found immediately).

2. When it is found, insert the string of characters specified in the <insert string>.

3. Delete characters to the point where the second <search string> matches characters in the line (the null string is found immediately).

The above action is taken for each line specified by the <line option>s. Suppose the line in question were ABCDEFGHLMNOP. The following table shows the result of a <change> command on that line.

Change Command	Resulting Line
[2["]] GH"IJK"LM	ABCDEFGHIJKLMNOP
[LAB["]] GH"IJK	ABCDEFGHIJKLMNOP
[2["]6] GH"IJK"	ABCDEFGHIJKLMNOP
[["]] GH"IJK"P	ABCDEFGHIJKP
[LAB["]5] GH "IJK"L	ABCDEFGHLMNOP ("GH " is not found)

7.2.4 The <delete> Command

The <delete> command takes the form [<line reference>] or [<line reference>] [<line reference>]. The first form removes the single line referred to by <line reference>, while the second form deletes lines from the first <line reference> through the second <line reference>, inclusive. The following table illustrates APL's response.

Delete Command	APL Response
[2]	Removes line 2.
[3] [10]	Removes lines 3 through 10.
[LAB-1] [LAB+1]	Removes lines LAB-1 through LAB+1.

7.2.5 The <resequence> Command

The user may resequence the lines of his stored program by typing [IOTA]. The new line numbers start at [1] with an increment of 1.

7.3 Execution of a Stored Program

A stored program is invoked by the appearance of its name (surrounded by the appropriate number of arguments) in a <basic statement> (Section 5).

At this time, the correspondence is set up between copies of the actual arguments and the <formal parameter>s given in the <header>. Local variables and the value of the stored program (if it is a function) are initialized to null, and <identifier>s used in <label>s in the definition are initialized to the statement numbers of the lines on which they appear.

Each line of the stored program is analyzed and executed as if it had been input directly except that the conventions about local variables apply in referencing variables, and that line-feed, carriage-return is given only if the line has generated some output. Lines are fetched sequentially according to statement number except when a branch occurs (Section 5.3). The stored program terminates normally either by "running off the end" of the definition or by transferring to a statement number that is not in the definition (this method is called a return transfer). Global variables are not permanently changed in the user's workspace until the <basic statement> which invoked the stored program is evaluated successfully. The same rule applies when there are nested calls, except that successful evaluation of the original <basic statement> is required. If the stored program is a function, the value is returned upon normal exit.

If an error occurs, an error message is typed out and the stored program becomes suspended--see Section 8.4.4.

Helpful hints: For more efficient execution, previously executed lines of a stored program are kept in a semi-compiled form until the <basic> statement which caused the stored program to be invoked has been completed. It is faster to execute a stored program which loops, than one which calls a non-looping version iteratively since the labels, local variables and parameters do not have to be re-initialized in the former case.

If a stored program is in an infinite loop, hit the break key when a time-out "jiggle" occurs (Section 1.2.1). This will stop the loop and suspend the stored program (Section 8.4.4).

8. Miscellaneous APL Conventions and Notations

Certain concepts and notations, while not operators in the strictly manipulative sense, need further explanation. Error messages and modes of operation are also described.

8.1 Quad ([]) and Quote-Quad (["]) Input

When [] appears as an operand in a <basic statement>, APL types []: when evaluation reaches the [] and waits for the user to supply input. The input given is evaluated as an APL <expression>. The value of the [] then becomes the result of the given <expression>, provided that no errors have occurred.

When ["] appears in a <basic statement>, APL spaces to the left margin and waits for input when evaluation reaches the ["]. The value of the ["] becomes the character string the user supplies, up to the first ←. Quotes (") may appear in the string since ["] assumes character input and does not require the string to be enclosed in quotation marks.

8.2 Display

When the form []:= appears in an <expression>, the action is the same as it would have been had the [] been an <identifier> except that the value is displayed on the teletype rather than associated with an <identifier> and stored. Thus, for following through a calculation, the sequence of APL <basic statement>s

```
A := 2 + B := 3 + C := 2 + (D := 7) + 5←
D←
C←
B←
```

gives the same result as

```
A := 2 + [] := 3 + [] := 2 + ([] := 7) + 5←
```

except that the intermediate results indicated are not stored and the user does not have to request results one at a time in the latter case. In either case, APL would respond with

```
7
14
17 .
```

8.3 The <subscript option>

When an <identifier> is referenced and the <subscript option> is <empty>, the entire scalar, vector, or array is referenced.

When the <subscript option> is not <empty>, it takes the form [<subscript list>] and the <subscript list> references a subset of a previously defined array. Unnamed <expression>s may not be subscripted.

APL provides a powerful extension of the usual capability of selecting or replacing single elements of arrays. (e.g. if A is $\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$ then A[2;1] is 4.) In APL, a subscript can be an expression , and its value can be an array. The following rules and examples explain this concept more precisely.

The number of <subscript>s in a <subscript list> must be the same as the rank (number of dimensions) of the subscripted array. The <subscript>s in the list are separated by semicolons (;). Note that for this reason the <dyadic mixed operator>, ";", is not allowed in any <expression> within a <subscript list>.

A <subscript> can be either an <expression> or <empty> where the value of the <expression> must be either numeric or null. (E.g. A[2+2;6], B[3;;5], and A[X;3] where X is $\begin{pmatrix}1 & 3\\ 6 & 2\end{pmatrix}$ are all valid <subscript list>s.) If any value is not an integer, it will be rounded to an integer. If the Ith <subscript> is neither null nor <empty>, it must take values in the range ØRIGIN to ØRIGIN + (size of Ith dimension) - 1.

If every <subscript> evaluated is a scalar, the element is selected in the usual fashion (rightmost represents column, the next gives the row, and additional <subscript>s select the higher dimensions as one moves left). The result in this case is a scalar.

When any of the <subscript>s is an array, <empty>, or null, further rules are used to select the elements referenced.

A null or <empty> in the Ith subscript indicates that all values of that dimension are to be selected. The subscript effectively becomes IØTA (size of the Ith dimension).

The rank of the set of elements selected is the sum of the ranks of the <subscript>s (recall that the rank of a scalar is 0 and since <empty>s and nulls have been effectively replaced by vectors, their ranks are considered to be 1). The dimension vector of the set can be found by concatenating the dimension vectors of each <subscript>, taken left to right, except that the size of the Ith dimension of the subscripted array is substituted if the Ith subscript is <empty> or null. (Recall that the dimension vector of a scalar is null.)

The elements of the result are selected by first selecting the first element of each <subscript>, and then finding the corresponding element of the subscripted array as is done when all of the <subscript>s are scalars. Succeeding elements are found by stepping through the <subscript> elements with the

rightmost <subscript> "running fastest". At each step the corresponding element of the subscripted array is selected. The elements found in this manner are then structured according to the dimension vector as discussed above. A few simple examples should illustrate this. More complicated examples are given in Appendix E.

Let A be $\begin{pmatrix}1 & 5 & 16\\ 9 & 4 & 7\\ 3 & 8 & 2\end{pmatrix}$ and let B be $\begin{pmatrix}1 & 2\\ 2 & 3\end{pmatrix}$. A[1;1 2] is 1 5. A[2 3;2] is 4 8.
A[1 2; 3 2] is $\begin{pmatrix}16 & 5\\ 7 & 4\end{pmatrix}$. A[2;] is 9 4 7. A[;3] is 16 7 2. A[B;2] is $\begin{pmatrix}5 & 4\\ 4 & 8\end{pmatrix}$.

If an <assign operand> is subscripted, (e.g. A[;1] := 2 4 6), the structure of the subset of the array specified by the <subscript list> must be the same as the structure of the <expression> on the right hand side of the ":=". In this case, replacement of corresponding elements occurs. (e.g. A[;1] := 2 4 6 would change A to $\begin{pmatrix}2 & 4 & 6\\ 9 & 4 & 7\\ 3 & 8 & 2\end{pmatrix}$).

8.4 Modes of Operation

The mode of operation a user is currently in will affect what he may do and may alter the effects that certain actions have.

8.4.1 Execution or Calculator Mode

Definition: The user is typing in statements to be executed. Before this, there must have been an even number of "$" so he is not in stored program definition mode.

Effects: Each line up to the "←" is evaluated. APL responds according to the nature of the line and indents six spaces on the teletype to signal that evaluation of the line has been completed. <basic statement>s and <monitor command>s are allowed. Stored program editing and <transfer statement>s are not allowed.

8.4.2 <stored program definition>

Definition: While in execution mode, the user has typed a "$" and has not typed a second "$".

Effects: <monitor command>s are ignored since lines given are stored rather than evaluated. If the user has typed)SYN before entering <stored program definition> mode, each line will be checked for syntax errors as it is given. <edit> commands are allowed. If execution of APL terminates during <stored program definition> mode, the definition will continue where it left off and the user will get the message "CØNTINUE DEFINITIØN ØF <stored program name>" the next time he logs in to APL.

8.4.3 Stored Program Execution Mode

Definition: During evaluation of a line given while in execution mode or in stored program execution mode, a stored program has been invoked and it is being executed.

Effects: Each line of the stored program is evaluated in order as if it had been given in calculator mode (except that <monitor command>s will produce syntax errors) and the appropriate responses are made. Permanent changes in the user's workspace are not made until the entire <basic statement> which invoked the stored program(s) has been evaluated successfully--but temporary changes are made as described in the discussion on the <assignment statement>.

8.4.4 Suspended Mode

Definition: While in stored program execution mode, an error has occurred or user has hit the "break key" and input from the user is needed to recover from the error.

