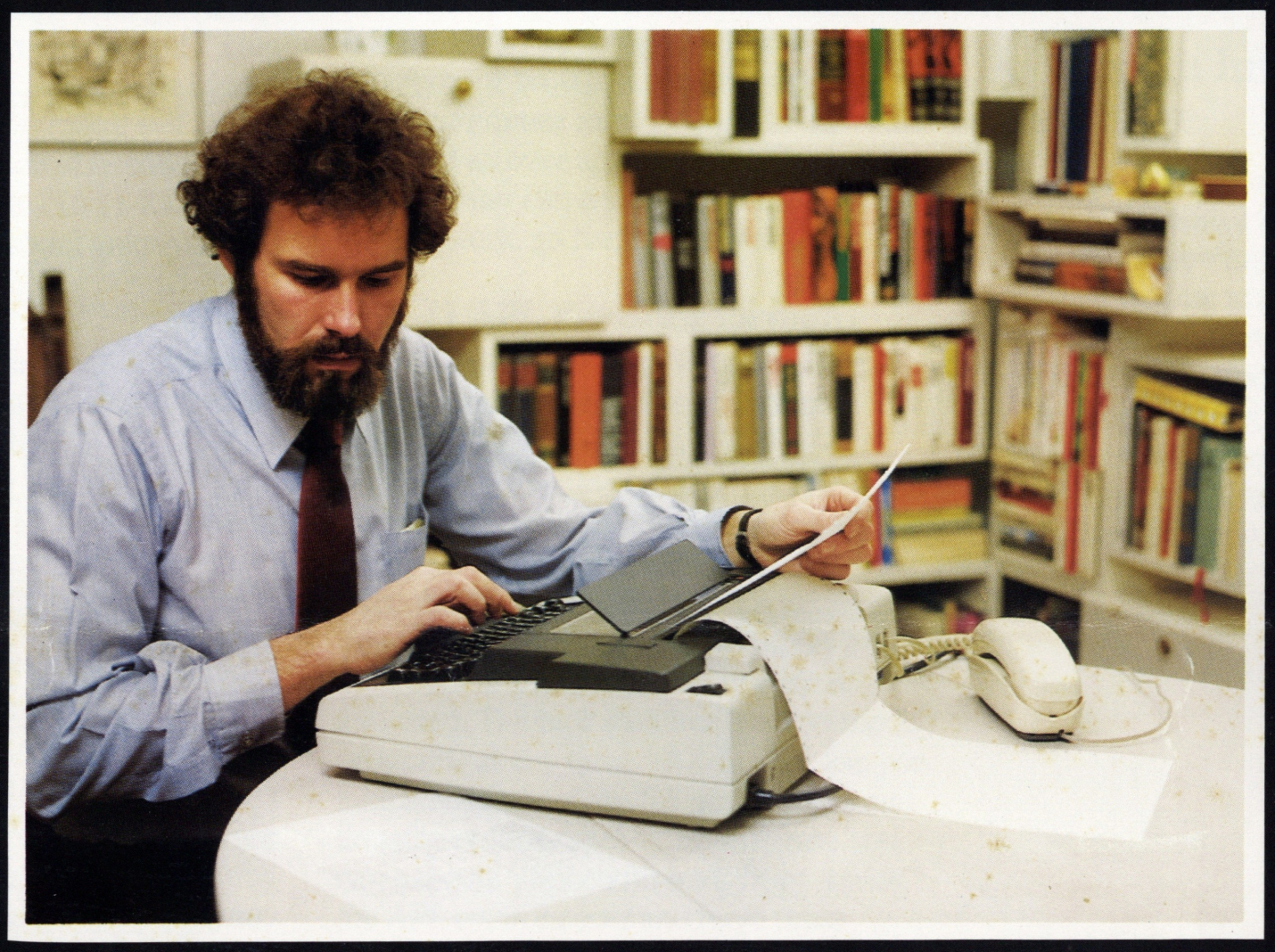

VAX11 APL

digital

Studies indicate that APL programmers *typically* achieve 10 times the productivity of programmers who use lower level-languages. APL is a more productive language because it was *designed to be used by people.* It forces the computer to understand people, not the other way around. With VAX-11 APL, the programmer tells the computer *what* to do, not *how* to do it.

