10/14

ANL-PHY-1968E

Argonne Physics Division
Informal Report PHY-1968E
December 1968
(Revised May 1971)

AN INTRODUCTION TO SPEAKEASY

by

S. Cohen and C. M. Vincent
Argonne National Laboratory, Argonne, Illinois 60439

DISCLAIMER

The facilities of Argonne National Laboratory are owned by the United States Government. Under the terms of a contract (W-31-109-Eng-38) between the U. S. Atomic Energy Commission, Argonne Universities Association and The University of Chicago, the University employs the staff and operates the Laboratory in accordance with policies and programs formulated, approved and reviewed by the Association.

MEMBERS OF ARGONNE UNIVERSITIES ASSOCIATION

The University of Arizona
Carnegie-Mellon University
Case Western Reserve University
The University of Chicago
University of Cincinnati
Illinois Institute of Technology
University of Illinois
Indiana University
Iowa State University
The University of Iowa
Kansas State University
The University of Kansas
Loyola University
Marquette University
Michigan State University
The University of Michigan
University of Minnesota
University of Missouri
Northwestern University
University of Notre Dame
The Ohio State University
Ohio University
The Pennsylvania State University
Purdue University
Saint Louis University
Southern Illinois University
The University of Texas at Austin
Washington University
Wayne State University
The University of Wisconsin

Argonne Physics Division
Informal Report PHY-1968E
December 1968
(Revised May 1971)

AN INTRODUCTION TO SPEAKEASY*

by

S. Cohen and C. M. Vincent

Argonne National Laboratory, Argonne, Illinois 60439

The authors have a limited number of copies for general distribution. Anyone who desires a copy should contact one of them directly; the PHY Informal Reports are not handled through the ANL Technical Publications Department.

*Work performed under the auspices of the U. S. Atomic Energy Commission.

Table of Contents

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

Abstract

SPEAKEASY is a computer language designed for scientists. This document is primarily an introduction to that language, but it also serves as a reference manual for the version currently available for IBM-360 series computers. The specific control cards needed to run jobs at the Argonne installation are described.

The report contains two major sections. The first is a detailed description of the basic language. The last section of the report is an illustration of the use of the language in the form of an actual computer run.

I. INTRODUCTION

SPEAKEASY is a user-oriented language. It is intended to provide scientists with the means of quickly formulating a problem for computer processing and for obtaining answers to their problems in a minimum of time. The language is easily learned since its form is similar to that of scientific mathematics. Furthermore it has built into its basic structure the commonly used operations of most scientific disciplines. The user may freely draw on a very large library of such operations and may thus formulate his problem with a very concise directive program.

In designing SPEAKEASY, every effort has been made to keep the language natural as viewed by the scientist. A scientist using SPEAKEASY need know little about a digital computer and, in particular, need never concern himself with the actual structure of the program as run by the computer. In a sense, SPEAKEASY is to be viewed as a humanized interface between a scientist and a computer; occurences that force the user to think about a digital computer rather than about his scientific problem are to be viewed as failures in the language.

Because of the concise and natural form of the language, it is very likely that a new program will give correct answers on the first trial. If this is not the case, extensive error-detection aids will provide enough information to make the second attempt almost certain to succeed. The user will therefore obtain an answer to his problem in a very short time.

This document is a description of the SPEAKEASY programing language. The description is not intended to be exact in every detail. To attempt an exact description would so burden the reader with details that he would lose sight of the basic simplicity of the language.

Neither is any attempt made to describe the entire language. The language is still being developed and new features are added as needs arise. The description here should be viewed as an introduction to the basic language.

Detailed descriptions of parts of the language are given in the sections that follow. The second chapter describes the basic language and shows how this subset of the language can be used in the so-called manual mode. The third chapter describes the use of stored programs. The last chapters describe the diagnostic features of the language. Each chapter contains numerous examples of the features being described. It is felt that examples provide the best means of demonstrating the capability of the language. It is hoped that the new user will look carefully at a few of the major examples and attempt to understand the rather fundamental differences between this language and other computer languages.

Appendix I is a summary of all of the keywords used in the language. Several alternative arrangements are given.

Appendix II contains a description of the conventions used in the card-input version of this language. It also contains samples of the Job Control Language cards necessary to execute a SPEAKEASY program. Several alternative sets are described.

Appendix III is the collection of the schedules of this report. Since this writeup is intended to serve as a reference manual, it was felt that the schedules should be arranged for quick access. A list of the titles of schedules is given at the beginning of this appendix.

II. BASIC SPEAKEASY (The Manual Mode)

In this chapter we describe the basic notation and conventions of the SPEAKEASY language. The concepts of structured objects and the algebra for these objects is discussed in detail. The built-in functions of the language are then enumerated. In the final section of this chapter, examples of the use of this part of the language for realistic calculation are given.

A. BASIC NOTIONS

1) Data

All numeric data are specified in decimal form. The lack of a decimal point in a numeric specification implies that it belongs just to the right of the number. Numbers can be real, imaginary, or complex. A terminal letter I implies that the preceding number was imaginary. Examples of numeric information are:

Real Data:	2.14, 27, -.0025
Imaginary Data:	2.14I, 27I, -1I
Complex Data:	6 + 2I, 4 - 6I, 2.7 + 3I

Very large and very small numbers can be expressed by terminating the numeric field by the letter E followed by the power of 10 associated with that number. (No spaces should be inserted anywhere in the numeric specification.)

1.057E-5	means	1.057×10^{-5}
236E+26	means	2.36×10^{28}
61IE05	means	$61i \times 10^{5}$

Literal data are defined by enclosing the data in apostrophes, e.g.,

'BOOKS'	'15EB7'
'....'	'***'

2) Names of Objects

An object can be given a name consisting of up to eight characters. The name must begin with an alphabetic character and must have no imbedded special characters or blank spaces. Any additional characters beyond the first eight are ignored. Examples of allowed names are:

VALUE BOOKMARK I2 I2K7

The examples given below are not allowed names.

1H27	(not allowed)	non-alphabetic leading character
B*A9	(not allowed)	imbedded special character
A.27	(not allowed)	imbedded special character
A 15	(not allowed)	imbedded space

3) Classes

The objects used in statements may belong to any of several classes. Although most computer languages provide for construction of multi-dimensional arrays, SPEAKEASY also provides the mathematical tools for manipulating the arrays as single entities.

Two families of objects are available in this language: the matrix-vector family (MFAM) or (VFAM) and the array family (AFAM). In addition, scalar quantities can be defined. Scalars are implicitly members of both of the families. The classes of objects that can be defined and used in SPEAKEASY are

a) Scalars — Class [S]

Single numbers

Matrix/vector family

b) Vectors — Class [V]

One-dimensional arrays of numbers that behave like row, column, or diagonal matrices according to context.

c) Matrices — Class [M]

Two-dimensional arrays of numbers that obey the rules of matrix algebra.

Array family

d) Arrays (1 dim.) — Class [A1]

One-dimensional arrays in which operations take place element-by-element.

e) Arrays (2 dim.) — Class [A2]

Two-dimensional array in which operations take place element-by-element.

An object of any class having only a single element is treated as a scalar in all algebraic operations.

4) Elements of an Object

Individual elements of a structured object are referred to by index parameters and the name of the variable is attached to that object. Thus

M(1, 3) is the element in row 1 column 3 of M

V(3) is the third element in V
Similarly if K is a scalar variable
F(K) is the Kth element in F.

Each element of any object is a number and is therefore of Class [S].

B. MEANS OF DEFINING OBJECTS

In SPEAKEASY, each named entity has certain attributes that describe the class to which it belongs and other information relating to its size and structure. In all operations that use this names quantity, these attributes are examined and are used to decide the contextually-implied operation.

The attributes for a given names variable are not fixed quantities and may vary dynamically during a calculation.

Each names item appearing on the right-hand side of an equation must have been assigned attributes — i.e., it must have been defined. Such definitions can be made by use of explicity defining statements. If the item appears on the left side of a previously executed equation, it has an implied structure and is therefore defined. In most cases it is therefore not necessary to specify the class to which a variable belongs; this definition will have been made from the logic of the preceding statements.

In this section the explicit defining statements are described. Little is said about the implicit definitions. It should be understood, however, that definition by the implicit form is far more common than by the more cumbersome explicit forms.

1) Explicit Definitions

This section gives all of the explicit defining statements. In each case they appear as equations in which the names object on the left is being defined. The right-hand side of the equation is an expression that specifies the class and often the numerical values of elements of the object. This form of expression defines a temporary object that can be used directly in more complicated expressions. They can be considered as defining functions.

In every case, a constant appearing on the right can be replaced by an expression or named quantity. It is essential, however, that structure-determining quantities be positive integer scalars. If an object appears in the definition of the elements of a new object, then the elements of the old object are inserted into the new object.

a) Scalars

Scalars are explicitly defined by statements in which the named variable is equated to a constant, e. g.,

X = 1. 57

or

X = 2. 36 + 4. 7I

b) Vectors

A vector is defined by use of the special word VECTOR. SPEAKEASY vectors are objects that behave like row, column, or diagonal matrices—depending on their use. They are particularly useful in operations involving square matrices, where they perform operations commonly involving transpositions and the like. X is defined as a vector of 12 components (all of which are set to zero) by the statement

X = VECTOR (12:)

A vector may be defined with its component values set by the statement

X = VECTOR(:5. 3, 2 + 3. 5I, 7, 26. 3 + 2I)

If the defining statement has only one argument and if this argument is real, then the statement defines the number of components of the vector. In contrast, the second form defines both the number of components and values of the components.