Effects: <transfer statement>s may be input to transfer to other statements within the suspended stored program--normal rules on valid <transfer statement>s and actions taken apply, except that statement numbers refer to the stored program most recently suspended. A return transfer causes the last-suspended stored program to be removed from the list of suspended stored programs and it becomes "un-suspended". Values of local variables can be displayed (e.g. X←) or changed (e.g. X:=52←). When <local variable>s have the same name as other <local variable>s in a chain of suspended functions or have the same name as global variables, the most recently active local variable is the one referenced. The <monitor command>s)SI,)SIV,)ABØRT and)STØRE may be used to diagnose the problem and/or recover. Other <monitor command>s and <basic statement>s (except for)CLEAR and)ERASE) may also be used. If the user attempts to enter stored program definition mode from suspended mode, he will receive the message "ABØRT SUSP. FCNS" and will not be allowed to enter.

8.5 Error Messages

Syntax scan and execution of APL statements goes from right to left. When an error is found, APL types out the name of the error and the six characters to the right of the point where the error was discovered. If any of these errors occur during evaluation of a <basic statement> while in execution mode, permanent assignment of values to variables does not occur (see <assignment statement>). If they occur during stored program execution mode, the line number is also typed out and the stored program becomes suspended (see suspended mode). The APL error messages and typical causes are listed on the next page.

DØMAIN The values of operands given do not fall within the class of allowable operands listed in Section 6. Things like dividing by zero or using a null operand where it is not allowed are included.

INDEX An attempt was made to access a non-existent subset of an array.

RANK Rank of operand not consistent with restrictions given in Section 6 or wrong number of subscripts given.

SYNTAX Syntax inconsistent with Appendix A or invalid parameters on monitor command. (E.g.)ØFF DRØP,)WIDTH 97, A := 2 + 2 NØT B)

NØNCE What you tried has not been implemented.

DEPTH Either the execution stack or list of active and suspended stored programs is too long (there are practical--but reasonable--limits on how complicated a statement may be and how deeply stored program calls may go).

LENGTH Input or output line too long or arguments are not conformable due to unequal lengths (e.g. 2 3 + 2 3 4)

LABEL Attempt to transfer to a label that doesn't exist. Attempt to send a message to a station not logged in. Attempt to CLEAR, ERASE, or delete a non-existent identifier or <user code>. Attempt to assign a <user code> already assigned.

FLYKITE User is politely told to go fly a kite for attempting to calculate too large an integer or real number, for attempting to make too large an array, in general, for trying something unreasonable.

SP FULL Users have managed to fill all allowable temporary storage without completing a calculation. Also, a <basic statement> has used too many global variables or is too complex.

SYSTEM Something weird happened but APL recovered enough to continue. Please save your listing and give it and the time the error occurred to the APL systems programmers.

APPENDIX A - SYNTAX

<apl program> ::=)<login>←<statement set>←)<logout>
<login> ::= {normal job activation process}
<user code> ::= {user account number}
<logout> ::= ØFF<off option>
<off option> ::= DISCARD|<empty>
<statement set> ::= <statement>|<statement set>←<statement>
<statement> ::= <monitor command>|<apl statement>|<empty>
<monitor command> ::=)<command>
<command> ::= <library maintenance>|CLEAR|ERASE<identifier list>|FNS|
VARS|SI|SIV|ABØRT|STØRE|<buffer edit>|<run parameter>
<library maintenance> ::= LØAD<library name>|<copy>|<clear>|<save>|FILES
<library name> ::= <identifier>
<copy> ::= CØPY<library name><copy name>
<copy name> ::= <stored program>|<variable name
<stored program name> ::= <identifier>
<variable name> ::= <identifier>
<clear> ::= CLEAR<library name>
<save> ::= SAVE<library name><lock option>
<lock option> ::= LØCK|<empty>
<identifier list> ::= <identifier>|<identifier list><space><identifier>
<buffer edit> ::= "<line edit>
<line edit> ::= <search string>"<insert string><quote option>
<search string> ::= <proper string>|<empty>
<insert string> ::= <proper string>|<empty>
<quote option> ::= "<search string>|<empty>
<run parameter> ::= <parameter type><number>|SYN|NØSYN|<parameter type>
<parameter type> ::= ØRIGIN|WIDTH|DIGITS|SEED|FUZZ

APPENDIX A (Continued)

<apl statement> ::= <stored program definition>|<basic statement>
<stored program definition> ::= $<definition entry><stored program body>$
<definition entry> ::= <stored program name>|<header>
<header> ::= <stored program options><local variables>←
<stored program options> ::= <function specifier><parameter options>
<function specifier> ::= <variable name> := |<empty>
<parameter options> ::= <niladic name>|<monadic name><formal parameter>|<formal parameter><dyadic name><formal parameter>
<niladic name> ::= <niladic subroutine name>|<niladic function name>
<dyadic name> ::= <dyadic subroutine name>|<dyadic function name>
<monadic name> ::= <monadic subroutine name>|<monadic function name>
<niladic subroutine name> ::= <identifier>
<niladic function name> ::= <identifier>
<dyadic subroutine name> ::= <identifier>
<dyadic function name> ::= <identifier>
<monadic subroutine name> ::= <identifier>
<monadic function name> ::= <identifier>
<formal parameter> ::= <identifier>
<local variables> ::= <local set>|<empty>
<local set> ::=; <identifier>|<local set>;<identifier>
<stored program body> ::= <stored program statement>|<stored program body>←<stored program statement>
<stored program statement> ::= <edit>|<compound statement>|<empty>
<edit> ::= [<edit command>
<edit command> ::= <resequence>|<display>|<insertion>|<change>|<delete>
<resequence> ::= ⌶OTA]
<display> ::= <line option>[]<line option>]
<line option> ::= <line reference>|<empty>
<line reference> ::= <label expression>|<number>
<label expression> ::= <identifier><relative location>
<relative location> ::= <direction><number>|<empty>

APPENDIX A (Continued)

<direction> ::= +|-
<insertion> ::= <line reference>]<compound statement>
<change> ::= <line option>["]<line option>]<line edit>
<delete> ::= <line reference>]<delete option>
<delete option> ::= [<line reference>]|<empty>
<compound statement> ::= <label set><basic statement>
<label set> ::= <label>|<label set><label>|<empty>
<label> ::= <identifier>:
<basic statement> ::= <expression>|<subroutine call>|<transfer statement>
<expression> ::= <operand>|<assignment statement>|<left part><expression>
<operand> ::= <constant>|<identifier><subscript option>|(<expression>)|[]|["]|<niladic function name>
<subscript option> ::= [<subscript list>]|<empty>
<subscript list> ::= <subscript>|<subscript list>;<subscript>
<subscript> ::= <expression>|<empty>
<assignment statement> ::= <assign operand>:=<expression>
<assign operand> ::= <identifier><subscript option>
<left part> ::= <monadic operator>|<operand><dyadic operator>
<monadic operator> ::= <monadic function name>|<monadic scalar operator>|<monadic mixed operator>|<monadic suboperator>
<monadic scalar operator> ::= +|-|&|%|*|LØG|CEIL|FLR|ABS|FACT|RNDM|NØT|CIRCLE
<monadic mixed operator> ::= ,|RHØ|IØTA|BASVAL|TRANS|XEQ|EPS
<monadic suboperator> ::= <monadic suboperator type><dimension part>
<monadic suboperator type> ::= <reduction type operator>|PHI|SØRTUP|SØRTDN
<reduction type operator> ::= <dyadic scalar operator>/|<dyadic scalar operator>\
<dimension part> ::= [<expression>]|<empty>
<dyadic operator> ::= <dyadic function name>|<dyadic scalar operator>|<dyadic mixed operator>|<dot operator>|<dyadic suboperator>

APPENDIX A (Continued)

<dyadic scalar operator> ::= +|-|&|*|LØG|MAX|MIN|%|RESD|CØMB|AND|ØR|NAND|NØR|LSS|LEQ|=|GEQ|GTR|NEQ|CIRCLE

<dyadic mixed operator> ::= EPS|,|RHO|IOTA|;|BASVAL|REP|RNDM|TAKE|DROP|TRANS

<dot operator> ::= <dyadic scalar operator>.<dyadic scalar operator>

<dyadic suboperator> ::= <dyadic suboperator type><dimension part>

<dyadic suboperator type> ::= PHI|/|\

<subroutine call> ::= <operand><dyadic subroutine name><expression>|<monadic subroutine name><expression>|<niladic subroutine name>

<transfer statement> ::= =:<expression>

APL SYNTAX - CONSTANTS & IDENTIFIERS

<data element> ::= <identifier>|<constant>

<identifier> ::= <letter>|<identifier><letter>|<identifier><digit>

<letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<constant> ::= <number>|<string>

<number> ::= <decimal number><exponent part>|<decimal number>|<exponent part>

<decimal number> ::= <integer><decimal fraction>|<integer>|<decimal fraction>

<integer>::= <unsigned integer>|+<unsigned integer>|#<unsigned integer>

<unsigned integer> ::= <digit>|<unsigned integer><digit>

<decimal fraction> ::= .<unsigned integer>

<exponent part> ::= <exponent symbol><exponent sign><unsigned integer>

<exponent symbol> ::= @|E

<exponent sign> ::= #|-|+|<empty>

<empty> ::= {the null string of symbols}

<string> ::= "<proper string>"

<proper string> ::= <string element>|<proper string><string element>

APPENDIX A (Continued)

<string element> ::= <string character>| ""
<string character> ::= <visible string character>| <space>
<visible string character> ::= <letter>| <digit>| <special symbol>
<special symbol> ::= .|(|)|,|&|$|*|+|;|:|#|%|=|@|/|\|[|]|-
<space> ::= <single space>| <space><single space>
<single space> ::= {a single unit of horizontal spacing which is blank}

APPENDIX B
SUMMARY OF <EDIT COMMAND>S

Display: [<line option>[]<line option>]

[[]]	Display the entire stored program.
[2[]]	Display line 2 of the stored program.
[LAB-3[]LAB+5]	Display lines LAB-3 through LAB+5 of the stored program.