It Takes Most Programmers Less Time To Write Most Programs In APL

VAX-11 APL frees programmers from tedious and time-consuming housekeeping functions that keep the computer happy but contribute nothing towards the solution of the problem. With some languages, programmers seem to spend most of their time getting around the limitations imposed on them by the language.

VAX-11 APL programmers don't spend valuable time reserving space for variables (DIMENSION, DATA DIVISION, etc.) or specifying data types (REAL, INTEGER, etc.). There are no required input or output format statements. VAX-11 APL uses an interactive interpreter, so the programmer does not need to compile or link programs. Since VAX-11 APL users spend less time on housekeeping functions, they have more time to spend solving problems.

Reduce Complex Functions To A Single Line Of Code.

VAX-11 APL has exceptionally powerful primitive functions which greatly simplify many programs. One APL special character can often perform a function that would require a paragraph of code in another language.

For example, APL has two sort characters, one for an ascending sort and one for a descending sort. When you want to sort a table or list, you simply type the name of the table or list, the sort character, and the item or field you want to sort on. Even a complex multikey sort requires only one short line of code.

VAX-11 APL Works At The Speed of Thought

APL uses special symbols to represent mathematical and logical functions that every programmer understands, but which can be difficult to express in words. Once programmers get used to these symbols (it rarely takes more than a few days) they will find that they now have the tools they need to write simple, short, efficient programs. The result will often be an immediate jump in productivity. You can expect to see a steady increase in productivity for some time to come.

Tables And Matrices Are As Easy To Work With As Single Numbers

APL has a number of special characters which let programmers handle lists and tables as easily as they now handle numbers. In most languages, a single character, called the DIVISION SIGN, lets you divide one number by another. In APL, another single character, called the DOMINO, lets you divide one *matrix* by another.

The RESHAPE function, represented by a single character, lets the programmer define the size and shape of a table or even a multidimensional array. This same single character can be applied to the table to determine how many elements are in the table. The programmer, using the RESHAPE function, can change the shape of the table or array *at any time* by typing one short line of code. Another special character, called the CEILING, can be applied to a list or table to find the highest or largest element. In the same way, a special character called the FLOOR lets the programmer find the smallest or lowest element.

VAX-11 APL Lets The Computer Work With You

APL puts the full power of the computer to work *helping* the programmer write programs. The interactive interpreter lets the programmer use the computer dynamically as an *active* programming and debugging tool.

With many languages (even so-called interactive languages with threaded compilers) the programmer first writes a program using an editor. Then the programmer must leave the editor and compile the program. After the program has been compiled, the programmer may enter data and try to run the program. If there are any problems or bugs (a not infrequent event), the programmer must go back to the editor and start all over again. With APL, the programmer can try one solution and if it doesn't work, go back and try another.

VAX-11 APL Provides Virtually Everything A Programmer Needs In A Comprehensive Problem-Solving Environment.

It includes a function editor, debugging aids, system communications facilities and a file system. The interactive interpreter can execute each line of code as it is entered. The extensive debugging aids report immediate results when each line of code is executed. The built-in editor lets you modify programs at any time. VAX-11 APL lets you take advantage of the full power and functionality of VAX/VMS™* commands. You can execute VMS commands *from within the APL environment* and have those commands return the output to APL (or even to a variable within APL). There is no need to leave the VAX-11 APL environment during the creation, testing or execution of programs.

VAX-11 APL Is Easy To Learn

VAX-11 APL is an exceptionally easy language to learn, even for beginners who have never programmed. The user can quickly learn to use a simple, but very useful subset of APL to obtain meaningful results almost immediately. Users will gradually expand their knowledge of APL while continuing to perform highly productive work. Experienced programmers learn a larger subset of APL initially and progress rapidly.

APL Is An International Language

APL is an international programming language because it is not based on English verbs or function names. The special APL characters follow in the tradition of mathematical symbols that transcend lingual boundaries. The translation of English-based programming languages can be as expensive as it is frustrating. Any APL programmer anywhere in the world can read your APL code.