A third form of definition combines the two forms already described. It is of the form

X = VECTOR (4: 1, 2, 2)

The number preceding the colon defines the number of components of the vector and sets all components equal to zero. The list of arguments following the colon sets successive components starting with the first; any unset components are therefore zero.

c) Matrices

Matrices can be defined in a variety of ways to make use of symmetries. The general nonsquare matrix is defined by the statement

M = MATRIX (3, 4:)

This defines M as a matrix with 3 rows and 4 columns. (All elements are set to zero.)

Values of components can be set at the time of definition by placing a colon after the argument and adding values. The equation

M = MATRIX (3, 3: 1, 2, 5, 7, 3, 8, 9)

defines

$$M = \begin{pmatrix} 1 & 2 & 5 \\ 7 & 3 & 8 \\ 9 & 0 & 0 \end{pmatrix} .$$

Symmetric matrices are defined by the expressions*

M = SYMMAT (3, 3:) or M = SYMMAT (3:)

for a rank-3 symmetric matrix. Components supplied with the defining statement give successive elements in the lower diagonal form of the matrix. Thus

M = SYMMAT (3: 1, 2, 3, 4, 5)

defines

$$M = \begin{pmatrix} 1, & 2, & 4 \\ 2, & 3, & 5 \\ 4, & 5, & 0 \end{pmatrix} .$$

---

*Note that symmetric matrices etc. are <u>not</u> classes of objects. The functions given here are just convenient ways of defining square matrices and setting selected components.

Similar definitions exist for diagonal and antisymmetric matrices.

M = DIAGMAT (:1, 2, 3)

or

M = DIAGMAT (3: 1, 2, 3)

defines $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$

M = ASYMMAT (4: -1, -2, -3, 1) defines $\begin{pmatrix} 0 & 1 & 2 & -1 \\ -1 & 0 & 3 & 0 \\ -2 & -3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$

d) Arrays (1 dimension)

Arrays of one dimension are defined by expressions similar to those for vectors.

V = ARRAY (5:)	A 5-component array all zero.
V = ARRAY (:2.7, 3, SQRT(8))	A 3-component array.
V = ARRAY (6: 1, 2, 3)	A 6-component 1-dimensional array with the first 3 components set. The last 3 are zero.

In addition, a one-dimensional array can be defined as an equally-spaced grid of points between specified limits by the special defining statement

X = GRID (0, 10)

i.e., X is

(0, Δ, 2Δ, · · · , 10 - Δ, 10)

where Δ is chosen by the computer to give a preselected number of grid points (normally 101).

Specific grid spacing can be obtained by use of a third argument

X = GRID (0, 10, .05)

i.e., X is defined as

(0, .05, .10, .15, · · · , 9.95, 10.00)

Note that a complex set of grid points can be constructed by a statement such as

X = GRID (0, 10 + 10 I)

e) Arrays (2 dimensions)

A two-dimensional array has a defining statement of the form

V = ARRAY (5, 2: 1, 2, 3)

and its components are set as in the definition of the nonsquare matrix. A resumé of the explicit defining expressions is given in Schedules 1, 2.

2) Implicit Definitions

The appearance of an object on the left-hand side of an equation is an implicit definition of the class of the object. Thus

X = M1 + M2,

where M1 and M2 are matrices, implicitly defines X as a matrix. No explicit definition of X need precede such a statement. Indeed if X were previously defined it would be redefined by this statement.

The appearance of a previously undefined indexed variable on the left of an equation also implicitly defines it as an array. Thus

V(3) = 27.5

implies that V is a one-dimensional array of 3 components if it is not previously defined. The third component is set equal to 27.5.

Similarly,

M(2, 1) = 5

implies that M is a two-dimensional array with at least 2 rows and 1 column. If the quantity is not previously defined, it is defined by this statement with the minimum required number of components. The nondefined components are zero, the element in the second row and first column is 5.

C. MATHEMATICAL OPERATIONS

The notation described in this section will appear at first glance to be identical to that of FORTRAN. The appearance is deceptive. FORTRAN is oriented towards operations on scalar quantities only, and the meaning of statements is restricted to operations that produce a single numerical result. In SPEAKEASY the operations are dependent on the class of objects involved, and the class of the results will be determined by the classes of the objects in the statement.

1) Operators

The following are the allowed operators for arithmetic operations in SPEAKEASY. The symbols are those available on input devices to the IBM 360.

+	plus
-	minus
*	times
/	divided by
** or @	raise to a power

The operation implied by these symbols depends on the class of the elements that appear to the left and right of the operator. In general the operation is the natural one for the class. For example, the meaning of A * B depends on the classes to which A and B belong. If A and B are matrices, the operation implied is that of matrix multiplication. If A and B are scalars, it is a scalar multiplication.

Schedules 3—6 contain a condensed description of the arithmetic operations in SPEAKEASY. In these schedules the class and also the form of each element of the resultant is indicated. Those involving mixed operations between arrays of 1 and 2 dimensions should be noted. They have been included to provide needed features for a compact directive program.

In the schedules the structure of objects (i. e., the number of elements, rows, or columns) is indicated by the parenthetic values. It should be

understood that operators must not connect objects with incompatible structures. For example, it is impossible to add a four-component array to a seven-component array. Similar restrictions exist in almost all cases.

2) Mathematical Expressions

A mathematical expression is constructed by connecting operators and operands. Redundant blank spaces have no significance, these and parentheses can be freely used to make the expression easy to read. Expressions are evaluated beginning with the innermost parentheses and working up through the entire expression. Within a particular parenthesis, the evaluation is first carried out for exponentiation, then for multiplication and division, and finally for addition and subtraction. The evaluation takes place from left to right. For example:

$$3 * A + B/C * E$$

is equivalent to

$$(3 * A) + (B/C) * E$$

As has been stated, the class of operands determines the actual operations. The expression

$$A + B * C$$

can have a variety of meanings depending on the classes of A, B, and C. Some examples are:

Class A	Class B	Class C	Resultant Class
M	M	M	M
S	S	S	S
V	M	M	M
V	M	V	V
A1	A1	S	A1
V	S	M	M
S	V	V	S

3) Mathematical Statements

a) Replacement Statement

Mathematical statements follow the usual computer notation. This means that an equal sign is best translated as the expression "is replaced by." Thus the statement

$$X = X + A$$

menas that X is now defined as the former X plus the quantity A.

In SPEAKEASY it should be noted that the class and structure of the object on the left is defined by the classes of the objects on the right by implication. As an example, in the above statement the class of X on the left and right need not be the same. If X was originally a scalar and A a square matrix, then the statement above would have redefined X to be a square matrix. This is an important property of SPEAKEASY and should be understood clearly.

Any expression can appear to the right of the equal sign. The left-hand side of the statement must, however, contain a single object name or an indexed reference to an element of an object. All objects appearing on the right-hand side must have been defined by appearing on the left-hand side of a previous statement. The object on the left need not have a previous definition and usually such previous definitions are irrelevant.

SPEAKEASY mathematical statements therefore can all be viewed as defining statements—defining the class, structure, and values for elements of the object on the left-hand side.

Some examples of SPEAKEASY mathematical-replacement statements are

```
X = 2.57 * SIN (X) / (3 * ALPHA)
Y = (MAX(X) + MIN(X))/(MAX(X) - MIN(X))
A = 2 * PI * R ** 2
```

b) Operations on Elements of an Object

The orientation of SPEAKEASY is towards operations on arrays of numbers treated as single entities. Whenever possible, this is not only the most convenient way to direct execution of a problem, it is also the most efficient. It is, however, sometimes necessary to use or set values for particular elements. This can be done by use of index values enclosed in parentheses following the name of the object. These indices may be expressions; in such cases they are always truncated to the next lower integer to obtain the true index value. Thus statements of the form

```
A(3, 5) = 2 * S (9.7 * SIN(X), 2 * I + 7)
```

or

```
A(2 * I + I) = SIN (X)
```

are allowed.

In the case of objects with a two-dimensional structure, addressing the array with a single index is equivalent to addressing the whole row or column. A comma may be used to indicate the missing index but is not necessary in addressing rows.

M (1) M (1,)	is row 1 of matrix M.
M (, 1)	is column 1 of matrix M.

Examples:

M (1) = 1	The entire row is set equal to 1 .
M (3) = M (, 4)	The fourth column replaces the third row.

c) Structured Indices

Operations on selected parts of structured objects are also possible. The indices described in the previous section were scalar quantities. In this section we shall describe the use of indices that are one-dimensional arrays. This natural extension of the notation provides the means of avoiding most of the logical branching common to operations on elements of an array in usual computer languages.

II. C3c

If I is a one-dimensional array and A is a structured object, then A(I) is a structured resultant of the same class as A but with the Ith elements selected as described in the previous section. Thus, if

```
I = ARRAY (1, 3)
V = VECTOR (1, 2, 3, 4)
M = MATRIX (3, 3: 1, 2, 3, 4, 5, 6, 7, 8, 9)
A = ARRAY (3, 2: 1, 2, 3, 4, 5, 6)
```

then

V(I) is a vector (1, 3)

M(I) is a 2 by 3 matrix $\begin{pmatrix} 1, & 2, & 3 \\ 7, & 8, & 9 \end{pmatrix}$

A(I) is a 2 by 2 array $\begin{pmatrix} 1, & 2 \\ 5, & 6 \end{pmatrix}$

i.e., M(I) and A(I) are rows 1 and 3 of M and A, respectively. Similarly

M(, I) is a 3 by 2 matrix $\begin{pmatrix} 1, & 3 \\ 4, & 6 \\ 7, & 9 \end{pmatrix}$

i.e., M(, I) consists of columns 1 and 3 of M.