Insertion: [<line reference>]<compound statement>

[46.5]A:=2	Insert the line A:=2 at position 45.6 of the stored program and change line increment to .1, replacing line 45.6 if it existed.
[LAB]A:=2	Replace the statement on line labelled LAB with A:=2.

Change: [<line option>["]<line option>]<search string>"<insert string>"<search string>
or <empty>

[["]]AAA"BBB"CCC	For every line of the stored program, find the first occurrence of the string AAA, insert the string BBB after it, and delete all characters until the string CCC is found.
[2["]]AAA"BBB"CCC	Perform above operation on line 2 of the stored program only.
[LAB["]RAB]AAA"BBB"CCC	Perform above operation on lines labelled LAB through RAB, inclusive.
[2["]]LAB"R"	Search line 2 for the string LAB and insert an R directly after it. The rest of the line remains unchanged.
[2["]]LAB"R	Same as above.

Delete: [<line reference>][<delete option>]

[2]	Delete line 2 of the stored program.
[2][7]	Delete lines 2 through 7.

Resequence: [I0TA] Resequences the lines of the stored program, starting at [1] with an increment of 1.

APPENDIX C
SUMMARY OF <MONITOR COMMAND>S

)FILES	Lists the <library name>s of the user's APL workspace library files that have been SAVEd and are present on disk.
)SAVE XXX	Saves the current active workspace on disk and associates /XXX and the user's <USERCODE> with it for future reference. (i.e.,)SAVE WKA will save the current active workspace by creating a file on disk called "/WKA"/<USERCODE>
)SAVE XXX LOCK	Same as above, except file is locked and may only be loaded by executing APL with the same <USERCODE> that the file was SAVEd under.
)LOAD XXX	Adds the workspace SAVEd under "/XXX"/ <USERCODE> to the active workspace.
)COPY XXX FUNC	Adds the <stored program definition> or variable named FUNC, which is SAVEd in file "/XXX"/<USERCODE>, to the active workspace.
)CLEAR	Removes current workspace and provides a "clean" workspace.
)CLEAR XXX	Removes the SAVEd workspace "/XXX"/ <USERCODE> from disk.
)ERASE NTØP	Removes the variable or <stored program definition> named NTØP from your workspace.
)FNS	Lists the stored program names in your active workspace.
)VARS	Lists the stored program names (followed by "(F)") and variable names in your active workspace.
)SI	Lists names of stored programs which have been suspended (suspended mode).
)SIV	Lists names of stored programs which have been suspended and the names of their local variables (suspended mode).
)ABØRT	Terminates all suspended stored programs and returns user to execution mode (suspended mode).

APPENDIX C (Continued)

)STØRE	When in suspended mode, stores values of global variables which have been changed while in stored program execution mode.
)"<search string>"<insert string> "<search string> or <empty>	When in execution mode, edits last input line according to specifications given after the)" (see APPENDIX B).
)ØRIGIN 3	*Changes ØRIGIN of subscripts on arrays to 3.
)WIDTH 65	*Changes width of output line to 65 characters.
)DIGITS 5	*Changes number of digits written after decimal point to 5.
)SEED 47235475	*Changes current base of pseudo-random number generator used with the operator RNDM to 47235475.
)FUZZ 1@-9	*Changes "fuzz factor" 10^{-9} used in relational tests to 10^{-9}.
)SYN	Causes lines input during <stored program definition> mode to be checked for syntax.
)NØSYN	Turns off)SYN--syntax check not done in <stored program definition> mode.

---

*If no number is given, APL types the current value of the <run parameter> specified.

APPENDIX D
INDEX TO APL/B5500 SYMBOLS AND THEIR APL\360 EQUIVALENTS

APL\360	APL/B5500	MONADIC FORM REFERENCE		DYADIC FORM REFERENCE	
+	+	identity	6.1	addition	6.4
-	-	add. inverse	6.1	subtraction	6.4
×	&	sign	6.1	multiplication	6.4
÷	%	mult. inverse	6.1	division	6.4
*	*	exponential	6.1	exponentiation	6.4
⍟	LØG	natural log.	6.1	logarithm	6.4
⌈	CEIL or MAX	ceiling	6.1	maximum	6.4
⌊	FLR or MIN	floor	6.1	minimum	6.4
|	ABS or RESD	absolute value	6.1	residue	6.4
!	FACT or CØMB	factorial	6.1	combinatorial	6.4
?	RNDM	random number	6.1	random deal	6.5
~	NØT	negation	6.1		
○	CIRCLE	circular	6.1	circular	6.4
<	LSS			less than	6.4
≤	LEQ			less or equal	6.4
=	=			equals	6.4
≠	NEQ			not equal	6.4
≥	GEQ			greater or equal	6.4
>	GTR			greater than	6.4
∧	AND			and	6.4
∨	ØR			or	6.4
⍲	NAND			nand	6.4
⍱	NØR			nor	6.4
⍳	IØTA	index generator	6.2	indexing	6.5
⍴	RHØ	dimension vector	6.2	restructuring	6.5
,	,	ravel	6.2	catenation	6.5
⍉	TRANS	transpose	6.2	dyadic transpose	6.5
⊥	BASVAL	base-2 value	6.2	base value	6.5
none	XEQ	execute string	6.2		
⊤	REP			representation	6.5
∈	EPS	execute string	6.2	membership	6.5
↑	TAKE			take	6.5
↓	DRØP			drop	6.5
;	;			het. output	6.5
/	/	reduction	6.3	compression	6.7
\	\	scan	6.3	expansion	6.7
⌽	PHI	reversal	6.3	rotation	6.7
⍋	SØRTUP	sorting up	6.3		
⍒	SØRTDN	sorting down	6.3		
.	.			inner/outer prod.	6.6

		USAGE	REFERENCE
	←	end of line signal	1.2
⎕	[]	input/display	8.1/8.2
⍞	["]	character input	8.1
→	=: or GØ	transfer	5.3
←	:=	assignment	5.1.3
[...;...;...;...]	[..;..;..;..]	subscripts	8.3
¯	#	minus sign	4.2.1
E	@ or E	power of ten	4.2.1
∇	$	stored program definition	7

APPENDIX E - EXAMPLES

```
      APL/B5500 UW COMPUTER SCIENCE # 01-11-71
       LOGGED IN THURSDAY 01-28-71  08:13
      ? )VARS←
 INTERP (F) LAGRANG(F) POLY   (F) SUM     VECT
      ? VECT←
2  4  6  8  10  12

      ? +/VECT←
42
      ? 4 6 RHO VECT←
2  4  6  8  10  12
2  4  6  8  10  12
2  4  6  8  10  12
2  4  6  8  10  12

      ? 1 1 TRANS 4 6 RHO VECT←
2  4  6  8

      ? VECT +.& TRANS VECT←
364
      ? XEQ "VECT+VECT"←
4  8  12  16  20  24

      ? SUM←
-0.026041667  0.520833333  -3.645833333  11.416666667  -8

      ? SUM[2 3 5]←
0.520833333  -3.645833333  -8

      ? $Z:= FIB N←
[1]   ? =: 0&IOTA Z:=N LSS 2←
[2]   ? Z:=(FIB N-1)+FIB N-2←
SYNTAX ERROR AT  N-2
[3]   ? [[]]←

      Z:= FIB N
[1]   =: 0&IOTA Z:=N LSS 2
[2]   Z:=(FIB N-1)+FIB N-2

[3]   ? [1["]]TA"(Z:=N)" L←
[3]   ? [1[]]←

[1]   =: 0&IOTA(Z:=N) LSS 2

[3]   ? $←
      ? FIB 2←
1
      ? FIB 5←
5
      ? FIB 3←
2
      ? FIB 4←
3
```

APPENDIX E (Continued)

```
      ? $X:=FX NEWTON DFX; ERR←
[1]   ? X:=X0←
[2]   ? =:((ABS ERR) GEQ 0#6)/2,0 RHO X:=X-ERR:=EPS FX,"%",DFX←
[3]   ? $←
      ? "((X*2)-2" NEWTON "2&X"←
SYNTAX ERROR AT  (X*2)-2%2&

NEWTON
[2]   SYNTAX ERROR AT  EPS FX,"%"

      ? )SIV←
      NEWTON  S  DFX     ERR     FX      X
      ? DFX←
2&X

      ? ERR←
      ? FX←
((X*2)-2

      ? X←
      ? FIB 3←
2
      ? )SIV←
      NEWTON  S  DFX     ERR     FX      X
      ? FX:=(X*2)-2←
DOMAIN ERROR AT  *2)-2
      ? FX:="(X*2)-2"←
      ? =:2←
NEWTON
[2]   DOMAIN ERROR

      ? )SIV←
      NEWTON  S  DFX     ERR     FX      X
      ? X:= 0←
      ? =: 2←
NEWTON
[2]   DOMAIN ERROR

      ? )ABORT←
      ? X0:=1←
      ? "(X*2)-2" NEWTON "2&X"←
1
      ? FIB 1←
1
      ? X0:=2←
      ? FUNC:="(X*2)-2"←
      ? DERIV:="2&X"←
      ? $NEWTON [[]]←

      X:=FX NEWTON DFX; ERR
[1]   X:=X0
[2]   =:((ABS ERR) GEQ 0#6)/2,0 RHO X:=X-ERR:=EPS FX,"%",DFX
[3]   ?
        $←
```