*VAX/VMS is a trademark of Digital Equipment Corporation.

How Much Can You Save Using VAX-11 APL?

We all know that programming time is expensive. Do you know how much it costs you for each hour of a programmer's time?

Let's look at a simple program that not only tells you what it costs you for each hour of programming, but also demonstrates how much you can save by using VAX-11 APL instead of COBOL or FORTRAN.

Suppose you give your programmers two weeks of vacation each year and they work a forty-hour work week. To find their hourly rates you could divide their annual salary by fifty (for the number of work weeks in the year) and then again by forty (for the number of work hours in a week). You could then average each programmer's hourly rate to get an average hourly programming cost.

It sounds like an easy problem to solve, and it is, *in VAX-11 APL*. We asked three experienced programmers to write and debug a program to solve this problem.

To find out how much you would have saved by using VAX-11 APL, run this program (in the programming language of your choice). Of course, your programmers may be either faster or slower than ours, and *some* applications might take longer to write in APL. However, this example is consistent with a number of reported productivity comparisons between APL and other programming languages.

VAX-11 APL

```
     ▽LEN
[1]   STAFF←0 50⍴' '
[2]   SALARY←0⍴0
[3]  LOOP: 'ENTER NAME (TYPE (CR) ONLY WHEN DONE):'
[4]    →(∧/' '=NEWNAME←1 50⍴50↑⍞)/CALCULATE
[5]   STAFF←STAFF,[1]NEWNAME
[6]    'ENTER ANNUAL SALARY WITH NO COMMAS:'
[7]   SALARY←SALARY,1↑⎕FI⍞
[8]   →LOOP
[9]   CALCULATE:TOTAL←(+/HOURLY)÷ROWS←⍴,HOURLY←(SALARY÷50)÷40
[10]  '     ANNUAL      HOURLY'
[11]  '     SALARY      SALARY NAME'
[12]  '---------- ---------- --------------------'
[13]  ''
[14]  (10 2⍕(ROWS,1)⍴SALARY),' ',(10 2⍕(ROWS,1)⍴HOURLY),' ',STAFF
[15]  '           ----------'
[16]  'AVERAGE    ',10 2⍕TOTAL
     ▽
```

It took an APL programmer about ½ hour to write this program.

FORTRAN

```
      CHARACTER*50 NAME
      REAL         SALARY, TOTAL
      INTEGER      COUNT

      OPEN (UNIT=1, FILE='LEN.DAT', ACCESS='SEQUENTIAL',
     1     STATUS='NEW', FORM='UNFORMATTED')

      COUNT = 0
10    WRITE (*,1000)
1000  FORMAT ('$', '''name'', yearly salary [/ to stop]: ')
      NAME = ' '                        ! input / => IO list unchanged
      READ (*,*) NAME, SALARY
      IF (NAME .NE. ' ') THEN
          COUNT = COUNT + 1
          WRITE (1) NAME, SALARY
          GO TO 10
          ENDIF

      IF (COUNT .EQ. 0) THEN
          WRITE (*,*) 'no data was inputted - so none to print'
          CLOSE (UNIT=1, DISPOSE='DELETE')
          STOP
          ENDIF

      REWIND (UNIT=1)

      WRITE (*,1001)
1001  FORMAT (' programmer', T50, 'yearly', T70, 'average', /
     1        ' name',       T50, 'salary', T70, 'hourly',  /
     2                              T70, 'salary',  // )

      TOTAL = 0
      COUNT = 0

20    READ (1, END=100) NAME, SALARY
      COUNT = COUNT + 1
      TOTAL = TOTAL + SALARY
      WRITE (*,1002) NAME, SALARY, (SALARY/50)/40
1002  FORMAT (' ', A50, T50, F12.2, T70, F8.2)
      GO TO 20

100   CLOSE (UNIT=1, DISPOSE='DELETE')
      WRITE (*,1003) ((TOTAL/COUNT)/50)/40
1003  FORMAT (' ', //, ' average hourly salary for all is ', F8.2)

      END
```

It took an experienced FORTRAN programmer over 2 hours to write this program.

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    LEN.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
        CONSOLE IS DISPLAY-DEV.
DATA DIVISION.