If I and J are one-dimensional arrays and A is an object of a two-dimensional class as are the selected rows and columns, then in the above examples,

M(I, I) is the matrix $\begin{pmatrix} 1 & 3 \\ 7 & 9 \end{pmatrix}$

An example of the use of structured indices is

```
A = ARRAY (1, 2, 3, 4, 5)
I = ARRAY (2, 3, 4)
B = A(I) * I + A(I - 1)
```

The result will be the array

(5, 11, 19).

d) Automatic Extension of Defined Objects

If a defined structured object is referenced on the left-hand side of an equation and the indices refer to elements outside the range defined for that object, then the size of the object is increased to allow room for the newly defined elements. All extra elements created in this way but not explicitly set are put equal to zero. For example, if A is a defined three-component vector and a statement of the form

```
A(7) = 9
```

is encountered, then A will be extended to become a seven-component vector with elements 4, 5 and 6 set equal to zero. The newly created element 7 will be set equal to 9.

Similarly if A can now be extended to a 12-component vector by the statements

```
B = VECTOR (:10, 11, 12)
A(10) = B
```

The newly created elements 8 and 9 are set equal to zero and elements 10, 11 and 12 are set equal to the values 10 11 and 12, respectively.

4) Built-in Functions

In designing SPEAKEASY an attempt has been made to provide the most commonly used operations as an integral part of the language itself. In order to do this a very large number of special functions have been included. Many of these are natural extensions of the algebraic operations described already. Others are the straightforward extension of FORTRAN-like functions to structured objects. In addition, SPEAKEASY provides a large number of functions that lend themselves naturally to problems formulated in terms of structured variables. All of the functions described here can be used anywhere in any SPEAKEASY statement. The result is available for use within that statement. For example, the statement

```
Y = (MAX(X) - MIN(X))*SUM(SIN(X))
```

makes use of several built-in functions.

a) Element-by Element Functions

This set of functions operates on objects of any class and produces an object of the same class. Each element of the answer is the result of applying the function to the corresponding element of the original object. These functions are shown in Schedule 7. The allowed ranges of values are also indicated in that schedule.

b) Sums and Products of Elements

Since SPEAKEASY is oriented towards operations on structured objects as a whole, special functions must be provided to efficiently carry out the common operations such as obtaining the sums and products of elements of objects. The functions available are shown in Schedule 8.

c) Structure Functions

In order to obtain information about the structure of defined objects and specifics about its contents, a few special functions are provided. These are shown in Schedule 9. The answers in all cases are scalar objects.

d) Functions for One-Dimensional Arrays (Functions of 1 variable)

Although one-dimensional arrays can be used for many purposes, one rather common use is for defining functions of a single independent variable. For these applications a set of special SPEAKEASY functions are provided. In each case the orientation is toward two arrays, one the function and the other the array for the independent variable. A typical use might be the following sequence of statements.

X = GRID (0, 10, 0.1)	Defines a grid 0, 0.1, 0.2 · · · , 9.9, 10.
Y = SIN(X)*EXP(-X)*X*X	Evaluates the function $Y = SIN(X)e^{-X}X^2$
T = DERIV(Y:X)	Differentiates Y with respect to X

The functions of this type are described in Schedule 10.

e) Functions for Matrices

In order to carry out operations of matrix algebra, it is necessary to provide the standard functions of that field.* These forms are given in Schedule 11. Note that all operations except transposition are restricted to square matrices.

f) Ranking Functions

Two SPEAKEASY functions are provided to help order (i.e., rank) the elements of an object according to algebraic size. These functions are described in Schedule 12.

g) Transfamily functions

In order to make the full power of the operations of SPEAKEASY available to all problems, it is necessary to provide means of effectively altering the class of objects. This is done in SPEAKEASY by use of the special function shown in Schedule 13. These functions can be applied to any object; the resultant object is identical in structure but belongs to the specified family.

---

*The initial release of SPEAKEASY allows only real matrices in these functions.

h) Logical Functions

In keeping with the SPEAKEASY approach of dealing with objects as a whole, it is necessary to provide means of selecting groups of elements on a logical basis. A built-in SPEAKEASY function has been included to provide the indices corresponding to nonzero (i.e., "true") elements of one-dimensional structures. This function is called LOC or LOCS.

LOCS(A) gives the locations (indices) of nonzero elements of a one-dimensional object A. The answer is normally used as a structured index (as explained in Sec. II.C3c). For example,

```
GOODVALS = A(LOCS(A  GT 5))
```

will produce a new array containing only those values of A that are greater than 5.

D. LOGICAL AND RELATIONAL OPERATIONS

In addition to the common arithmetic operations described in the previous section, SPEAKEASY allows for relational and logical operations. These operations can be applied to variables of any of the classes and operate on an element-by-element basis. Results of applying these operators are either 1 (TRUE) or 0 (FALSE). For logical operations the operands are either TRUE (nonzero) or FALSE (zero). These operations provide the means for carrying out rather elaborate masking operations on arrays.

1) Operators

The logical and relational operators are expressed by special keywords.

a) Logical Operators

NOT	Logical not
OR	Inclusive or
AND	Logical sum

b) Relational Operators

EQ	Equal to
NE	Not equal to
GT	Greater than
LT	Less than
GE	Greater than or equal to
LE	Less than or equal to

2) Logical and Relational Expressions

These expressions can be used to form special-purpose objects. The logical and relational operators can connect either two objects of the same class or an object of any class with a scalar. The result of such operations is an entity of the same class as that of the object and has elements that are either 0 (False) or 1 (True). Nonzero input values for logical expressions are true, zero input values are false. Thus, A AND B will produce an object of the same class as A and B but its elements will be 1

wherever the corresponding element of A and B are both nonzero and 0 wherever either or both have a zero element. Similarly, A GT 7 will give a resultant of the same class as A but will have a 1 everywhere that the corresponding element of A is greater than 7. Similarly A OR B gives a resultant of the same class as A and B but its element will be 1 wherever the corresponding element of either A or B is nonzero. Finally, NOT A is an object with the same structure as A but with a zero everywhere that A is nonzero. It is 1 elsewhere.

Mixed logical, relational, and arithmetic expressions are allowed. In such cases the order of evaluation is 1) arithmetic operations, 2) relational operations, 3) logical NOT operations, and finally 4) logical OR and logical AND operations. Within each hierarchy, evaluation is from left to right.

An expression of the form

A GE B+C AND E NE F*G+H

is equivalent to

(A GE (B+C)) AND (E NE (F*G+H))

3) Logical and Relational Statements

Any expression of the form described in the previous section can be used to define a new object whose elements will have the value 0 or 1 at each prescribed location. In addition, by proper use of logical expressions the newly defined variable can be cast into special forms.

Suppose one is dealing with a function of one variable and one wishes to place an upper and lower bound on the elements. For example, if the element exceeds 10 or is less than 5 it is to be replaced by 0. This could be done by the statement

F = ((F LT 10) AND (F GT 5)) * F

E. CONDITIONAL STATEMENTS

Two forms of conditional statements are provided within the language. The first is of the form

IF (Expression) Statement

The expression must be a scalar. If the numerical value of this expression is nonzero, then the associated statement is carried out; otherwise it is ignored.

Example:

```
IF(A GT B)     B = 9
```

The second form of conditional statement is designed for array operations. It is of the form

WHERE (Expression) Statement

In this statement, the associated statement must be an equation and the class and structure of the resultant must be the same as that of the test expression. An element in the associated equation will be replaced only where the corresponding element in the test expression is true (nonzero).

Example:

```
WHERE (A GT 3)     A = A + 4
```

This operation provides essential masking operations within the SPEAKEASY language.

F. COMPUTATIONAL CONTROL STATEMENTS

This section is a description of several control statements that specify the mode of the calculation. Control statements remain in force until explicitly canceled or overridden by other control statements.

1) Domain

SPEAKEASY is designed to operate either in the domain of real or complex numbers. When operating in the real domain, any calculation that leads to imaginary or complex results is treated as an error. The user may alter the domain of the computation at will by inserting a statement of the form

DOMAIN (REAL) or DOMAIN (COMPLEX)

In default of an explicit statement, the domain is real.

2) Accuracy

During the execution of a SPEAKEASY program, whenever two numbers are compared for equality any number less than a small number (called the accuracy) is regarded as zero. The value of this number can be set by means of a statement of the form

ACCURACY (VAL)

which sets the accuracy equal to VAL. In default of such a statement, the value for accuracy is 10^{-8}.

3) Freeing Defined Objects

At any time during a calculation it is possible to free the definitions of objects. This can be done for selected variables by the statement

FREE (N1, N2, . . . , NN)

where N1, N2, ··· , NN are the names of defined objects. All defined numerical data can be freed at once by the statement

CLEARDATA

G. INPUT AND OUTPUT

The design of formats for output often reduces the computer user to counting on his fingers. This indeed seems odd in the context of the application.

SPEAKEASY automatically provides a set of output formats that will be more than satisfactory for most applications. These automatic formats enable users to forget about this phase of the computer run; he can be assured that he will obtain legible results without direct intervention.

The reason this section says so little about the standard input/output facilities is that the user need not concern himself directly with their operations. Disproportionately large subsections describe alternative ways of reading data or producing output; but for the most part these are specialized features that do not concern most users.