APPENDIX E (Continued)

```
      ? X:=1←
      ? EPS FUNC,"%",DERIV←
0
      ? (X*2)-2%2&X←
0
      ? FUNC:="(",FUNC,")"←
      ? FUNC←
((X*2)-2)

      ? FUNC NEWTON DERIV←
1.414213562
      ? $INTERP [[]]$←

      INTERP;X;Y;Z;D;N
[1]   =:(0=&/(IOTA N:=RHO X)=X IOTA X:=[])/UNIQERR,0 RHO []:="INPUT X VA
LUES"
[2]   =:(N NEQ RHO Y:=[])/DIMERR,0 RHO []:="INPUT Y VALUES"
[3]   =:(N GEQ D:=X IOTA Z:=[])/FOUNDZ,0 RHO []:="INPUT VALUE TO INTERPO
LATE"
[4]   =:0,0 RHO []:="INTERPOLATED VALUE IS";+/(Y&(&/D)%D:=Z-X)%&/(N,N-1)
RHO((N*2)RHO 0,N RHO 1)/,X CIRCLE . -X
[5]   FOUNDZ: =:0,0 RHO []:="INTERPOLATED VALUE IS";Y[D]
[6]   UNIQERR: =:0,0 RHO []:="X VALUES NOT UNIQUE ERROR"
[7]   DIMERR: "DIMENSIONS DO NOT MATCH  ERROR"

      ? INTERP←
INPUT X VALUES

[]:
      ? 2 4 6←
INPUT Y VALUES

[]:
      ? 1 2 3←

INPUT VALUE TO INTERPOLATE

[]:
      ? 10←



INTERPOLATED VALUE IS 5

      ? INTERP←
INPUT X VALUES

[]:
      ? 1 2 3←
INPUT Y VALUES

[]:
      ? 2 3 4 5 6←
DIMENSIONS DO NOT MATCH  ERROR
```

APPENDIX E (Continued)

```
      ? INTERP←
INPUT X VALUES

[]:
      ? 2 4 2←
X VALUES NOT UNIQUE ERROR

      ? INTERP←
INPUT X VALUES

[]:
      ? 1.6 49.3 12←
INPUT Y VALUES

[]:
      ? 3 64 5←
INPUT VALUE TO INTERPOLATE

[]:
      ? 1←
INTERPOLATED VALUE IS 3.076867953

      ? $LAGRANGE [[]]$←

      LAGRANGE;INDEX;X;Y;COMPV
[1]   "INPUT POINT PAIRS"
[2]   INPUTL: Y:=Y,[]
[3]   "ARE THOSE ALL THE POINTS"
[4]   SUM:=["]
[5]   =:INPUTL&IOTA"Y"NEQ SUM[1]
[6]   X:=(COMPV:=(RHO Y) RHO 1 0)/Y
[7]   Y:=(NOT COMPV)/Y
[8]   COMPV:=(SUM:=0),(#1+RHO X)RHO 1
[9]   BACKL: SUM:=SUM+(POLY COMPV/X)&Y[INDEX]%&/COMPV/X[INDEX:=(NOT COM
V)IOTA 1]-X
[9.5]  =:(1 RHO COMPV:=#1 PHI COMPV)/BACKL
[10]  SUM:=(0 NEQ +\(0 NEQ SUM))/SUM
[11]  SUM

      ? $POLY[[]]$←

      Z:=POLY X;LENGX
[1]   =:2&(LENGX:=RHO,X)GEQ RHO Z:=1,-1 RHO X
[2]   =:2&LENGX GEQ RHO Z:=(Z,0)+0,Z&-X[RHO Z]
      ? LAGRANGE←
INPUT POINT PAIRS

[]:
      ? 2 4 6←
ARE THOSE ALL THE POINTS

? NO←
[]:
      ? 12 7 14←
ARE THOSE ALL THE POINTS

? YES←
2  1.1641532180-10
```

APPENDIX E (Continued)

```
      ? POLY 1 1 1←
1   -3   3   -1

      ? POLY 5 RHO 1←
1   -5   10   -10   5   -1

      ? POLY 10 RHO 1←
1   -10   45   -120   210   -252   210   -120   45   -10   1

      ? POLY 2 3 4←
1   -9   26   -24

      ? POLY 2 #3←
1   1   -6

      ? VECT←
2   4   6   8   10   12

      ? POLY VECT←
1   -42   700   -5880   25984   -56448   46080

      ? LAGRANGE←
INPUT POINT PAIRS

[]:
      ? VECT←
ARE THOSE ALL THE POINTS

? NO←
[]:
      ? VECT←
ARE THOSE ALL THE POINTS

? YES←
LAGRANG
[9]   DOMAIN ERROR AT  %&/COMPV/X

      ? )SIV←
      LAGRANG S  BACKL   COMPV   INDEX   INPUTL  X        Y
      ? COMPV←
0   1   1   1   1   1

      ? INDEX←
1
      ? INPUTL←
2
      ? X←
2   6   10   2   6   10

      ? Y←
4   8   12   4   8   12

      ? INDEX IOTA 1←
1
```

APPENDIX E (Continued)

```
      ? X←
2  6  10  2  6  10

      ? X[1]-X←
0  -4  -8  0  -4  -8

      ? COMPV/X[1]-X←
-4  -8  0  -4  -8

      ? 1 DROP VECT←
4  6  8  10  12

      ? #1 DROP VECT←
2  4  6  8  10

      ? )ABORT←
      ? VECT:=#1 DROP VECT←
      ? VECT←
2  4  6  8  10

      ? LAGRANGE←
INPUT POINT PAIRS

[]:
      ? VECT,VECT←
ARE THOSE ALL THE POINTS

? YES←
-0.026041667  0.520833333  -3.645833333  11.416666667  -8

      ? SUM←
-0.026041667  0.520833333  -3.645833333  11.416666667  -8

      ? )VARS←
DERIV   FIB    (F) FUNC    INTERP (F) LAGRANG(F) NEWTON (F) POLY   (F)
SUM     VECT    X       XO
      ? )ERASE DERIV←
      ? )ERASE FUNC←
      ? )ERASE X←
      ? )VARS←
FIB    (F) INTERP (F) LAGRANG(F) NEWTON (F) POLY   (F) SUM     VECT

      ? )FNS←
FIB     INTERP  LAGRANG NEWTON  POLY
      ? )OFF←
      END OF RUN
```

APPENDIX E (Continued)

```
      APL/B5500 UW COMPUTER SCIENCE # 01-11-71
       LOGGED IN FRIDAY 01-29-71  02:35 PM
      ? A:=3 4 5 RHO IOTA 60←
      ? 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  25
26  27  28  29  30
31  32  33  34  35
36  37  38  39  40

41  42  43  44  45
46  47  48  49  50
51  52  53  54  55
56  57  58  59  60

      ? A[1;2;4]←
 9
      ? A[2;3;]←
31  32  33  34  35

      ? A[;3;1]←
11  31  51

      ? A[;;1]←
 1   6  11  16
21  26  31  36
41  46  51  56

      ? A[2 3;1;1]←
21  41

      ? A[1;2 3 RHO 1 2 3 2 4 1;2]←
 2   7  12
 7  17   2

      ? A[;2 3 RHO 1 2 4 3 1 1 ;2]←
 2   7  17
12   2   2

22  27  37
32  22  22

42  47  57
52  42  42

      ? B:="ABCDEFGHIJKLMNOPQRSTUVWXYZ "←
      ? B[2 7 RHO 27 27 23 1 12 12 1]←
 WALLA
 WALLA

      ? )OFF←
      END OF RUN
```

APPENDIX F

OCTAL EQUIVALENTS FOR B5500 CHARACTER CODES

Character	Dec.	Octal	Character	Dec.	Octal
blank	48	60	H	24	30
.	26	32	I	25	31
[	27	33	⊗	26	32
(	29	35	J	33	41
<	30	36	K	34	42
←	31	37	L	35	43
&	28	34	M	36	44
$	42	52	N	37	45
*	43	53	O	38	46
)	45	55	P	39	47
;	46	56	Q	40	50
≤	47	57	R	41	51
−	44	54	≠	60	74
/	49	61	S	50	62
,	58	72	T	51	63
%	59	73	U	52	64
=	61	75	V	53	65
]	62	76	W	54	66
"	63	77	X	55	67
#	10	12	Y	56	70
@	11	13	Z	57	71
:	13	15	0	0	00
>	14	16	1	1	01
≥	15	17	2	2	02
+	16	20	3	3	03
A	17	21	4	4	04
B	18	22	5	5	05
C	19	23	6	6	06
D	20	24	7	7	07
E	21	25	8	8	10
F	22	26	9	9	11
G	23	27	?	12	14

APPENDIX G

A COMPARISON OF

APL/B5500

and

APL/360

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

1. Introduction

An attempt is made in the following to compare and contrast APL/B5500 and APL\360 for the purpose of aiding in a transfer from one system to the other. Introducing APL or instructing in its use is not the goal of this effort; rather, the goal is describing and explaining the differences in two language implementations. It is assumed that the reader has some knowledge of APL.

Very little emphasis is placed on the equipment itself, with the major areas of discussion comprising communication with the systems and the language differences. Backus-Naur notation, which is used to define APL/B5500, is purposely avoided, since it is felt that while it may be an aid in formal language definition, it is not necessarily useful in learning the actual use of a language.

This APL/B5500 Manual (1) has been the sole available reference for the B5500 implementation. APL\360: User's Manual (2), APL\360 Student Primer (3), and the authors' own experience have provided the information on APL\360.

Frequently, the device of presenting a side-by-side comparison, where similar, identical, or equivalent commands, operators, or features are placed on the same line, is used in this paper. Upper-case letters in the text indicate keywords which are required or are actual system responses. Lower-case letters used in APL statements or system commands and responses describe variable information which appears in the position indicated. Braces ({}) indicate optional items.

2. Terminal Equipment

APL/B5500 supports the use of a Teletype model 33, 35, or 37. APL\360 is used with the IBM 2741, 2740, or 1050, all of which are basically Selectric typewriters utilizing removable typeheads. The use of Teletype terminals restricts the APL/B5500 character set considerably, as may be seen in Section 8.1. With the Selectric typewriter terminals, use may be made of different typeheads with APL programs that are designed to do so. Further discussion of terminal use for APL\360 is limited to the 2741, which is the most commonly used terminal. Special instructions for the 2740 and 1050 are included in the IBM manuals (2,3).