WORKING-STORAGE SECTION.
77 HRS-PER-WK                   PIC S9(4) COMP VALUE 40.
77 WKS-PER-YR                   PIC S9(4) COMP VALUE 50.
77 INDX                         PIC S9(4) COMP.
01 NAME-MAX                     PIC S9(4) COMP VALUE 50.
01 NAME-COUNT                   PIC S9(4) COMP VALUE 0.
01 TOT-SALARY                   PIC S9(7)V99 COMP .
01 TEMP-1                       PIC S9(7)V99 COMP .
01 TEMP-2                       PIC S9(7)V99 COMP .
01 AVG-HOURLY                   PIC S9(7)V99 COMP.
01 NEW-SALARY                   DISPLAY.
        05 INTEGER-PART         PIC 9(7).
        05 DECIMAL-PART         PIC 99.
01 WORK-SALARY                  PIC 9(7)V99.
01 IN-SALARY                    PIC X(10) DISPLAY.
01 IN-NAME                      PIC X(40) DISPLAY.
        88 ALL-NAMES-ENTERED    VALUE "EXIT".
01 P-TABLE.
        05 P-NAME               PIC X(40) DISPLAY OCCURS 50.
        05 P-SALARY             PIC S9(7)V99 COMP OCCURS 50.
01 DETAIL-LINE                  DISPLAY.
        05 DETAIL-SALARY        PIC $$$$$$9.99.
        05 FILLER               PIC X(5)  VALUE SPACES.
        05 DETAIL-HOURLY        PIC $$$$9.99.
        05 FILLER               PIC X(6) VALUE SPACES.
        05 PROGRAMMER-NAME      PIC X(40) VALUE SPACES.

01 TOTAL-LINE                   DISPLAY.
        05 FILLER               PIC X(14) VALUE
        "  AVERAGE    ".
        05 DETAIL-AVG-SALARY    PIC $$$$$9.99.
01 HDR-1                        PIC X(75) DISPLAY VALUE
        "  ANNUAL          HOURLY ".
01 HDR-2                        PIC X(75) DISPLAY VALUE
        "  SALARY          SALARY      NAME".
01 HDR-3                        PIC X(75) DISPLAY VALUE
        "  ------          ------      ----".
01 HDR-4                        PIC X(75) DISPLAY VALUE
        "                      ------ ".
PROCEDURE DIVISION.
MAIN-BODY.
        PERFORM INIT-RTN.
        PERFORM INPUT-RTN THRU INPUT-EXIT UNTIL ALL-NAMES-ENTERED.
        PERFORM HEADING-RTN.
        PERFORM COMPUTE-AND-PRINT VARYING INDX FROM 1 BY 1
                UNTIL INDX > NAME-COUNT.
        PERFORM CLEAN-UP.
        STOP RUN.

INIT-RTN.
        MOVE 0 TO TOT-SALARY, TEMP-1,TEMP-2, NAME-COUNT, AVG-HOURLY.
        MOVE SPACES TO IN-NAME.

INPUT-RTN.
        DISPLAY "ENTER NAME (OR EXIT TO EXIT): " UPON DISPLAY-DEV
                WITH NO ADVANCING.
        ACCEPT IN-NAME FROM DISPLAY-DEV.
        IF (ALL-NAMES-ENTERED OR NAME-COUNT = NAME-MAX)
           GO TO INPUT-EXIT
        ELSE DISPLAY "YEARLY SALARY : " UPON DISPLAY-DEV
                WITH NO ADVANCING
                ACCEPT IN-SALARY FROM DISPLAY-DEV
                        UNSTRING IN-SALARY DELIMITED BY "." OR " "
                           INTO INTEGER-PART,DECIMAL-PART
                COMPUTE WORK-SALARY = (DECIMAL-PART * .01) +
INTEGER-PART
                        ADD 1 TO NAME-COUNT
                        MOVE IN-NAME TO P-NAME(NAME-COUNT)
                        MOVE WORK-SALARY TO P-SALARY(NAME-COUNT).
INPUT-EXIT. EXIT.