1) Input

a) Standard input

Introduction of small arrays of data or individual numbers is part of the structure of the language as described in Sec. II. B. Any variable may be defined or redefined at any time in order to give it specific values or structure; for this reason, most applications of SPEAKEASY do not have the programs separated from input data. Instead, all of the input data in a normal run will be imbedded directly in the SPEAKEASY statements. The rest of this section can therefore be disregarded for most applications.

b) The READ statement

A READ statement has been provided in the language to enable users to read information that has been punched in some specialized format. Data produced by other computer programs, or experimental data, normally will have a specialized format that would conflict with the standard SPEAKEASY forms. A format for input must be provided for reading these cards. This format is in fact a standard 360-FORTRAN IV format with the single restriction that no numerical information can be entered into arrays in fixed-point form. F, D, or E formats are equally acceptable. The

format is defined by defining a literal constant; e.g.,

FMT = '(3X, 5F12. 3)'

The input array must exist. The number of components to be read is determined by its size; e.g.,

A = ARRAY(15)

The input data are then read in response to a statement

READ (FMT: A)

Cards will be read until the number of elements required to fill the array have been loaded.

c) The Data File

An intermediate form of input exists for blocks of data that are too large to fit on a single card. The numbers can be written in any SPEAKEASY form, separated by commas or spaces. The approach taken is to define a special area, called a data file, containing the input data. This file can be loaded into specific defined structures at any later time. The data file is defined by placing it between two cards. The header card contains the word DATA followed by a space and then the name to be assigned to this data file. The last card contains the single word END.

Once defined, the information in a data block is retrieved by a LOADDATA statement; e.g.,

LOADDATA (A,NAME)

will load the object A with the first N values from the data file NAME. N is the number of elements of A or the number of numbers in NAME, whichever is smaller.

2) Output

In this section we will define the various facilities available to produce output. In each case, a simple direct statement will produce output in an acceptable format. Users may, however, exercise control over the output by special-purpose statements. In the following statements the word namelist is used to refer to a list of names of defined objects, each separated from the others by commas.

a) Printed output

i. Standard print statements

Two standard print options are available in SPEAKEASY. The first results in the printing of the selected objects. The printed form reflects both the numerical contents of an object and its structure. This statement is of the form

PRINT (namelist)

The second form of standard output provides a tabular form for printing one-dimensional objects. The columns are headed by the names of the objects and correlated elements of members of the name list are printed side by side. This statement is of the form

TABULATE (namelist)

The objects whose names appear in namelist should all have the same number of elements.

ii. Formatted print statement

A WRITE statement, identical in form to that of the READ statement described in the previous subsection, is provided for special applications. The printed output can have all of the carriage-control characters of FORTRAN. The statement is of the form

WRITE (FMT: A)

where FMT is the format defined as for READ and A is the defined object.

iii. Print-control statements

The standard print statements described above produce highly legible output. In designing the format for output, the SPEAKEASY processor examines the information to be printed and makes a series of decisions on how best to display it. The user is able to control these decisions to some extent by the special control statements described here.

The first set of control statements relate to vertical spacing on the page. The user may reposition the paper by the statements

SPACE(N)

or

NEWPAGE

SPACE(N) causes N lines to be skipped. NEWPAGE causes the next information printed to appear at the top of the next page.

If he so desires, the user may cause a title line to appear at the top of each new page obtained by this means. He does so by defining an object called PAGETITLE by the statement

PAGETITLE = 'any desired title'

In printing numerical data, it is possible to specify the number of significant figures desired. Five significant figures are printed out in default of explicit specification. The number of significant figures to be printed is set by the statement

SIGNIFICANCE(N)

where N is the number of figures desired. In addition one can specify the range of sizes within which numerical data must fall if they are to be printed. For this purpose,

SETNULL(VAL)

requests that any number whose absolute value is less than VAL be printed as 0; and

SETINFINITY(VAL)

requests that any number whose absolute value is greater than VAL be printed as INF.

Note that the above apply only to the printing of numerical data. The actual computed values are unaffected by these control statements. The default options are

SETNULL(10^{-30})

and

SETINFINITY (10^{+30})

In using the standard PRINT statement, each object is printed in an easily read form but no attempt is made to correlate the printing of several objects. A compact form of output is produced by using a minimum number of extra spaces to provide a uniform column width for each object. The user may correlate the printing of several objects by specifying a minimum column width to be used in printing. The statement

COLWIDTH(N)

prevents the print routine from using a column width of less than N characters. If N is 10 larger than the number of significant figures desired, the print width will be uniform for all objects composed of real numbers. This value of N can be obtained automatically by the control statement

AUTOTAB

iv. Implied print statements

For ease of printing individual objects, a special implied print convention is adopted in SPEAKEASY. If a statement without an equal sign is processed and that statement cannot be classified as corresponding to any SPEAKEASY command, the word PRINT is assumed to be implied before that statement. In addition, the original statement itself is printed. Examples of implied print statements are shown in Schedule 14.

b) Punched output

Punched cards can be obtained from SPEAKEASY by a format statement similar to the READ and WRITE statements described before.

This option is provided primarily to offer the user a means of transmitting information to other non-SPEAKEASY programs. The form of the statement is

PUNCH(FMT: A)

Here FMT is the predefined FORTRAN format (as in the READ statement) and A is the array to be punched.

c) Graphical output

Output in graphical form is a built-in feature of the SPEAKEASY processor. Initially only a paper form of output (CALCOMP) is available and only a limited number of options are provided. These limitations will be removed in later versions of the language.

i. Design statements

Because of the flexibility inherent in this form of output, it is normally necessary that the user design the form of his graphical output. Default options are provided, but it unlikely that all of those will be proper for any application. More automatic facilities will be provided in the future.

The size of a graph can be chosen by the user. The default size is 8 inches tall and 10 inches wide. The user may override these by control statements

HSIZE(X)
VSIZE(Y) $Y \leq 10$

where X and Y are the size (in inches) to be used in graphs drawn after this statement is encountered.

One must select the scale to be used in drawing graphs. This is done by specifying the numerical values corresponding to the limits for the vertical and horizontal scales.

HSCALE=(left, right)
VSCALE=(bottom, top)

Note that the values along the scales will be labeled at inch marks. The user should choose scales that provide simple values at such points. The default values of the scales are 0—8 for the vertical scale and 0—10 for the horizontal scale.

Several additional options are available for designing the structure of a graph. The user may select these by the single control statement

$$\text{SETPLOT}\left(\left\{\begin{matrix}\text{BOX}\\ \text{or}\\ \text{NOBOX}\end{matrix}\right\},\left\{\begin{matrix}\text{SCALES}\\ \text{or}\\ \text{NOSCALES}\end{matrix}\right\},\left\{\begin{matrix}\text{LINES}\\ \text{or}\\ \text{POINTS}\end{matrix}\right\}\right)$$

The options selected are

BOX	Draw a frame around graph	NOBOX	No frame
SCALES	Indicate values at 1" intervals	NOSCALES	Omit indication of scales
LINES	Join designated points by lines, leaving points unmarked	POINTS	Mark points with crosses, without joining points

The default options are BOX, SCALES, LINES.

The point-plotting mode can be generalized by the special control statement

PLOTSYMB(freq, symb)

where symb is an integer from 0 through 12 which designates one of 13 different symbols to be used in plotting data, and freq determines the frequency with which the symbol is plotted (1 means every point, 2 means every other one, 3 every third, etc.). A negative value for freq indicates that only the symbol should appear. A positive value means that a line should join successive points.

Three forms of literal labels are provided. The top of the graph may be titled by defining the variable

PLOTTITLE = 'any message'

The vertical scale can be labeled by

VLABEL = 'any message'

The horizontal scale can be labeled by

HLABEL = 'any message'

ii. Graphing statements

The overall format having been specified, a new graph is produced each time the statement

GRAPH (namelist: hobject)

is encountered.

This graph is a graph of the members of namelist in the vertical direction against the object hobject in the horizontal. All objects should be one-dimensional and real, and have the same number of elements. A two-dimensional object in namelist is treated as several one-dimensional objects, each composed of a row of the original object. Therefore if a two-dimensional object appears in namelist, it must have as many columns as hobject has elements.

Each time the GRAPH statement is encountered, a graph is drawn on a new area of paper. All of the design statements accompany GRAPH.

It is possible to add information to a graph that has already been drawn, e.g., to add points on a graph containing curves. This is done by the statement

ADDGRAPH (namelist: hobject)

This statement has the same meaning as the GRAPH statement except that a new area of paper is not used. Design statements (except those relating to BOX, SCALES, and labels) are reexamined prior to adding to the graph. The user may therefore freely alter the plotting format for each addition.

If no graph has been drawn, the first ADDGRAPH statement acts as if it were a GRAPH statement. This in conjunction with the statement

NEWGRAPH

which completes references to a previously drawn graph make it possible to entirely avoid the use of the GRAPH statement.

H. THE MANUAL MODE

A major portion of the SPEAKEASY language has now been described. No reference has been made to the possibility of conditionally executing groups of statements, or to the possibility of repeated execution of such a group a specified number of times. The subset of SPEAKEASY already described is nevertheless usable. The existence of structured objects and routines for manipulating them as single entities makes it possible to carry out many straightforward computations with a series of statements that are executed only once.

The mode of operation in which each SPEAKEASY statement is processed but not saved is referred to as the MANUAL MODE of operation. In the examples shown in Schedules 14—36, we present a set of SPEAKEASY calculations that use the facilities described in this chapter. The figures are reproductions of output from the card-input version of SPEAKEASY. Information relating to the conventions for card input is given in Appendix II.