The difference in terminal equipment requires slightly different techniques for transmitting APL statements to the computer.

Teletype	2741
Terminate each logical transmission with "←".	Terminate each logical transmission with RETURN.
To transmit more than one line (72 characters) as a single logical transmission, depress LINE FEED and CARRIAGE RETURN. Continue typing. (200 characters maximum)	Inconvenient to transmit more than one line (130 characters). (May be done by manually returning carrier to left. Line limit not specified.)
No keyboard lock.	Keyboard locked during transmission and processing.
Automatic indention of 6 spaces when ready for next input.	Automatic indention of 6 spaces and keyboard unlock when ready for next input.
To suspend execution of a stored program or terminate output, depress BREAK key.	To interrupt execution or terminate output, depress ATTN key.

It is possible to correct typing errors before transmission on the 2741. This is done by backspacing to the error, depressing ATTN (which spaces down one line, types a caret, and spaces down one more line for the correction), and typing the line again from that point. Apparently this capability is not available with the Teletype. However, the APL/B5500 buffer edit feature (Section 7.) provides a similar capability.

3. Starting and Ending an APL Session

Information is not available on activating the APL/B5500, as it is implementation-dependent. The procedure does require the use of a B5500 account number. APL\360 is activated without a 360 account number, and the APL\360 user code is used as an account number. Directions for activating APL\360 (operation of the data set, etc.) are available in the IBM publications (2,3), with the telephone number the only variable.

3.1 Login

Once activation of the system has been accomplished, login procedures are somewhat similar, except that APL/B5500 will respond to activation by typing APL/B5500 UW COMPUTER SCIENCE #N (where N is a version identification). On APL/360, an account number is entered by typing)code. On APL/B5500, the user's usercode is used as the account number and is not required nor allowed to be entered. For unsuccessful login attempts, the system responds as follows.

	APL\360	APL/B5500
(a)	INCORRECT SIGN-ON	
(b)	NUMBER NOT IN SYSTEM for invalid number or password required	
(c)	NUMBER LOCKED OUT authorization for number withdrawn	
(d)	NUMBER IN USE	USER ALREADY LOGGED IN

Subsequent attempts are permitted in both systems.

Successful attempts cause the following response:

port #) time date user name code LOGGED IN date time

APL\360

{SAVED time date}

3.2 Workspace Activation at Login

A workspace is activated for the user upon completion of a successful login. With APL/B5500 the active workspace is the last active workspace, or if there is none, a new workspace is created. If the last active workspace has been removed without the user's knowledge, he is informed by the message WORKSPACE EMPTY.

Workspace activation at login is different for APL\360. The last active workspace is activated only if it was saved in CONTINUE

with no password at the termination of the last session. The termination could have been with)CONTINUE or)CONTINUE HOLD, or it could have been an abnormal termination with automatic saving of the workspace under the name CONTINUE. The SAVED message indicates that CONTINUE has been activated and gives the date and time it was saved.

3.3 Logout

To terminate the session, the user types:

APL\360	APL/B5500
(a))CONTINUE HOLD {lock}	(a))OFF

Save the active workspace and wait for another login.

(b))OFF HOLD {lock}	(b))OFF DISCARD

Do not save the active workspace, but wait for another login.

(c))CONTINUE {lock}

As in (a), but disconnect.

(d))OFF {lock}

As in (c), but disconnect.

APL/B5500 responds with END OF RUN. APL\360 responds with time, date, and CONTINUE for cases (a) and (c) only, and with accounting information for all cases.

3.4 APL\360 User Code Locks

A feature not available with APL/B5500 is the use of a user code password to prevent an unauthorized login under a specific user code. To assign a password to the user code, any of the four logout commands listed above for the APL\360 system may be followed by a colon

and any eight-character password. Subsequent logins must contain the colon and identical password as part of the user code. To remove the password, sign off with the colon only. To change the password, sign off with the colon followed by the new password. Failure to login with the correct password causes the response NUMBER NOT IN SYSTEM.

4. Workspaces and Workspace Commands

The workspace concept is identical in both APL/B5500 and APL\360; however, there are differences in the system commands regarding the workspace. The commands contrasted in this section deal exclusively with the active workspace, which is the workspace currently loaded and accessible from the terminal.

4.1 Run Parameters

The seven commands below constitute what are referred to as "run parameters" in APL/B5500 terminology. For APL/B5500 the commands when followed by an appropriate argument are orders to change the current setting of the run parameter specified. When not followed by an argument, the commands are inquiries to which the system responds by typing the current setting. An argument must follow the command for APL\360. The system responds by typing WAS and the former setting; thus, inquiry is possible even though an argument is required.

APL\360			APL/B5500		
command	initial value	restrictions	command	initial value	restrictions
)ORIGIN n starting index for arrays; starting point for monadic and dyadic ⍳ and ? .	1	0 or 1	)ORIGIN {n} starting index for arrays; starting point for modadic and dyadic IOTA and RNDM.	1	

APL\360 command	initial value	restrictions	APL/B5500 command	initial value	restrictions
)WIDTH n width of output line	120	30-130	)WIDTH {n} width of output line	72	10-72
)DIGITS n maximum number of significant digits on output	10	1-16	)DIGITS {n} maximum number of digits after decimal point on output	9	0-12
no corresponding command			)SEED {n} "seed" for random number generator	59823125	
no corresponding command 10^{-13} used for "fuzz"			)FUZZ {n} to counter truncation error and use in comparisons A=B if |A-B| ≤ FUZZ ×|B|	10^{-11}	
no corresponding command			)SYN causes APL to check syntax of each line of a stored program as it is entered	off	
no corresponding command			)NOSYN turns off syntax checker during stored program definition	on	

Note: APL\360 performs syntax checking only during execution of a defined function.

4.2 Workspace Maintenance and Interrogation

The following commands facilitate workspace maintenance and interrogation other than manipulation of run parameters. All apply to the active workspace only. (Note: the terms "stored program" (APL/B5500) and "defined function" (APL\360) may be used interchangeably.)

APL\360	APL/B5500
)CLEAR Discards active workspace. Resets run parameters to initial settings. Permitted at all times.	)CLEAR Discards active workspace except for run parameters. Not permitted in suspended mode.
)ERASE list Removes global names in list. Permitted at all times. Possible response: NOT ERASED: list.	)ERASE list Removes each global name in list. Not allowed in suspended mode.
)VARS {a} Lists alphabetically names of global variables. If {a} is used, where "a" is an alphabetic character, the list begins with those variables beginning with the indicated letter.	)VARS Lists all variables and stored programs. "(F)" marks stored programs.
)FNS {a} Lists alphabetically names of defined functions, or, if {a} is specified, the names beginning with the indicated letter and continuing through the alphabet.	)FNS Lists names of stored programs.
)GROUP names Collect global names into a group. First name in list is name of group.	no corresponding command
)GRPS {a} List alphabetically names of groups. {a} as in)VARS above.	no corresponding command
)GRP name List membership of designated group.	no corresponding command

APL\360	APL/B5500
)SI Lists halted functions and the line number in each where execution stopped. Suspended functions distinguished from pendent functions by *.	)SI Lists in order the names of suspended stored programs.
)SIV Same as)SI above, but with local variables listed for each entry in)SI list.	)SIV Same as)SI above, but with local variables listed for each entry in)SI list.
)WSID {name} {lock} Without {name}, response is name of active workspace. With {name}, name of active workspace is changed. Printed response is WAS followed by former name. A password may be associated with the workspace for security. (See Section 5.1.)	no corresponding command
no corresponding command ABORT may be accomplished by entering "→" for each asterisk in)SI list.	)ABORT Terminates all suspended stored programs and returns to execution mode.
no corresponding command (Variables are stored during execution.)	)STORE When in suspended mode, causes APL to store into the active workspace the values of global variables changed during execution of the suspended stored program.

Commands which request lists may receive the typed response NULL from APL/B5500 when the list are empty. For APL\360 the corresponding response is simply the standard indentation of six spaces and the freeing of the keyboard.

5. Libraries

Implementation of the notion of library is somewhat different in the two systems, although the commands which maintain and utilize libraries have some similarities. Consequently, a section is devoted to discussing the library concept for each system, after which the commands will be contrasted in the side-by-side fashion.

5.1 APL\360 Libraries

Each account number (user code or login number) is automatically assigned sufficient library space to contain a certain number of workspaces; this number is determined by the manager of the APL system. Each workspace within the library is identified by the account number and WSID. There is always a workspace with WSID CONTINUE in the library.

The library associated with the user code is a private library. Public libraries with workspaces containing defined functions of general interest are available to all users.

A password may be associated with a workspace name in the)WSID or the)SAVE commands. It is then necessary to use this password as a key to access the workspace after it has been placed in a library.

A workspace in a library may be changed or erased only from a terminal logged in under the same account number as that associated with the library. (If a password is associated with the account number, the password must be used for logging in to change a library workspace.) Library workspaces may be accessed by any user who knows the account number, WSID, and workspace password (if necessary).

5.2 APL/B5500 Libraries

A library may contain only one workspace. When created, a library is given a name consisting of the B5500 usercode under which APL was activated from the terminal and a "library prefix" or identifier chosen by the user. It is possible to exercise a LOCK option when creating a library. If exercised, this option prevents a user who is not logged into APL under the same B5500 usercode from accessing the library. Otherwise any user who knows the usercode and library prefix[1] may access the library, although it may be changed or deleted only from a terminal logged in under the B5500 usercode used in creating the library.

