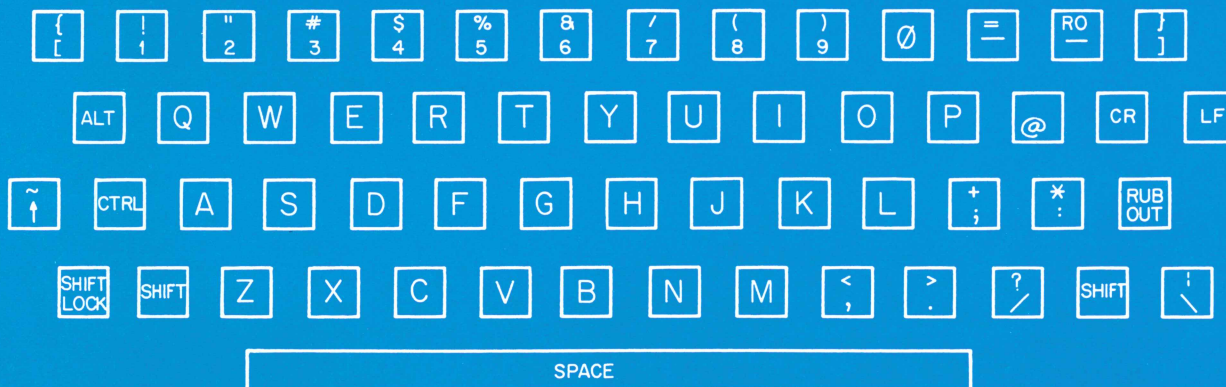


decsystem10

APL Symbolic Programming Language



APL Keyboard



LA30 Keyboard

APL SYMBOLIC PROGRAMMING LANGUAGE

APL is a concise programming language especially suitable for dealing with numeric and character array-structured data (a generalized extension over the more typical scalar operations found in most other programming languages). APL is rich in operators that facilitate array calculations, thereby enabling the programmer to write programs at a much higher level than he could with more conventional languages. This higher-level programming is accomplished by suppressing much of the programming detail inside single APL operators (for example, the operator Δ may be used to sort a vector of values in ascending order, thereby making 'sort' a primitive operation rather than a tedious subroutine). A great many useful functions required in computing have been defined as primitive (single character) operators in APL.

The accompanying figure illustrates typical APL keyboards. Many of the up-shift positions on the keyboard are occupied by symbols used to represent the powerful set of APL operators. The range of operators is further expanded by including many primitive symbols which consist of two characters overstruck to produce a single operator (for example, overstriking \circ with \backslash creates the APL operator $\circ\backslash$, which is used to take the transpose of an array).

Of course, with such a wealth of generalized operators, imposing an arbitrary order of precedence on the operators would be rather unrealistic in terms of programmability. Therefore, in keeping with APL's standard of uniformity and brevity, a single rule was chosen for evaluating all unparenthesized expressions:

Every operator takes as its right-hand argument the value of everything to its right (up to the right parenthesis of the pair that encloses it).

That is, evaluation is essentially from right-to-left. Thus, a nested polynomial may be written without parenthesis, as:

$$1 + Y \times 2 + Y \times 3 + Y \times 4$$

as compared to the Fortran equivalent:

$$((4 * y + 3) * y + 2) * y + 1$$

Despite its mathematically concise and consistent format, APL is intended to be used as a general data-processing language as well as a mathematician's tool. APL is flexible enough to solve problems in text-handling and commercial data-processing as concisely and easily as it can be used to solve problems in numerical mathematics and statistics. Furthermore, APL is a completely conversational system, which tends to increase programmer productivity and expertise by allowing the user to interact with the APL system and his running programs. APL also provides program clarity as an important by-product of its extensive operator set, since a single operator often provides an entire logical function which would require many more program steps in any other programming language. What APL has done is to move in the direction of a library, increasing the sophistication of the language while at the same time simplifying the notation by using an extended set of operators.

As a simple example of the concise nature of APL, consider the following function which calculates the average of a vector V of numeric values:

2741 Type	ASCII Standard
$\nabla R \leftarrow AVERAGE\ V$. DL R \leftarrow AVERAGE V
[1] $R \leftarrow (+/V) \div \rho V$	[1] R \leftarrow (+/V) \div @RV
∇	. DL

The operator ρ (or $@R$) is used to determine the length of the vector V (that is, the number of elements to be averaged); and the sum reduction ($+ /$) is used to sum the elements of the vector V (notice how APL conceals such irrelevant operations as the sequencing control by which the vector elements are accumulated). The resultant value is returned via the result variable R .

A sample calling sequence for this function might be:

```
AVERAGE 8 6 7 9
7.5
VECTOR  $\leftarrow$  81 73 77 89 100
ANSWER  $\leftarrow$  AVERAGE VECTOR
ANSWER
84
```

A few of the widespread areas of current usage of APL include numerical mathematics, statistics, actuarial mathematics, scientific data-reduction and analysis, simulation and forecasting, financial modeling, design engineering, engineering analysis, electrical-circuit analysis, inventory and payroll management, data-base manipulation, reservation systems, text handling and symbol manipulation, mechanical theorem proving, computer-assisted instruction (CAI), and student education (high-school and college level) in programming and the structure of algorithmic processes. The applicability of APL as a complete conversational programming system is unlimited.