III. STORED PROGRAMS

For all but the most straightforward calculations, it is necessary to repeatedly execute groups of statements. The use of stored programs for such purposes is familiar to most computer users. The programs, procedures, and subroutines of languages such as FORTRAN, PLI, and ALGOL constitute stored programs.

Stored programs are available in SPEAKEASY but they differ in important aspects from other languages. One of the most important differences is that defined objects have global definitions. This means that a given name refers to the same object whether the reference is in the manual mode or in any stored program. Any number of stored programs may simultaneously be defined in SPEAKEASY. Execution of any one of them can be initiated directly from the manual mode or during execution of any other stored program.

The purpose of this chapter is to describe the construction and execution of stored programs. Additional statements specific to stored programs are also described. For reasons of clarity, it is assumed that the programs are to be input on punched cards.

III. A

A. STRUCTURE OF A STORED PROGRAM

A stored program is defined by supplying cards beginning with a header card containing the word PROGRAM followed by a space and then the name of the program. The program is terminated by a card containing the single word END. All cards between these two constitute the program. Any card except these two may be labeled. The label, a SPEAKEASY name at the left on the card, is separated from the actual program statements by a colon.

In SPEAKEASY the name of a program is treated as a defined name. It should therefore never be the same as the name of an object used in computations.

Multi-statement cards are constructed of several SPEAKEASY statements separated from each other by semicolons. Multi-card statements can be constructed. If & is in the first column of a card, it is taken to be a continuation of the preceding card. Continuation cards may not be labeled.

```
PROGRAM  SAMPLE                                  Header card
    X = Q;    Y = 3.5;    Z = 27 * X             Multi-statement card
          ALPHA:    T = X + Y                    Labeled card
GAMMA:  W = T + X + Y; U = W +
&    Q - 3 * X                                   Continuation of GAMMA
 PRINT (X, Y, Z, T, W, U)
       END                                       End card
```

B. SPECIAL STATEMENTS FOR THE PROGRAM MODE

The statements described in this section are those whose use is restricted to the program mode. Two (RETURN and CONTINUE) are ignored if encountered during manual-mode execution. The others cannot be used in the manual mode since they refer to labeled statements or to groups of statements.

1) RETURN

The RETURN statement is used within a SPEAKEASY program to return the path of execution to the statement after the invoking statement. The next statement to be processed will be the one following the EXECUTE statement which invoked the program. A RETURN statement is always implied just before the END card of any program.

2) CONTINUE

A CONTINUE statement is a nonoperational statement to which a label can be attached.

3) GO TO

A GO TO or GOTO statement is used to alter the sequence of execution of statements. When a GO TO statement is encountered, the next statement executed will be the statement with the label indicated by the GO TO statement; e.g.,

```
GO TO A3
```

will transfer the path of execution to the statement with the label A3.

Logical branches are made by combining an IF statement with a GO TO statement such as

```
IF (A GT 7) GO TO ALPHA
```

which may be read as "If A is greater than 7 go to ALPHA, otherwise continue the sequential execution of statements."

4) FOR

A FOR loop is a section of a single program, beginning with a FOR statement and terminated by a corresponding ENDLOOP statement. All statements between these two are repeatedly executed as specified in the FOR statement.

A FOR statement is of the form

FOR n = start, stop

or

FOR n = start, stop, increment

Here n is the name of a scalar which may appear in any context within the FOR loop that does not alter the value of n; and start, stop, and increment are any scalar expressions (not involving n) whose values specify, respectively, the initial value of n, its final value, and the increment to be added to n every time the loop is repeated. If increment is not specified, its value is assumed to be 1.

The ENDLOOP statement is of the form

ENDLOOP n

where n is the name appearing in the corresponding FOR statement.

5) Nested FOR Loops

Up to 10 nested FOR loops are allowed in SPEAKEASY. Any FOR loop started within a FOR loop must be terminated within that loop.

Caution: The use of FOR loops in SPEAKEASY for operations available within the language is neither compact nor efficient. For example, if A and B are 5-component arrays, the statements

```
VAL = 0
    FOR I = 1,5
VAL = A(I) * B(I) + VAL
    ENDLOOP I
```

are equivalent to the single statement

```
VAL = SUM (A * B)
```

The latter is much more compact and makes use of the optimized features of the language.

The use of the built-in functions and structured algebra of SPEAKEASY is perhaps the most difficult problem facing users who are familiar with languages such as FORTRAN. It is important to understand that writing SPEAKEASY programs with FORTRAN conventions (such as extensive loops) defeats the purpose of the language.

The user is advised to begin by expressing a problem either in matrix notation or in ordinary mathematical subscript notation, the summations being explicitly indicated. He will then usually find that the problem can very readily be translated into a compact SPEAKEASY program without use of explicit subscript references or FOR loops.

C. EXECUTING A STORED PROGRAM

Once defined, any SPEAKEASY program can be executed by the statement

EXECUTE name

where name is the name of the program.

The execution of a program starts with its first statement and proceeds sequentially until this path is altered by a branching statement (GO TO). FOR loops result in repeated execution of selected sets of statements. If a RETURN statement or the END statement is encountered, the execution of this program is terminated and the statement after the one calling for execution of the program is then executed.

EXECUTE statements may occur in the manual mode or in any stored program. In the manual mode, the EXECUTE statement should occur alone and not as part of a multistatement card.

IV. AIDS TO ERROR DETECTION

All higher-level computer languages are intended to provide the means of quickly formulating and carrying out computations. A large fraction of the programmer's effort must normally be devoted to the process of finding and correcting errors in his programs. The extent to which a language meets its goals is therefore largely determined by how completely it detects errors and how well it informs the user of the faults found. Diagnostic facilities are therefore an essential feature of any higher-level language.

In SPEAKEASY the probability of errors is inherently small because of the compact and natural form of statements. In addition, the user can concentrate on the logical formulation of his own particular problem, since the built-in facilities of the language relieve him of the task of programming standard manipulations. It is therefore likely that even untrained users of SPEAKEASY will be able to write programs that work properly on the first attempt.

Correlated with this rather compact and easily used language is an extremely discriminating processor. The presence of structured objects in the language provides the SPEAKEASY processor with much more information than is available in other languages. Each algebraic operation, for instance, is preceded by an examination of the objects involved to see if they are compatible. Continuous checking of the calculation is therefore automatic and relatively complete. Any structural error is detected before it is able to propagate to later parts of the calculation. Thus the user is always presented with a detected error before it has had the chance to confuse the output. For involved computations, the fact that no error has been detected is some assurance that the structural aspects of the calculation are correct.

For a simple problem, these features combine to provide answers on the first try and offer some assurance that the processor has at least understood and checked the logic of parts of the program. For any but the most trivial problem, however, other facilities must be provided to enable users to follow the operations. This chapter describes those facilities and the normal error-detection features of SPEAKEASY.

Two classes of errors exist in SPEAKEASY. The first comprises the general syntax errors common to any language. Such errors include the use of illegal characters, parenthesis imbalances, etc. The second class of errors is specific to the structure of SPEAKEASY. Since definitions of objects may vary during a calculation, many errors can only be detected during execution. Such errors, referred to as execution errors, involve attempts to use undefined objects, to combine two objects that have incompatible structures, or to operate illegally with some structured object.

A. ERROR MESSAGES

In each case of ambiguity, the most likely intent of the statement is carried out. If this is not possible, however, a printed error message quotes the statement involved and describes the difficulty. Schedules 38 and 39 illustrate the form of error messages generated by SPEAKEASY.

1) Compilation Errors

During compilation of any user's SPEAKEASY stored program, the syntax of the program is examined. All syntactical errors are listed at the end of that program. Such errors do not affect the calculation until the program is executed.

2) Manual-Mode Errors

Each manual-mode statement is scanned to check the syntax before processing is attempted. If errors are detected the statement is printed along with an error message. Similarly execution errors are printed if detected during processing.

The next manual-mode statement will be processed in any event.

3) Errors in Execution of the Program

Errors of either class will result in the abortion of the SPEAKEASY program being executed. The error message will be printed and processing will normally continue with the next manual-mode statement. All currently defined information will be dumped. Commands described in Sec. IV. D can be used to alter these options.

B. DUMPS

It is frequently desirable for a user to examine all the information defined at a given point in a calculation. He may do this in SPEAKEASY with the single statement

DUMP

An easily-read complete printout of all defined objects will result. After this printout the calculation continues in the normal fashion.

C. AUTOPRINT

SPEAKEASY provides a particularly desirable feature for tracing the behavior of selected objects. This facility called AUTOPRINT enables a user to request that specified objects be printed every time they are evaluated. AUTOPRINT may be turned on or off by the use of appropriate statements. (Schedule 25 is an example of its use.)

The statement

AUTOPRINT (namelist)

where namelist is a set of object names separated by commas, will result in automatic printing of each of those objects every time they appear on the left in an equation.

The statement

AUTOPRINT

gives a complete printout of all objects as they are defined or altered.

The statement

AUTOPRINT (0)

turns off the automatic printing.

D. ERROR-CONTROL COMMANDS

While automatic dumping of currently defined data and continuation of computation in the manual mode are felt to be the desirable action after an error in program execution, provisions for user-chosen options are included in the language. The single command word ONERROR is used to control the options, the ones allowed at present being

$$\text{ONERROR} \left(\begin{array}{cc} \underline{\text{DUMP}} & \underline{\text{MANUAL}} \\ & , \\ \text{NODUMP} & \text{CONTINUE} \end{array} \right)$$

The underlined options are the standard defaults. NODUMP indicates that no dump of defined data is desired. CONTINUE means that the errors do not affect the path of execution.