5.3 Comparison of Library Commands

APL\360	APL/B5500
)SAVE {name {lock}} Without the option a copy of the active workspace will replace a stored workspace with the same identification, or a new workspace may be stored if)WSID followed by a name has been executed previously. A password associated with the active workspace will continue in effect. With the {name {lock}} option, a copy of the active workspace will be stored with the name and password, if used. A previously-stored workspace with the same identification will be replaced with password in this command.	)SAVE library suffix {LOCK} Saves the current active workspace under the user B5500 account number and library suffix. {LOCK} explained above. FILE ALREADY ON DISK is a possible response of obvious meaning in which case the active workspace is not saved.

APL\360	APL/B5500
The active workspace will assume identification used in this command. Possible responses: If successful, time and date will be printed. Also WSID if name option not used. NOT WITH OPEN DEFINITION if terminal is in function definition mode. NOT SAVED, WS QUOTA USED UP Library full, and this is new workspace. NOT SAVED, THIS WS IS name WSID of active workspace does not match that in command. IMPROPER LIBRARY REFERENCE INCORRECT COMMAND	
)LOAD {library number[2]} wsid {key} Activates a copy of the stored workspace. No protection of currently active variables of same name. Response: If successful, SAVED time date WS NOT FOUND WS LOCKED INCORRECT COMMAND	)LOAD library name[3] Loads the stored workspace into the active workspace (subject to LOCK). When a name is in both the stored and active workspaces, the item is not loaded unless it is a variable in the active workspace and a stored program in the library. An appropriate message indicates items not loaded.
)PCOPY {library number[2]} wsid {key} object Copies the object designated unless there is already a global variable by that name in the active workspace. Object may be variable, function, or group. Response: If successful: SAVED time date.	)COPY library name[3] global name Copies the value of a variable or definition of a stored program from the specified workspace subject to LOCK and the matching-name rule in)LOAD.

APL\360	APL/B5500
Unsuccessful: NOT COPIED: name NOT WITH OPEN DEFINITION WS NOT FOUND WS LOCKED OBJECT NOT FOUND WS FULL INCORRECT COMMAND	
)PCOPY {library[2] / number} wsid {key} Copy all global objects from the specified workspace. Protect global variables in active workspace with identical names. Responses: same as for)PCOPY above.	no corresponding command
)COPY {library[2] / number} wsid {key} obj. Copy specified object into active workspace, replacing existing object if necessary. Responses: same as above except NOT COPIED	no corresponding command
)COPY {library[2] / number} wsid {key} Copy all global objects, replacing existing objects if necessary. Responses: same as above except NOT COPIED.	no corresponding command
)DROP wsid Remove designated workspace form library. Response: If successful, time and date printed. Unsuccessful: WS NOT FOUND IMPROPER LIBRARY REFERENCE INCORRECT COMMAND	)CLEAR library prefix Remove the library file referenced by complete library name (including B5500 usercode) from disk.

APL\360	APL/B5500
)LIB {library number[2]} List names of workspaces in designated library. Passwords are neither listed nor indicated. Unsuccessful responses: IMPROPER LIBRARY REFERENCE INCORRECT COMMAND	no corresponding command

6. User Communications

The following commands identify other current users of the system and permit communication with them and the operator.

APL 360	APL/B5500
)PORTS Lists port numbers and user-name codes.	)LOGGED Lists station numbers and user phrases.
)PORT user-name code Lists port numbers associated with current users identified by the specified user-name code.	no corresponding command
)MSGN port # text The text, not to exceed one line, is sent to the specified port. If the port number designated is not in use, the message is reflected back to the sender. Keyboard is locked during transmission. Successful transmission response is SENT and keyboard is unlocked. MESSAGE LOST response results in the event of a transmission disturbance.	?TO station text The text, not to exceed 200 characters, is sent to the specified station. If the station designated is not in use, NOT ON is printed.

APL\360	APL/B5500
At receiving terminal: sending port #) text	At receiving terminal: FROM sending-station text
)MSG port # text Same as for MSGN except receiving terminal receives "R" as prefix and sending keyboard remains locked (after SENT response) until receiving terminal replies.	no corresponding command
)OPRN text Same as for)MSGN except text is directed to sytem recording terminal.	?TO SPO text Same as for ?TO station except text is directed to operator.
)OPR text Same as for)MSG except text is directed to system recording terminal.	no corresponding command

7. APL/B5500 Buffer Edit

Buffer edit is an APL/B5500 feature not available with APL\360. It permits the user to change the last line of input without retyping the entire line. The command is:

$$)\text{"}\ \begin{matrix}\text{search}\\\text{string}\end{matrix}\ \text{"}\ \begin{matrix}\text{insert}\\\text{string}\end{matrix}\ \left\{\text{"}\ \begin{matrix}\text{search}\\\text{string}\end{matrix}\right\}\ .$$

The operation is the same as that for the change command used in editing stored programs (Section 9.3.4). The insert string is inserted after the first occurrence of the first search string. If the second search string is specified, characters between the insert string and the first occurrence of the second search string are deleted. The resulting is retyped by the system and then processed.

8. The APL Language

In this section an attempt is made to list only those differences in the language implementations of APL/B5500 and APL\360 which may be determined from the available information and which might be of some importance to a user of both systems. It may well be that actual use of the two systems would reveal other differences which should be included in this section. Particularly, it is possible that such items as one-element arrays of different rank would be treated differently when used as operands for various operations.

While data structures and other features are not discussed unless they do not receive equivalent treatment in the two systems, a reasonably complete tabular comparison of operators is given. These operator comparison charts should provide a useful reference for converting from one system to the other.

8.1 Character Set

APL\360

alphabetic:
ABCDEFGHIJKLMNOPQRSTUVWXYZ

numeric:
0123456789

blank:

special characters:
().,:;+-=/[]\←⊂∩⊃∪⊥⊤|⍺⌈⌊_∇
∆∘'⍞?⍵∈⍴~↑↓⍳○*→⍬⍤<≤≥>≠∨∧×÷

Certain pairs of APL\360 characters may be used in combination by backspacing and overstriking: A...Z ⍶ ⍹ ⍰ ⍟ ⍋ ⍒ ⌶ ⍲ ⍱.

APL/B5500

alphabetic:
ABCDEFGHIJKLMNOPQRSTUVWXYZ

numeric:
0123456789

blank:

special characters:
().,:;+-+/$#*@%&[]\←

8.2 Reserved Words

APL\360

No reserved words.

APL/B5500

LOG	CIRCLE	TRANS
CEIL	LSS	BASVAL
MAX	LEQ	XEQ
FLR	NEQ	REP
MIN	GEQ	EPS
ABS	GTR	TAKE
RESD	AND	DROP
FACT	OR	PHI
COMB	NAND	SORTUP
RNDM	NOR	SORTDN
NOT	IOTA	GO
	RHO	

8.3 Identifiers

APL\360

Alphabetic, numeric, or underscored alphabetic characters; first must be alphabetic. No maximum length stated. (May be longer than seven characters.)

APL/B5500

Up to seven alphabetic or numeric characters; the first must be alphabetic. No maximum length is stated; if over seven characters, the remainder is truncated.

8.4 Negative Numbers

APL\360

Negative numbers indicated by ¯ in all cases. (Note distinction between ¯ and -.)

APL/B5500

Negative numbers are indicated by #, except - permitted for a negative exponent in floating point notation.

8.5 Floating Point Notation

APL\360

Only "E" may precede the exponent for floating point.

APL/B5500

Either "E" or "@" may precede the exponent in floating point.

8.6 Precision

APL\ 360

Integers less than 2^{52} = 4501599627370496 are carried to full precision.

Larger numbers and non-integers are carried to a precision of about 16 digits.

APL/B5500

Numbers whose significant digits do not exceed $2^{39}-1$ = 549755813887 are carried to full precision.

Absolute values of all numbers must be less than about 4.314@68.

Numbers with absolute value less than 10@-47 are converted to zero.

All numbers carried in floating point form.

8.7 Unassigned Identifiers

APL\360

References to identifiers which have not been assigned values receive the error response VALUE ERROR.

APL/B5500

Identifiers not assigned values are considered to have the value null.

8.8 One-element Character Strings

APL\360

A single character is treated as a scalar.

```
X←'A'
ρX

(blank line)
```

APL/B5500

A single character is treated as a one-element vector, rather than as a scalar.

```
      X:="A"←
      RHO X←
1
```

8.9 Character-vector Input

APL\360

No limit for character vector input is stated, but more than one line's length is inconvenient.

APL/B5500

A character vector of up to 200 characters may be input directly.

8.10 Symbol Variations - Other than Operators

APL\360	APL/B5500
Quad.	
⎕	[]
Uses of the symbols are equivalent.	
Quote-quad.	
⍞	["]
Uses of the symbols are equivalent.	
Assign symbol.	
←	:=
Uses of the symbols are equivalent.	
Transfer or branching symbol.	
→	=: or GO

Uses of the symbols are equivalent, except that with APL\360 the symbol "→" above while in suspended mode clears the status indicator back to the last previous suspended function.

8.11 Subscripting

The subscripting capabilities are identical in the two systems, except that for APL/B5500 unnamed expressions cannot be subscripted.

8.12

Comparison of APL\360 and APL/5500 Operators

APL\360 and APL/5500 operators differ as to the structure of the operands to which they may be applied and as to the types of values the operands may have. When no distinction is made between restrictions on values of the operands, the restrictions are identical for operands applied to APL\360 and APL/5500 operators. Restrictions on structure will be indicated by the identifiers used for arguments as follows:

APL\360	APL/5500
S for scalar	X,Y for scalar or one element array
V for vector	V,W for vector or scalar
M for matrix	L,M for matrix or vector
A for any structure	A,B for any structure

Except as the first argument of S⍴A or S[A] in APL\360, a scalar may be used instead of a vector. Also in APL\360, a one element array may replace any scalar.