COMPUTE-AND-PRINT.
        MOVE P-NAME(INDX) TO PROGRAMMER-NAME.
        MOVE P-SALARY(INDX) TO DETAIL-SALARY.
        ADD P-SALARY(INDX) TO TOT-SALARY.
        DIVIDE P-SALARY(INDX) BY WKS-PER-YR GIVING TEMP-1.
        DIVIDE TEMP-1 BY HRS-PER-WK GIVING AVG-HOURLY.
        MOVE AVG-HOURLY TO DETAIL-HOURLY.
        DISPLAY DETAIL-LINE UPON DISPLAY-DEV.

HEADING-RTN.
        DISPLAY SPACES UPON DISPLAY-DEV.
        DISPLAY HDR-1 UPON DISPLAY-DEV.
        DISPLAY HDR-2 UPON DISPLAY-DEV.
        DISPLAY HDR-3 UPON DISPLAY-DEV.

CLEAN-UP.
        DISPLAY HDR-4 UPON DISPLAY-DEV.
        DIVIDE TOT-SALARY BY WKS-PER-YR GIVING TEMP-1.
        DIVIDE TEMP-1 BY HRS-PER-WK GIVING TEMP-2.
        DIVIDE TEMP-2 BY NAME-COUNT GIVING TOT-SALARY.
        MOVE TOT-SALARY TO DETAIL-AVG-SALARY.
        DISPLAY TOTAL-LINE UPON DISPLAY-DEV.
```

It took an experienced COBOL programmer over 4 hours to write this simple program.

VAX-11 APL
The Productivity Of APL. . . The Power Of VAX Together They're "Virtually" Unbeatable

VAX-11 APL is a native-mode interpreter that provides virtually everything you need during a terminal session.
VAX-11 APL features include:
- Built-in function editor
- Debugging Aids
- System communications facilities
- File management system

All The Workspace You'll Ever Need

VAX-11 APL gives you a "virtually unlimited" workspace by taking advantage of the VAX/VMS Virtual Memory System. Your workspace can expand as you need it. You don't have to worry about wasting system resources. The VAX/VMS operating system allocates available resources when you need them.

The symbol table and function stack (SI stack) also expand dynamically according to need. The expandable workspace speeds processing by reducing file I/O.
- The ultimate size of your workspace is limited only by the available virtual memory.
- The workspace expands dynamically according to need.
- The symbol table and SI stack expand dynamically according to need.
- You can limit the size of your workspace.
- The VAX/VMS system manager can limit the amount of virtual memory available to any user.

Flexible Error Handling

VAX-11 APL gives you the flexibility you need for application-oriented error handling in user-defined functions. You can define error handling routines that will:
- Detect, analyze, and correct errors without interrupting the program.
- Halt execution of the program, prompt the user to enter a correction, then resume execution.
- Terminate the execution of a program when the severity of an error requires it. The user can resume executing the program from that point once the error is corrected.
- Display either the appropriate VAX-11 APL error message or an error message you write for that particular application.

□TRAP Executes a stored APL expression when an error is detected.

□ERROR Contains the error-message text of the last error detected.

□SIGNAL Lets you define new error situations unique to the application.

□BREAK Immediately stops program execution and returns control to the user.

Meaningful Error Messages

VAX-11 APL gives you meaningful error messages that pinpoint the location and the nature of a detected error. For example, if you have a domain error that was caused by an attempted division by zero, VAX-11 APL displays the line where the error occurred and the message, "DOMAIN ERROR (DIVISION BY ZERO)." In the same way, should you ever manage to use all of your available virtual memory, you will see, "WORKSPACE FULL (VIRTUAL MEMORY EXHAUSTED)."

Debugging Tools

VAX-11 APL supplies several features that help you find logic errors in your programs. These features include:
- More detailed error messages that exactly pinpoint the source and nature of a problem.

□TRACE Lets you see the results of each requested line of APL as it executes without changing the source.

□STOP Sets breakpoints on lines of APL without changing the source. You can examine variables and the VAX-11 APL environment. You can continue programs right from where they were stopped.

□BREAK You can use inside a VAX-11 APL program to set breakpoints.

File Compatibility

VAX-11 APL applications can access the same files used by applications written in other VAX/VMS languages, including VAX-11 FORTRAN, VAX-11 BASIC, VAX-11 PASCAL, VAX-11 PL/1, VAX-11 COBOL, and VAX-11 C. Many users can read the same file at the same time. It does not matter which language created or last wrote that file. This is possible because VAX-11 APL uses VAX-11 RMS (Record Management System) for all file input and output.