APPENDIX I. KEYWORDS AND SYNONYMS

Several levels of keywords exist in SPEAKEASY. Some are restricted words that may be used only for their intended purpose. The number of this type is small. The majority of the keywords of the language are designed so that a user will not be affected by any that are not known to him. In such cases the use of a keyword as the name of an object automatically eliminates the normal function of that keyword. Its normal function will resume if the name is freed. For example, if the user's program has executed a statement of the form

```
SIN = 2.73
```

then the sine function is unavailable until the statement

```
FREE(SIN)
```

is encountered.

1) Restricted Words

The following is a list of restricted words. Users may not use these as names in SPEAKEASY. In addition, normal usage of these keywords in the manual mode requires that they occur in single-statement cards.*

CALL	FREE	PROCEDURE
CONTINUE	FUNCTION	RETURN
DATA	GLOBAL	RUN
DO	GOTO	SPACE
END	IF	SUBROUTINE
ENDLOOP	LOADDATA	USE
EXECUTE	LOCAL	WHERE
FIN	NEWPAGE	WHEREVER
FOR	PRINT	

*Some of the keywords in this list are included for future additions to the language. These are not yet restricted words but are included here for completeness.

2) Logical Operators

All logical operators are restricted keywords and may not be used as variable names. These are

NOT OR AND EQ NE GT LT GE LE

3) Nonrestricted Keywords and Synonyms

These keywords may also be used as names of objects. During the time their definitions as objects remain in force, the normal functions of these keywords are suppressed.

In designing the keywords, the objective was always to provide the "right" word. Often the decision narrowed down to alternative words that seemed equally good. Sometimes it was apparent that very short words would be desirable because of the frequency of their use within expressions. These small words, however, often appeared to reflect a bit of "computerese." For this reason a large number of synonyms were included. In the following list, we present the keywords grouped according to operations. Synonyms follow defined keywords. Examples of the use of these words are given in the schedules, as noted.

a) Defining Functions (Schedules 1 and 2)

VECTOR (VEC), MATRIX (MAT)
SYMMAT (SMAT), ASYMMAT (ASMAT), DIAGMAT (DMAT)
ARRAY, ARRAY2D, INTEGERS
GRID (VARIABLE)

b) Elemental Functions (Schedule 7)

ABS, SIGN, SQRT, EXP, LOG
SIN, COS, TAN, COT
ASIN, ACOS, ATAN, ACOT
FRACPART, REALPART, IMAGPART, CONJUGATE (CONJ)
SINH, COSH, GAMMA, LOGGAMMA

c) Sums and Products (Schedule 8)

SUM, SUMSQ, PROD
SUMROWS, SUMSQROWS, PRODROWS
SUMCOLS, SUMSQCOLS, PRODCOLS

d) Structure Functions (Schedule 9)

NOELS (LENGTH), NOROWS, NOCOLS
MIN, LOCMIN, ROWMIN, COLMIN
MAX, LOCMAX, ROWMAX, COLMAX

e) Functions of one variable (Schedule 10)

DERIV, INTEGRAL, TOTALINT
ROOTS, NOROOTS, INTERP

f) Matrix Operators (Schedule 11)

EIGENVALS, EIGENVECS, DET, DIAGELS
INVERSE, TRACE, TRANSPOSE (TRANSP)

g) Ranking Functions (Schedule 12)

RANKED, RANKER

h) Transfamily Functions (Schedule 13)

AFAM (ATYP, ATYPE)
VFAM (VTYP, VTYPE)
MFAM (MTYP, MTYPE)

i) Graphics (Sec. II. G2c)

GRAPH, ADDGRAPH, NEWGRAPH
HSCALE, VSCALE, HSIZE, VSIZE
SETPLOT, PLOTSYMB

j) Input/Output (Sec. II. G1, 2)

PRINT, TABULATE, WRITE, PUNCH
NEWPAGE, SPACE
AUTOTAB, COLWIDTH, SIGNIFICANCE, SETNULL, SETINFINITY
LOADDATA, DATA, READ

k) Commands (Sec. II. F)

FREE, DOMAIN, ACCURACY, CLEARDATA

ℓ) Program Mode (Sec. III)

PROGRAM, FOR, ENDLOOP
GOTO, RETURN, CONTINUE
RETURN, END, EXECUTE

m) Others

AUTOPRINT (Sec. IV. C)
DUMP (Sec. IV. B)
ONERROR (Sec. IV. D)

4) Alphabetic Listing of Keywords

This is an alphabetic listing of the keywords. Nonstandard synonyms have the standard form given in parentheses. Restricted keywords are underscored. Note that in very long keywords only the first 8 characters are meaninfgul; all others are ignored.

The numbers beside the keywords refer to the schedule containing a description a sample of the use of the word. If the word does not occur in a schedule, the reference is to the section describing the word, this is given by page number and indicated by enclosing parentheses.

USE

APPENDIX II. CARD-INPUT SPEAKEASY

1) Card-Input Conventions

SPEAKEASY jobs can be submitted on standard 80-column tabular cards. IBM-029 keypunch should be used in punching the cards. All 80 columns of cards are usable and all input is of a free-format form. Spaces between terms are usually ignored and the user may design input to reflect his own tastes.

It has been found that the usual FORTRAN cards provide a highly readable input form, i.e., statements normally start in column 7 unless they are labeled. Labels appear in columns 2—5 and a colon follows in column 6. It should be noted that these conventions appear desirable but are not necessary.

A single dollar sign indicates that all the rest of a card is a comment. Two dollar signs on a single card indicate that the part of the card between the dollar signs is a comment and is to be ignored.

Multistatement cards have semicolons separating the statements. Multicard statements (i.e., continuation cards) are allowed only in stored programs and are indicated by an $\mathcal{E}$ in column 1 of continued cards.

The processor will accept any number of continuation cards but only a limited complexity in a statement. For this reason the use of multicard statements should be avoided when possible and statements should be kept as concise as possible.

2) Job-control Cards

In order to run a SPEAKEASY job on the Argonne 360/75 the card deck should include the following card. This card is inserted between the accounting information and your SPEAKEASY cards.*

```
Your accounting information

//        EXEC  SPEAKEZ
          Your SPEAKEASY deck
```

*This form of deck is usable as of April 1971.

If graphical output is requested, two additional cards are needed to provide access to the Calcomp tape. The deck would then be of the form

Your accounting information

```
/*SETUP        DEVICE = 2400-7, ID = (, RING, SAVE, NL), DDNAME = PLOTTAPE
//             EXEC     SPEAKEZ, VERSION = GRAPHEZ
//PLOTTAPE     DD   UNIT = TAPE7TRK, DISP = (, PASS), LABEL = (, NL)
```

Your SPEAKEASY deck

APPENDIX III. SCHEDULES

This report is intended both as an introductory writeup and as a reference manual for users. For the latter role it is useful to have quick access to the tables and examples of the writeup. For this reason all schedules of the report have been collected together in this appendix. The titles are summarized here.

Schedule	
1, 2	Explicit defining expressions
3—6	Description of the algebraic operations
7	Element-by-element functions
8	Sum and product functions
9	Structure functions
10	Operators for functions of one variable
11	Matrix functions
12	Ranking functions
13	Transfamily functions

The rest of the schedules in this report are actual reproductions of part of a run made with the SPEAKEASY processor. They provide examples of the use of the language.

14	Operations with scalar objects
15	Examples of explicit definitions (matrix/vector family)
16	Examples of explicit definitions (array family)
17	Operations on square matrices
18	Matrix/vector operations
19	Operations on 1-dimensional arrays
20	Operations on 2-dimensional arrays
21	Operations between 1- and 2-dimensional arrays
22	Simple-index operations

Schedule

23	Structured-index operations
24	Logical and relational operations
25	The WHERE and IF statements
26	Sample uses of the logical function LOCS
27	Sample operations using element-by-element functions
28	Samples of sum and product functions
29	Use of built-in structure functions
30	Examples of the use of ranking functions
31	Sample transfamily operations
32	Samples of the use of special operations for functions of 1 variable
33	Sample of a function of 2 variables
34	Sample of a crude contour plot
35	Sample of the construction and use of a data file
36	Sample of the use of the graphical features of the language
37	Sample program and its execution
38	Errors detected during compilation
39	Execution error messages (manual mode)

Schedule 1. Explicit defining expressions for structured objects in the matrix- vector family. Note that all structure-defining quantities (i.e., n and m) must be positive integers.

Expression	Meaning	Comment
VECTOR(n)	Defines a vector with n components all of which are zero.	
VECTOR($e_1, e_2, \ldots, e_\ell$)	Defines an ℓ-component vector with components set to e_i.	$\ell > 1$
VECTOR(n: $e_1, e_2, \ldots, e_\ell$)	Defines an n-component vector with the first ℓ elements set to e_i. All others are zero.	$\ell \leq n$
MATRIX(n, m)	Defines an n × m matrix, all components of which are zero.	n rows, m columns
MATRIX(n, m: $e_1, e_2, \ldots, e_\ell$)	Defines an n × m matrix in which some elements are preset. They are entered row by row. All non-preset elements are zero.	$\ell \leq n \cdot m$
SYMMAT(n: $e_1, e_2, \ldots, e_\ell$)	Defines an n × n symmetric matrix. Elements are loaded in lower diagonal form by rows and then the portion above the diagonal is made symmetric.	$\ell \leq \frac{1}{2}n \cdot (n + 1)$
ASYMMAT(n: $e_1, e_2, \ldots, e_\ell$)	Defines an n × n antisymmetric matrix. Elements are loaded in lower diagonal form by rows and then the portion above the diagonal is made antisymmetric.	$\ell \leq \frac{1}{2}n \cdot (n - 1)$
DIAGMAT(n: $e_1, e_2, \ldots, e_\ell$)	Defines an n × n matrix with nonzero elements $e_1, e_2, \ldots, e_\ell$ along the diagonal.	$\ell \leq n$

Schedule 2. Explicit defining expressions for array objects.