APL\360 and APL/5500 operators also differ as to the characters which symbolize the operations performed. Where no distinction is made between results of performing analogous operations, the results are identical for APL\360 and APL/5500 operators.

8.12.1

Monadic Scalar Operators with Identical Results

APL Symbol 360	APL Symbol 5500	Name of Operator 360	Name of Operator 5500	Form Used 360	Form Used 5500
+	+	plus	identity	+B	+B
-	-	negative	additive inverse	-B	-B
×	&	signum	sign	×B	&B
÷	%	reciprocal	multiplicative inverse	÷B	%B
*	*	exponential	exponential	*B	*B
⍟	*LOG*	natural logarithm	natural logarithm	⍟B	*LOG B*
⌈	*CEIL*	ceiling	ceiling	⌈B	*CEIL B*
⌊	*FLR*	floor	floor	⌊B	*FLR B*
|	*ABS*	magnitude	absolute value	|B	*ABS B*
!	*FACT*	factorial	factorial	!B	*FACT B*
?	*RNDM*	roll	random number	?B	*RNDM B*
~	*NØT*	not	negation	~B	*NØT B*
○	*CIRCLE*	pi times	circular	○B	*CIRCLE B*

8.12.2

Monadic Mixed Operators

APL Symbol 360	APL Symbol 5500	Name of Operator 360	Name of Operator 5500	Form Used 360	Form Used 5500	Distinctions 360	Distinctions 5500
⍳	*IOTA*	index generator	index generator	⍳*N*	*IOTA X*	N non-negative integer	X in integers If X less than ORIGIN, NULL vector.
⍴	*RHO*	size	dimension vector	⍴*A*	*RHO A*		
,	,	ravel	ravel	,*A*	,*A*		
⍉	*TRANS*	transpose	transpose	⍉*A*	*TRANS A*		
	XEQ		execution of character string		*XEQ V*		V in character set Result is the value of the APL expression given by V.
	BASVAL		base-2 value		*BASVAL V*		V in reals Result is 2 BASVAL V. (see dyadic mixed operators)

8.12.3

Monadic Suboperators (X in set of integers)

Name of Operator		Form Used		Distinctions or Restrictions	
360	5500	360	5500	360	5500
reduction	reduction	f/[X]A or f/A	d/[X]A or d/A	f-dyadic scalar operator Reduction over first coordinate of M obtained by using expression f≠M; equivalent to f[1]M.	d-dyadic scalar operator.
	scan		d\[X]A or d\A		Proceeds along Xth coordinate with respect to ORIGIN where X=(rank of A) +ORIGIN-1 if X not given. Result has same structure as A. Scan goes from left to right or top to bottom.
reverse	reversal	⌽[X]A or ⌽A	*PHI*[X]A or *PHI A*	⌽A denotes reversal along the first coordinate which is equivalent to ⌽[1]A.	A≠NULL
grade up	sorting up	⍋[X]A or ⍋A	*SØRTUP*[X]A or *SORTUP A*	A-array of rank greater than zero.	If A is a scalar, result is ORIGIN. A≠NULL
grade down	sorting down	⍒[X]A or ⍒A	*SORTDN*[X]A or *SORTDN A*	Same as grade up	Same as sorting up

Dyadic Scalar Operators with Identical Results

APL Symbol 360	APL Symbol 5500	Name of Operator 360	Name of Operator 5500	Form Used 360	Form Used 5500
+	+	plus	addition	*A+B*	*A+B*
-	-	minus	subtraction	*A-B*	*A-B*
×	&	times	multiplication	*A×B*	*A&B*
÷	%	divide	division	*A÷B*	*A%B*
*	*	power	exponentiation	*A*B*	*A*B*
⍟	*LØG*	logarithm	logarithm	*A⍟B*	*A LØG B*
⌈	*MAX*	maximum	maximum	*A⌈B*	*A MAX B*
⌊	*MIN*	minimum	minimum	*A⌊B*	*A MIN B*
|	*RESD*	residue	residue	*A|B*	*A RESD B*
!	*CØMB*	binomial coefficient	combinatorial	*A!B*	*A CØMB B*
<	*LSS*	*less	less than	*A<B*	*A LSS B*
≤	*LEQ*	*not greater	less than or equal	*A≤B*	*A LEQ B*
=	=	*equal	equals	*A=B*	*A=B*
≠	*NEQ*	*not equal	not equal	*A≠B*	*A NEQ B*
≥	*GEQ*	*not less	greater than or equal	*A≥B*	*A GEQ B*
>	*GTR*	*greater	greater than	*A>B*	*A GTR B*
∧	*AND*	and	and	*A∧B*	*A AND B*
∨	*ØR*	or	or	*A∨B*	*A ØR B*
⍲	*NAND*	nand	nand	*A⍲B*	*A NAND B*
⍱	*NØR*	nor	nor	*A⍱B*	*A NØR B*
○	*CIRCLE*	circular	circular	*A○B*	*A CIRCLE B*

*APL/5500 compares characters by using their octal equivalents. APL\360 compares scalars and does not make character comparisons.

8.12.5

Dyadic Mixed Operators

APL Symbol		Name of Operator		Form Used		Distinctions	
360	5500	360	5500	360	5500	360	5500
⍳	*IOTA*	index of	indexing	*V⍳A*	*V IOTA A*		If V is null, result is ORIGIN.
⍴	*RHO*	reshape	restructuring	*V⍴A*	*V RHO A*	If any element of V is 0, result is array with one of the dimensions equal to zero.	If any element of V is 0, the result is null.
,	,	catenation	catenate	*V,V*	*V,W*		
?	*RNDM*	deal	random deal	*S?S*	*X RNDM Y*		X less than or equal to Y-ORIGIN+1.
⊥	*BASVAL*	decode	base value	*V⊥V*	*V BASVAL W*	Arguments V and W must be of the same dimension except that either may be a scalar.	V is effectively modified to make the lengths of V and W equal by extending V to the left as many times as necessary.
⊤	*REP*	encode	representation	*V⊤S*	*V REP S*		
∈	*EPS*	membership	membership	*A∈A*	*A EPS B*		
↑	*TAKE*	take	take	*V↑A*	*V TAKE A*	If A is an array, then *V↑A* valid only if V has one element for each dimension of A.	If V is a scalar, it is extended to a vector of the appropriate length. If V(I)=0 for any I, result is null.

Dyadic Mixed Operators (continued)

APL Symbol		Name of Operator		Form Used		Distinctions	
360	5500	360	5500	360	5500	360	5500
⍉		transpose		*V*⍉*A*		Coordinate *I* of A becomes *V*[*I*] of result.	

8.12.6

Dyadic Suboperators (X in set of integers)

Name of Operator		Form Used		Distinctions or Restrictions	
360	5500	360	5500	360	5500
rotate	rotation	*A⌽[X]A* or *A⌽A*	*A PHI[X] B* or *A PHI B*	A⌽A denotes rotation along the first coordinate of A.	
compress	compression	*V/[X]A* or *V/A*	*V/[X]A* or *V/A*	V⌿A denotes compression along the first coordinate.	If A is null, result is NULL.
expand	expansion	*V\[X]A* or *V\A*	*V\[X]A* or *V\A*	V⍀A denotes expansion along the first coordinate.	

8.12.7

The Dot Operators

Name of Operator		Form Used		Distinctions or Restrictions	
360	5500	360	5500	360	5500
inner product	inner product	Md1.d2M	Ld1.d2M	M can be a matrix or a vector.	d1 not CIRCLE The first coordinate of L and the last coordinate of M are null if they are vectors.
outer product	outer product	*A*∘.g*A*	*A CIRCLE*.d*B*		

9. Stored Programs (Defined Functions)

The concept of permanently defining and storing a logical sequence of APL statements is identical in both systems. The APL/B5500 manual (1) makes the distinction between subroutines and functions, referring to both types as "stored programs." The APL\360 manuals (2,3) refer to both as "defined functions" but distinguish those which return an explicit result. (Returning an explicit result is the identifying feature of an APL/B5500 "function.")

9.1 Returned Value of a Function

APL\360	APL/B5500
No initialization. Will return VALUE ERROR if no assignment is made during function execution.	Initialized to null at each call.

9.2 Local and Global Variables

APL\360	APL/B5500
A variable which is local to function P1 may be referred to by any function P2 used within P1, unless the variable name is also local to P2.	A variable which is local to stored program P1 is not global to stored program P2 when P1 invokes P2.

9.3 Function Definition Mode

In both APL\360 and APL/B5500, a special mode of operation is to be entered for the purpose of defining and editing stored programs. An outline of the differences in function definition for the two systems follows.

9.3.1 Opening and Closing Definition Mode

APL\360	APL/B5500
∇	$

The symbols start and end function definition mode for the indicated systems.

9.3.2 Procedures of Function Definition

Basic procedures of entering a function definition are equivalent, although editing capabilities differ. Local variables follow semicolons at the end of the header line; labels are identifiers followed by colons at the beginning of a line; the system prints the line numbers. Equivalent sample function definitions follow.

9.3.2.1 Sample APL\360 Function Definition

```
      ∇S←STDEF X
[1]   AVE ← (+/X)÷N ← ⍴X
[2]   S ← ((+/(X- AVE)*2)÷N - 1)*.5
[3]   ∇
```

9.3.2.2 Sample APL/B5500 Stored Program Definition

```
      $S := STDEV X←
[1]   AVE := (+/X) % N := RHO X←
[2]   S := ((+/(X-AVE)*2) % N-1)*.5←
[3]   $←
```

9.3.3 APL\360 Function Definition Editing

It is hoped that the following examples of editing a function definition will be adequate for illustrating the editing capabilities of APL\360.

Display

[□]	Display the entire function.
[2□]	Display line 2.
[□2]	Display all statements from 2 onward.