VAX-11 APL Supports Three File Types:
- Sequential files
- Relative files
- APL component files

Workspaces Are Saved As VMS Files

When you use the **)SAVE** command, your entire VAX-11 APL workspace is stored as a VMS file and is listed in your directory. You can use the full power of VAX/VMS to manage workspace files.
- Assign protection to workspace files.
- Copy, rename, or delete workspace files.
- Send workspace files to other VAX/VMS systems using DECnet.™

*DECnet is a trademark of Digital Equipment Corporation.

Execute Any VMS Command From Within APL

VAX-11 APL lets you execute any VMS command from within the APL environment—even from within an APL program.

The **)DO** command lets you execute a single VMS command line from within APL. Optionally, the results of this command can be stored in an APL variable.

For instance, you could use the **)DO** command to determine who is using the system. You can write a program that sends a message to another terminal when that program has completed processing.

The **)PUSH** command places you in a VMS subprocess without disturbing your APL environment. While you are in this subprocess you can use all of the VAX/VMS features and commands. When you are done with the subprocess, you are returned to APL exactly where you left off.

For example, you could use **)PUSH** when you want to use the VMS Mail utility. You can read and send mail, then return to your APL session.

Batch Processing

VAX-11 APL accepts commands from a file as well as from a terminal. This allows for batch-like processing within the APL environment. You can use the VAX/VMS Batch Control System to run APL programs with no modification to the APL workspaces or user input conventions.

APL Command Files

Command files help you write applications for people who are not familiar with APL. You can store all the information you need to start a session and load a workspace in a VMS command file. This workspace can automatically start executing by placing an expression in the variable **⎕LX.**

APL Initialization Files

Frequent APL programmers can store the APL environment they like in an initialization file. This automatically initializes APL each time it is run. For example, if you use the same terminal every time you use VAX-11 APL, you can store your terminal type in the initialization file.

Sharing Data

VMS mailboxes and event flags provide an easy way for a running VAX-11 APL process to pass information and data to other running processes. The VAX-11 APL process can then wait for the other process to use the information and pass back the results. The other processes can be written in VAX-11 APL or any VAX language that supports mailboxes. These include VAX-11 FORTRAN, VAX-11 BASIC, VAX-11 COBOL, and VAX-11 C.

Writing Output

Many APL products require extensive user-written file operations to create a report or any text that would eventually go to a lineprinter or system editor. VAX-11 APL has features that automatically write sequential text files in the supported character set of the user's choice.

When invoked, **)OUTPUT** directs everything that would have appeared on a terminal to a file. When invoked, **)OUTPUT /SHADOW** directs output to both the terminal and to a file. The file looks the same as the output would look on the terminal, complete with formatting and control characters.

APL With Powerful Extensions

VAX-11 APL includes the APL language elements most programmers would expect. It has every primitive function and operator described in *Development of an APL Standard,* by Falkoff and Orth, in the APL79 Conference proceedings. The interpreter has an extensive set of DIGITAL language extensions.

VAX-11 APL Extensions:

Execute Function
Allows any APL character string to be evaluated just as if it were entered from a terminal. Therefore, a variable can contain the name of another variable for analysis. Any APL system command may be executed from within an APL program.

Diamond Statement Separators
Allow more than one APL expression on a single line.

Ambivalent Functions
Allow the user to define programs that will accept either one or two arguments. The program can test to see how many arguments it has.

Replicate
Adds new capabilities to Compression. Not only can elements be eliminated, but the same expression can insert spaces or zeros and can repeat any element as many times as the user wishes.

Scalar Extension