Expression	Meaning	Comment
ARRAY(n)	Defines a 1-dimensional array with n components, all of which are zero.	
ARRAY(e_1, e_2, . . . , e_ℓ)	Defines a 1-dimensional array with components set to e_i.	Note: ARRAY(e_1, e_2) is a 2-component, 1-dim. array!
ARRAY(n: e_1, e_2, . . . , e_ℓ)	Defines a 1-dimensional array with the first ℓ elements set to e_i. The last (n - ℓ) elements are zero.	$e \leq n$
ARRAY(n, m:) or ARRAY2D(n, m)	Defines a 2-dimensional n × m array with all elements set to zero.	Note colon. (See note above.)
ARRAY(n, m: e_1, e_2, . . . , e_ℓ)	Defines a 2-dimensional n × m array with preset elements. Loaded row by row. Non-set elements are zero.	$\ell \leq n \cdot m$
GRID(lim1, lim2)	Defines a 1-dimensional array with 101 equally spaced elements starting at lim 1 and going to lim 2.	lim1 ≠ lim2
GRID(lim1, lim2, delta)	Defines a 1-dimensional array with elements equally spaced starting at lim 1 and adding delta until lim 2 is reached or passed.	lim1 ≠ lim2 delta ≠ 0
INTEGERS(n, m)	Defines a 1-dimensional array with elements containing the integers from n to m.	n and m have integer values.
INTEGERS(n, m, ℓ)	Defines a 1-dimensional array with elements containing the integers from n to m in steps of ℓ.	All n, m, and ℓ are integers.

Class of Right Operand

Class of Left Operand		Operator ±	S	M Family		A Family	
				V(n)	M(n,m)	A1(n)	A2(n,m)
S			$A = L \pm R$ Class S	$A_i = L \pm R_i$ Class V(n)	$A_{ij} = L\delta_{ij} \pm R_{ij}$ $n = m$ Class M(m,m)	$A_i = L \pm R_i$ Class A1(n)	$A_{ij} = L \pm R_{ij}$ Class A2(n,m)
M Family	V(n)		$A_i = L_i \pm R$ Class V(n)	$A_i = L_i \pm R_i$ Class V(n)	$A_{ij} = L_i\delta_{ij} \pm R_{ij}$ $n = m$ Class M(m,m)		
	M(n,m)		$A_{ij} = L_{in} \pm R\delta_{ij}$ $m = n$ Class M(n,n)	$A_{ij} = L_{ij} \pm R_i\delta_{ij}$ $m = n$ Class M(n,n)	$A_{ij} = L_{ij} \pm R_{ij}$ Class M(n,m)		
A Family	A1(n)		$A_i = L_i \pm R$ Class A1			$A_i = L_i \pm R_i$ Class A1(n)	$A_{ij} = L_i \pm R_{ij}$ Class A2(n,m)
	A2(p,n')		$A_{ij} = L_{ij} \pm R$ Class A2(p,n')			$A_{ij} = L_{ij} \pm R_j$ $n' = n$ Class A2(p,n)	$A_{ij} = L_{ij} \pm R_{ij}$ $p = n,\ n' = m$ Class A2(n,m)

Schedule 3. Description of the operation ± between objects of various classes. The subscripts refer to elements. A is the answer object, L the lefthand object, and R the righthand object.

± ± ± ± ± ± ± ± ± ± ±

Class of Right Operand

Operator *		S	M Family		A Family	
			V(n)	M(n,m)	A1(n)	A2(n,m)
S		$A = L * R$ Class S	$A_i = L * R_i$ Class V(n)	$A_{ij} = L * R_{ij}$ Class M(n,m)	$A_i = L * R_i$ Class F1(n)	$A_{ij} = L * R_{ij}$ Class F2(n,m)
M Family	V(n)	$A_i = L_i * R$ Class V(n)	$A = \sum_i L_i * R_i$ Class S inner product	$A_i = \sum_j L_j * R_{ji}$ Class V(m)		
M Family	M(p,n)	$A_{ij} = L_{ij} * R$ Class M(p,n)	$A_i = \sum_j L_{ij} * R_j$ Class V(p)	$A_{ij} = \sum_k L_{ik} * R_{kj}$ Class M(p,m)		
A Family	A1(n)	$A_i = L_i * R$ Class A1(n)			$A_i = L_i * R_i$ Class A1(n)	$A_{ij} = L_i * R_{ij}$ Class A2(n,m)
A Family	A2(p,n')	$A_{ij} = L_{ij} * R$ Class A2(p,n')			$A_{ij} = L_{ij} * R_j$ if $n' = n$ Class A2(p,n)	$A_{ij} = L_{ij} * R_{ij}$ if $p = n$, $n' = m$ Class A2(n,m)

Class of Left Operand

Schedule 4. Description of the operation * between objects of various classes. The subscripts refer to elements. A is the answer object, L the lefthand object, and R the righthand object.

* * * * * * * * * * * * *

Class of Right Operand

Class of Left Operand /		S	M Family		A Family	
			V(n)	M(n,n)‡	A1(n)	A2(n,m)
S		$A = L/R$ Class S		$A_{ij} = L * I_{ij}$ Class M(n,n)	$A_i = L/R_i$ Class A1(n)	$A_{ij} = L/R_{ij}$ Class A2(n,m)
M Family	V(n)	$A_i = L_i/R$ Class V(n)		$A_i = \sum_k L_k * I_{ki}$ Class V(n)		
M Family	M(p,n)	$A_{ij} = L_{ij}/R$ Class M(p,n)		$A_{ij} = \sum_k L_{ik} * I_{kj}$ Class M(p,n)		
A Family	A1(n)	$A_i = L_i/R$ Class A1(n)			$A_i = L_i/R_i$ Class A1(n)	$A_{ij} = L_i/R_{ij}$ Class A2(n,m)
A Family	A2(p,n')	$A_{ij} = L_{ij}/R$ Class F2(n')			$A_{ij} = L_{ij}/R_j$ if n' = n Class A2(p,n)	$A_{ij} = L_{ij}/R_{ij}$ if p = n, n' = m Class A2(n,m)

‡ Here the right operand must be a square matrix and I is its inverse

Schedule 5. Description of the operation / between objects of various classes. The subscripts refer to elements. A is the answer object, L the lefthand object, and R the righthand object.

/ / / / / / / / / / / / / / /

Class of Right Operand

Class of Left Operand		Operator @ or **	S	M Family		A Family	
				V(n)	M(m,n)	A1(n)	A2(n,m)
S			$A = L^{R}$ Class S			$A_i = L^{Ri}$ Class A1(n)	$A_{ij} = L^{Rij}$ Class A2(n,m)
M Family	V(m)			$A_{ij} = L_i * R_j$ Class M(m,n) outer product			
	M(m,n)		R^{th} power of L Class M(m,n) ‡				
A Family	A1(n)		$A_i = L_i^{R}$ Class A1(n)			$A_i = L_i^{R_i}$ Class A1(n)	$A_{ij} = L_i^{R_{ij}}$ Class A2(n,m)
	A2(p,n')		$A_{ij} = L_{ij}^{R}$ Class A2(p,n')			$A_{ij} = L_{ij}^{R_{ij}}$ n' = n Class A2(p,n)	$A_{ij} = L_{ij}^{R_{ij}}$ p = n, n' = m Class A2(n,m)

‡ R integer only. L must be square.

Schedule 6. Description of the operation ** between objects of various classes. The subscripts refer to elements. A is the answer object, L the lefthand object and R the righthand object. Within the table ** means exponentiation.

** ** ** ** ** ** **

Schedule 7. Element-by-element functions available in SPEAKEASY. The resultant object in each case is of the same class as X. Each element of the answer is the result of operation on the corresponding element of X.

Function	Meaning	Comment
SIGN(X)	± 1 where $X \lessgtr 0$; 0 where $X = 0$	Real X only
ABS(X)	Absolute magnitude	
FRACPART(X)	Fractional part	
INTPART(X)	Integer part	
REALPART(X)	Real part	
IMAGPART(X)	Imaginary part	
CONJUGATE(X)	Complex conjugate	
SQRT(X)	Square root	Real X only
EXP(X)	Exponent e^X	Real $X \leq 170$, $\lvert\text{imag } X\rvert < 5 \times 10^6$
LOG(X)	Natural logarithm	$X \neq 0$
SIN(X)	Sine	Real X only; $\lvert X\rvert < 10^{15}$
COS(X)	Cosine	
TAN(X)	Tangent	
COT(X)	Cotangent	
ASIN(X)	Arc sine	Real X only; $\lvert X\rvert \leq 1$
ACOS(X)	Arc cosine	
ATAN(X)	Arc tangent	Real X only
ACOT(X)	Arc cotangent	
SINH(X)	Hyperbolic sine	Real X only; $\lvert X\rvert < 170$
COSH(X)	Hyperbolic cosine	
GAMMA(X)	Γ function	Real X only; $10^{-50} < X < 56$
LOGGAMMA(X)	Natural logarithm of Γ function	$0 < X < 4 \times 10^{60}$

Schedule 8. Built-in SPEAKEASY functions for obtaining sums and products of elements of structured objects.