Note: Label names may not be used. There is no capability for displaying from line m to line n.

Insertion

[5.5] *A*←2	Insert the line *A*←2 at position 5.5, replacing line 5.5 if it exists. Line increment is changed to 0.1 until next entry of a line number with a different implied increment.

Revision

[2] *A*←2	Replace line 2 with new statement.
[5□15] *A*←*B*←*C*←*F*←*E*←0 /1 2	Display line 5 and position carriage under position 15 of line 5. (15 is an estimate of position of first change required.) A digit, letter, or / may be typed under any position in line 5. When RETURN is depressed the line is retyped with each character understruck by / deleted, each character understruck by a digit k preceded by k blanks, and each character understruck by a letter preceded by 5×R blanks, where R is the letter's rank in the alphabet. The carriage moves to the first injected space. Corrections are now inserted and the statement is reentered as corrected.
[5] *A*←*B*←*C*← ←*E* ←0 ↑ ↑↑ insert *D* ←*F*	

Deletion

[3] ^	Delete statement 3. Type [3] followed by ATTN and RETURN.

APL\360 automatically renumbers the lines with integers when definition mode is closed by entering ∇.

Reopening a function definition, modifying the definition, and closing definition mode may be accomplished in one statement. For example: ∇*NAME*[3] *X*←*Y* ∇ .

9.3.4 APL/B5500 Stored Program Definition Editing

The following examples taken from Appendix B of the APL/B5500 manual (1) should suffice to illustrate the editing capabilities of APL/B5500.

Display: $[{}^{\text{line}}_{\text{option}}[]{}^{\text{line}}_{\text{option}}]$

[[]]	Display the entire stored program.
[2[]]	Display line 2 of the stored program.
[LAB-3[]LAB+5]	Display lines LAB-3 through LAB+5 of the stored program.

Insertion: $[\ {}^{\text{line}}_{\text{reference}}]$ APL statement

[46.5]A:=2	Insert the line A:=2 at position 46.5 of the stored program and change line increment to .1, replacing line 46.5 if it existed.
[LAB]A:=2	Replace the statement on line labelled LAB with A:=2.

Change: $[{}^{\text{line}}_{\text{option}}\ ["]\ {}^{\text{line}}_{\text{option}}]$ search string " insert string $\left\{{}^{"\text{search}}_{\text{string}}\right\}$

[["]]AAA"BBB"CCC	For every line of the stored program, find the first occurence of the string AAA, insert the string BBB after it, and delete all characters until the string CCC is found.
[2["]]AAA"BBB"CCC	Perform above operation on line 2 of the stored program only.
[LAB["]RAB]AAA"BBB"CCC	Perform above operation on lines labelled LAB through RAB, inclusive.

[2["]]LAB"R"	Search line 2 for the string LAB and insert an R directly after it. The rest of line remains as is.
[2["]]LAB"R	Same as above.

Delete: [line reference] {[line reference]}

[2]	Delete line 2 of the stored program.
[2][7]	Delete lines 2 through 7.

Resequence a function: [IOTA]

[IOTA]	Resequence open function.

9.3.5 APL\360 Locked Functions

With APL\360 a function may be locked to display (though not to)ERASE) by opening or closing its definition with ⍒ instead of ∇. This feature is not available with APL/B5500.

9.3.6 Syntax Checking

APL\360	APL/B5500
Syntax check is made only during execution.	Syntax checking at each line entry during program definition may be requested with)SYN,)NOSYN settings.

9.3.7 System Commands in Function Definition Mode

APL\360	APL/B5500
Most system commands may be executed. They are never taken to be part of a program definition.	Monitor commands are not executed but accepted as part of program definition. Will produce SYNTAX error at execution of program.

9.3.8 System Failure in Function Definition Mode

APL\360	APL/B5500
Definition mode closed, workspace stored in CONTINUE. Workspace reloaded at subsequent sign-on. User must reopen definition mode.	User gets message CONTINUED DEFINITION OF name at next login. He may continue from the point of failure at previous session.

9.4 Execution of Stored Programs

Stored programs are executed in both systems by a reference to the program name as indicated in the header line. Each line is evaluated in sequence as if it had been entered one line at a time in "calculator" mode. The following two comments regarding stored program execution apply to APL/B5500 only: 1) permanent changes to the workspace are not made until the invoking statement has been evaluated successfully, and 2) previously executed lines of a stored program are kept in a semi-compiled form to improve execution of loops.

9.4.1 Normal Termination

APL\360	APL/B5500
Normal termination results from: (a) "running off at the end" (executing the last statement when the last statement does not branch). (b) transferring to a statement number outside the bounds of the program. (c) executing →0.	Normal termination results from: (a) "running off at the end." (b) transferring to a statement not in the program definition. (c) executing=:0.

9.4.2 Abnormal Termination

For both systems, if any error occurs during execution of a stored program, the session is put in suspended mode, and the appropriate error message and line number of the statement in error are printed. (See Error Messages, Section 10, and Suspended Mode, Section 11.)

9.4.3 Trace Control

For APL\360 only, it is possible to set a trace vector, denoted $T\Delta P$, where P is the function to be traced, to any set of statement numbers in P. As the function is executed, an automatic printout is produced for each line of the function which is contained in the trace vector. The printout gives the function name, line number, and final value produced by that statement. The trace may be discontinued by typing $T\Delta P \leftarrow 0$.

9.4.4 Stop Control

Another feature of APL\360 only is the stop vector, $S\Delta P$, where P is a function name. By setting this vector to statement numbers in P, it is possible to cause execution to stop just before each of the designated statements. After the stop, the function name and statement number are printed, and the system is in the normal suspended state. Resumption of execution is controlled in the usual ways. (See Suspended Mode, Section 11.)

10. Error Messages

All error messages may occur in either "calculator" or stored program execution mode.

APL\360	APL/B5500
When an error is found, the name of the error is printed, the offending statement is retyped with a caret placed at the point where the error was found. Function name and line number also typed if in function execution mode.	When an error is found, the name of the error is printed with the six characters to the right of the point where the error was discovered. If in program execution mode, line number is also typed.

The following error messages are identical in appearance and interpretation for both systems.

DOMAIN
INDEX
RANK
SYNTAX
SYSTEM

The following error messages are identical in appearance, but there are slight differences in meaning.

APL\360	APL/B5500
DEPTH	
(a) excessive depth of function execution.	(a) execution stack is too long. (b) list of active and suspended stored programs is too long.
LABEL	
(a) name of already existing function used as label. (b) colon misused.	(a) attempt to transfer to non-existent label. (b) attempt to send a message to a station not logged in. (c) attempt to)ERASE a non-existent identifier.

APL\360	APL/B5500
LENGTH	
(a) shapes not conformable.	(a) arguments not conformable due to unequal lengths. (b) input or output line too long.

The following error messages are found in APL\360 but not in APL/B5500.

CHARACTER - illegitimate overstrike.

DEFN - misuse of ∇ or □.

RESEND - transmission failure.

SYMBOL TABLE FULL - too many names used.

VALUE - use of name which has not bee assigned a value.

WS FULL - workspace is filled.

The following error messages are found in APL/B5500 but not in APL\360.

NONCE - attempt to utilize a non-implemented feature.

FLYKITE - attempt to calculate too large a number; attempt to make too large an array, etc.

SP FULL - have filled all allowable temporary storage without completing a calculation; a statement uses too many global variables or is too complicated.

11. Suspended Mode

Suspended mode is attained when an error occurs in a stored program execution, the "break key" (ATTN) is depressed during execution of a stored program, or, for APL\360, just before execution of a statement whose number is in the appropriate stop vector.

11.1 Permissible Actions while in Suspended Mode

APL\360	APL/B5500
Display values of local and known global variables.	Display values of local and known global variables.
Change values of variables.	Change values of variables.
Execute system commands.	Execute monitor commands except)CLEAR and)ERASE.
Execute basic APL statements.	Execute basic APL statements.
Transfer (→) to any statement within suspended function.	Transfer (=:) to any statement within suspended function.
Transfer to 0 or to statement outside of suspended function. Causes normal termination of that function.	Return transfer to a statement outside suspended program. Causes function to become "unsuspended."
Reopen definition of suspended function or of any function which is not pendent.	Cannot enter stored program definition mode. Will receive message ABORT SUSP. FCNS
Set trace and stop control vectors.	
Enter → to clear state indicator of this suspended function and any pendent functions back to next last suspended function.	

12. System Dependent Functions

System dependent (or I-beam) functions are defined for APL\360 but not for APL/B5500. They permit inquiry of system information. The function is the ⊥ overstruck with T (or vice-versa) followed by a number. Here is a list of currently available system dependent functions with their definitions.

I19	Accumulated keying time this session.
I20	Time of day.
I21	CPU time this session.
I22	Amount of available space (bytes).
I23	Number of terminals currently connected.
I24	Time at beginning of this session.
I25	Date.
I26	First element of the vector I27.
I27	Statement numbers in the state indicator.

FOOTNOTES

1. The APL/B5500 manual (1) is not clear about this. The commands seem to indicate that the user must know both the B5500 usercode and the library prefix. If this is true, it requires revealing B5500 account numbers in order to share libraries; however, a user would have to know both the B5500 account number and the APL user code in order to log in and change or delete a library.

2. If {library number} is not specified, the user code used in the login is assumed.

3. "Library name" is B5500 account number followed by ",", followed by library suffix.

REFERENCES

1. Kildall, G., Smith, L., Swedine, S., and Zosel, M. APL/B5500 Manual, University of Washington Computer Center, Seattle, 1971.

2. Falkoff, A. D. and Iverson, K. E. APL\360: User's Manual, IBM Corporation, New York, 1968.

3. Berry, P. C. APL\360 Primer, IBM Corporation, New York, 1968.