DECsystem-10 APL SYSTEMS AND FEATURES

Two APL systems exist for the DECsystem-10 computer: a Basic System and an Extended System.

The Basic APL System is nearly identical to the APL system described in the APL/360 Reference Manual (Sandra Pakin, Science Research Associates, 1972). Essentially, no changes have been made to the APL language except to extend its capabilities. Some of the more significant extensions that are available in the Basic System include:

1. The user's active workspace size is dynamically variable, so APL automatically uses only as much core storage as is necessary at any given time. The maximum possible workspace is 128K words.
2. The workspace symbol-table size is variable, thereby allowing the user to allocate only as much table space as is actually required for a given workspace.
3. Two terminal-input/output systems are supported:
 - a. APL Terminals that accept a standard APL type-ball printing element or otherwise produce the APL character set (IBM 2741-type terminals). 2741 support is available as an option to the DC10 data-line scanner.

- b. Standard ASCII hard copy or display terminals, such as Teletype 33/35, VT05, LA30, etc. These terminals do not produce the APL character set; hence, a set of alternate input representations for the special APL characters is available (such as .RO or @R for ρ).
- 4. Immediate-mode line editing is provided, whereby the user may edit the last line he typed via the same mechanisms used for function definition-mode line editing. Thus, complex APL expressions may be entered, executed, corrected, and re-executed without having to re-type the entire line again.
- 5. The scan operator (\) is introduced to complement the existing reduction operator (/).
- 6. Statement branching (→) may occur anywhere in a statement line. Multiple statements may appear in a single line separated by semi-colons.
- 7. The) *TABS* command is used to inform the APL system of user-set tab stops to facilitate both input and output operations. Tabs will be inserted whenever possible on output, greatly increasing the speed of graphic and columnar output.
- 8. System command formats have been expanded to take advantage of standard DECsystem-10 file conventions. Users may save workspaces on disk, DECtape, and magnetic tape.

Also available is an Extended APL System that combines all the capabilities of the Basic System with the following major extensions:

- 1. The divide-quad operator (⊘) is implemented, providing a single operator to solve a set of linear equations, take the inverse of a matrix, or solve an overdetermined set of linear equations using a least-squares fit.
- 2. The execute operator (monadic ⊥ or ε) allows any APL character string to be evaluated as if it were typed in from the terminal in immediate-mode. There is no restriction on the contents of the character string to be evaluated; in particular, the string may contain system commands, function definitions, or several statement lines delimited by carriage-returns. The execute operator allows the user the power to construct total subsystems within APL, including the ability to handle error analysis and recovery.
- 3. The quote operator (monadic τ) provides the ability to convert numeric data to a character string and to convert a function definition to a corresponding character string (the lines of the function being separated by carriage-returns). The quote operator gives the user the capability to write user-defined functions to perform special output formatting and function editing.
- 4. The dyadic format operator (⍎) gives the user a convenient mechanism for formatting his output data.
- 5. A file system provides the user easy access to any of three file formats:
 - a. Standard ASCII sequential files,
 - b. Internal format random-access files, and
 - c. Internal format sequential files.

As examples of the use of the file system, consider the following:

- a. A user-defined APL function may request its data input from a standard ASCII sequential file, which had been created previously by the output from a Fortran program (or another APL program).
- b. An APL program can store its computation results (intermediate results from a tape-merging process, for example) on an internal sequential file for use by another APL function. Since no format conversion is necessary, there is practically no overhead in doing I/O to internal sequential files.
- c. An airline reservation system may be implemented as a set of APL programs that operate on a common data base constructed as an internal format random-access file (APL also has provisions to allow several users to concurrently share the same data file).

Operations involving the use of the APL file system are invoked by using two new operators: ⊞ for input and ⊞ for output, with suitable subscripts and arguments. For example, $A \leftarrow \boxplus 2$ may be used to input the next element of the file associated with logical channel number 2; and $(2 \ 4 \ \rho \ 1 \ 8) \boxminus 6$ may be used to output the indicated matrix along channel number 6.

A further capability of the file system is the ability to handle immediate-mode I/O through devices other than the user's terminal. For example, output that would normally be printed at the terminal can be diverted to the line printer by first executing the system command) *OUTPUT LPT* : (or divert it to an ASCII disk file by) *OUTPUT AFILE.DAT*).

THE OPERATING ENVIRONMENT: MACHINE REQUIREMENTS

All the code of both APL systems is completely sharable and resides in the DECsystem-10 high segment. The high segment size is 20K words (36-bit words) for the Basic System and 24K words for the Extended System. Each user's low segment contains about 1500 words of system variables plus the user's workspace area (which is dynamically varying in size). User workspaces are typically 5 or 6K words in size.

The hardware configuration required to run APL is minimal. Approximately 29K words of permanent disk or drum storage space are required for the Basic System to reside in secondary storage (about 33K for the Extended System). No permanent user disk space is really required, since APL allows workspaces to be saved on DECtape and magnetic tape.

APL-supporting full-duplex ASCII terminals (e.g., Teletypes and ASCII APL terminals) will run under a standard DECsystem-10 timesharing monitor.

APL is a product of the DECsystem-10 Advanced Systems Group.

Prices for APL, including 2741 support, are available from your local DIGITAL Sales Office.