Extended to vectors and matrices. A single element object of any shape will be expanded just as if it were a scalar.

End-Of-Line Comments
Make it easier to document APL code.

Sorts
Can be made on characters as well as numbers.

User Defined FMT Functions
Add formatting capabilities beyond the Format function.

APL From Any ASCII Terminal

A user can access every part of the VAX-11 APL interpreter without a special terminal because VAX-11 APL has a full complement of mnemonic names for each of the APL characters. For graphic applications, VAX-11 APL can transmit any character, including escape sequences. VAX-11 APL also supports the full APL character set for users with APL terminals.

Transferring Workspaces

VAX-11 APL fully supports the Workspace Interchange Convention which enables the user to exchange workspaces with APL products running on other systems, including the fully-featured APL products on the DECsystem-10 and DECSYSTEM-20 computers.

APL System Variables

⎕AUS	Automatically backs up the current workspace periodically.
⎕CT	Determines the degree of tolerance applied in numeric comparisons ("fudge factor").
⎕DML	Sets the maximum record length used to save the workspace or to create a file.
⎕ERROR	Contains the text of the error message for the last error detected.
⎕GAG	Indicates whether to accept messages sent from other users.
⎕IO	Sets index origin for arrays.
⎕LX	Causes expression to be executed automatically when the workspace is loaded.
⎕NG	Controls the recognition and printing of the negative sign (with standard ASCII terminals).
⎕PP	Controls the number of significant digits in noninteger output.
⎕PW	Sets the page width.
⎕RL	Forms the seed for random numbers.
⎕SINK	Discards unwanted output. Always **iota** 0.
⎕SF	Prompt for evaluated input. Can be changed from ⎕:
⎕TERSE	Suppresses display of secondary error messages.
⎕TRAP	Causes expression to be executed when error occurs in a user defined function.

APL System Functions

⎕AI	Maintains account information on the current APL session. Includes user identification, CPU time and connect time.
⎕ALPHA	Vector of 27 characters: Δ and A through Z.
⎕ALPHAU	Vector of 27 underscored characters.
⎕ARBOUT	Writes arbitrary output to the terminal.
⎕ASS	Associates a file or mailbox with a channel.
⎕AV	Vector of all APL characters.
⎕BREAK	Suspends function execution and returns control to immediate mode.
⎕CHANS	Identifies channel numbers associated with files.
⎕CHS	Returns file organization and open status.
⎕CIQ	Unpacks data packed by **⎕COQ**.
⎕CLS	Closes the files on one or more channels.
⎕COQ	Packs data of different types for storage as one record.
⎕CR	Returns a canonical representation of a defined function.
⎕CTRL	Vector of ASCII control characters.
⎕DAS	Disassociates files from one or more channels.
⎕DL	Delays execution by the number of seconds specified.
⎕DVC	Returns the device characteristics for one or more files.
⎕EFC	Clears event flags associated with one or more channels.
⎕EFR	Returns the setting for event flags associated with one or more channels.
⎕EFS	Sets event flags associated with one or more channels.
⎕EX	Expunges existing use of a name in the workspace.
⎕FLS	Returns information about files on one or more channels.
⎕FI	Converts character argument to numeric, placing 0s in each position not corresponding to a valid number.
⎕FX	Establishes a function from its canonical representation.
⎕LC	Vector of line numbers in state indicator. Most recently suspended function appears first.
⎕MBX	Returns information about mailboxes on one or more channels.
⎕NC	Returns the classification of one or more names.
⎕NL	Constructs a list of named objects residing in the active workspace.
⎕NUM	Vector of 10 digits: 0 through 9.
⎕QCO	Quietly copies a workspace.
⎕QLD	Quietly loads a workspace.
⎕QPC	Quietly copies a workspace with certain protection.
⎕RELEASE	Releases all locked records in files on one or more channels.
⎕RESET	Clears the state indicator.
⎕SIGNAL	Passes an error up the stack one level to the caller of the function in error.
⎕STOP	Sets or clears stop bits associated with function lines.
⎕TRACE	Sets or clears trace bits associated with function lines.
⎕TS	Current date and time in base 10 format.
⎕TT	Terminal type for current APL session.
⎕UL	Process identification number.
⎕VERSION	Interpreter and workspace versions.
⎕VI	Returns logical vector giving position of valid numbers in **⎕FI** of argument.
⎕WA	Maximum amount in bytes by which the active workspace can be increased.

Order Code: EA-23365-18