Function	Meaning	Comment
SUM(X)	Sum of all elements	Answer is scalar
SUMSQ(X)	Sum of squares of all elements	
PROD(X)	Product of all elements	
SUMROWS(X)	$\sum_j X_{ij}$	Answer is a 1-dimensional
SUMSQROWS(X)	$\sum_j (X_{ij})^2$	member of the family of X
PRODROWS(X)	$\prod_j X_{ij}$	
SUMCOLS(X)	$\sum_i X_{ij}$	Answer is a 1-dimensional
SUMSQCOLS(X)	$\sum_i (X_{ij})^2$	member of the family of X
PRODCOLS(X)	$\prod_i X_{ij}$	

Schedule 9. The built-in SPEAKEASY functions for obtaining information about the structure of objects.

Function	Meaning
NOELS(X) LENGTH(X)	The number of elements in the object X. If X is undefined, the answer is zero.
NOROWS(X)	Number of rows in X
NOCOLS(X)	Number of columns in X
MIN(X)	Minimum element in X
MAX(X)	Maximum element in X
LOCMIN(X)	Location of minimum of X. X is one-dimensional.
LOCMAX(X)	Location of maximum of X. X is one-dimensional.
ROWMIN(X)	Row containing minimum element of X
ROWMAX(X)	Row containing maximum element of X
COLMIN(X)	Column containing minimum element of X
COLMAX(X)	Column containing maximum element of X

Schedule 10. Built-in SPEAKEASY operators for functions of one variable. In these functions X is a one-dimensional array X = ARRAY $(x_1, x_2, x_3, \ldots, x_n)$.

Function	Meaning	Comment
DERIV(F:X)	dF/dx (derivative)	Numerical differentiation
INTEGRAL(F:X)	$\int_{x_1}^{x_i} F\,dx$ an integral with a variable upper limit	Numerical integration. (Answer is an array.)
TOTALINT(F:X)	$\int_{x_1}^{x_n} F\,dx$ definite integral	Numerical integration. (Answer is a scalar.)
INTERP(Y,F,X)	Numerical fitting, interpolation	Resultant is an array with values of the function F evaluated at the points Y. F(X) must be given.
ROOTS(F:X)	Finds x_i of roots of F	Trapezoidal interpolation
NOROOTS(F)	Number of roots of F	Uses sign changes

Schedule 11. Built-in SPEAKEASY functions for matrices. Note that aside from TRANSPOSE, these operations can be used only with square matrices.

Function	Meaning
EIGENVALS(X)	Eigenvalues in order of descending values*
EIGENVECS(X)	Unitary matrix whose rows are eigenvectors of X, belonging to the corresponding eigenvalues*
DET(X)	Determinant of matrix X
DIAGELS(X)	Diagonal elements in original order
INVERSE(X)	Inverse matrix Note: 1/X is also the inverse
TRACE(X)	Sum of the diagonal elements
TRANSPOSE(X)	Transpose of the object

*Restricted to real, symmetric matrices.

Schedule 12. Built-in SPEAKEASY functions for ranking the elements of a structured object.

Function	Meaning
RANKED(X)	Produces a new object of the same structure as X but with elements arranged in increasing numerical order.
RANKER(X)	For a one-dimensional object X. This function produces the indices of the elements of X arranged in order of increasing numerical order. RANKED(X) = X(RANKER(X))

Schedule 13. The built-in SPEAKEASY functions for respecifying the family of an object.

Function	Meaning
AFAM(X)	The resultant has the structure of X but is a member of the array family
VFAM(X) or MFAM(X)	The resultant has the structure of X but is a member of the matrix/vector family

```
INPUT...$
INPUT...$          SCHEDULE 14  OPERATIONS WITH SCALAR OBJECTS.
INPUT...$
INPUT...       X=27 ; Y=16 ; Z=X*Y+8/3 ; PRINT(X,Y,Z)
          X =  27 , Y =  16 , Z =  434.67
INPUT...       PI=2*ACOS(0) ; PRINT(PI)
          PI =  3.1416
INPUT...       X=SIN(3*PI/8);PRINT(X)
          X =  .92388
INPUT...$
INPUT...$      THE IMPLIED PRINT FEATURE IS DEMONSTRATED HERE.
INPUT...$
INPUT...       SIN(2*PI/3)
          SIN(2*PI/3)  =  .86603
INPUT...       SIN(3)**2+COS(3)**2
          SIN(3)**2+COS(3)**2  =  1
INPUT...       DOMAIN(COMPLEX)
INPUT...$
INPUT...$      THE COMPLEX DOMAIN IS NOW ALLOWED
INPUT...$
INPUT...       T=3+4I;EXP(T)
          EXP(T)  = -13.129-15.201I
INPUT...       SIN(T)**2+COS(T)**2
          SIN(T)**2+COS(T)**2  =  1
INPUT...       T**2
          T**2  = -7+24I
INPUT...       LOG(T)
          LOG(T)  =  1.6094+.9273I
```

```
INPUT...$
INPUT...$          SCHEDULE 15.  EXAMPLES OF EXPLICIT DEFINITIONS (MATRIX/VECTOR FAMILY)
INPUT...$
INPUT...       VECTOR(5);VECTOR(1,2,3);VECTOR(6:1,2,3,4)

          VECTOR(5)    (A VECTOR WITH 5 COMPONENTS)
           0  0  0  0  0

          VECTOR(1,2,3)    (A VECTOR WITH 3 COMPONENTS)
           1  2  3

          VECTOR(6:1,2,3,4)  (A VECTOR WITH 6 COMPONENTS)
           1  2  3  4  0  0
INPUT...       MATRIX(2,2);MATRIX(2,2:1,2,3)

          MATRIX(2,2)    (A 2 BY 2 MATRIX)
           0  0
           0  0

          MATRIX(2,2:1,2,3)  (A 2 BY 2 MATRIX)
           1  2
           3  0
INPUT...       SYMMAT(3:1,2,3,4);ASYMMAT(3:1,2,3,);DIAGMAT(1,2,3,4,)

          SYMMAT(3:1,2,3,4)    (A 3 BY 3 MATRIX)
           1  2  4
           2  3  0
           4  0  0

          ASYMMAT(3:1,2,3,)    (A 3 BY 3 MATRIX)
           0 -1 -2
           1  0 -3
           2  3  0

          DIAGMAT(1,2,3,4,)  (A 4 BY 4 MATRIX)
           1  0  0  0
           0  2  0  0
           0  0  3  0
           0  0  0  4
```

```
INPUT...$
INPUT...$        SCHEDULE 16.  EXAMPLES OF EXPLICIT DEFINITIONS (ARRAY FAMILY)
INPUT...$
INPUT...      ARRAY(5);ARRAY( 1,2 );ARRAY(5:1,2,3)

          ARRAY(5)    (A 5 COMPONENT ARRAY)
           0  0  0  0  0

          ARRAY( 1,2 )   (A 2 COMPONENT ARRAY)
           1  2

          ARRAY(5:1,2,3)  (A 5 COMPONENT ARRAY)
           1  2  3  0  0
INPUT...      ARRAY(2,3:);ARRAY(2,3:1,2,3,4,5)

          ARRAY(2,3:)    (A 2 BY 3 ARRAY)
           0  0  0
           0  0  0

          ARRAY(2,3:1,2,3,4,5)  (A 2 BY 3 ARRAY)
           1  2  3
           4  5  0
INPUT...      GRID(1.2,1.9,.1);INTEGERS(1,15)

          GRID(1.2,1.9,.1)    (A 8 COMPONENT ARRAY)
           1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9

          INTEGERS(1,15)  (A 15 COMPONENT ARRAY)
           1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
```

```
INPUT...$
INPUT...$        SCHEDULE 17.  OPERATIONS ON SQUARE MATRICES
INPUT...$
INPUT...       X=MATRIX(3,3:1,2,4,1,3);PRINT(X)

            X (A 3 BY 3 MATRIX)
            1  2  4
            1  3  0
            0  0  0
INPUT...       X=X+TRANSPOSE(X);PRINT(X)

            X (A 3 BY 3 MATRIX)
            2  3  4
            3  6  0
            4  0  0
INPUT...       X(3,3)=5;PRINT(X)

            X (A 3 BY 3 MATRIX)
            2  3  4
            3  6  0
            4  0  5
INPUT...       DET(X)
           DET(X)  = -81
INPUT...       TRACE(X)
           TRACE(X)  =  13
INPUT...       EIGENVALS(X)

           EIGENVALS(X)  (A VECTOR WITH 3 COMPONENTS)
            9         5.6056 -1.6056
INPUT...       EIGENVECS(X)

           EIGENVECS(X)  (A 3 BY 3 MATRIX)
           -.57735  -.57735  -.57735
            .098784 -.7513    .65252
            .8105   -.3197   -.4908
INPUT...$
INPUT...$      THE EIGENVECTORS ARE THE ROWS OF THIS MATRIX.
INPUT...$      THE VECTOR CORRESPONDING TO THE LARGEST EIGENVALUE IS FIRST.
INPUT...$
INPUT...       1/X

           1/X  (A 3 BY 3 MATRIX)
           -.37037   .18519   .2963
            .18519   .074074 -.14815
            .2963   -.14815  -.037037
INPUT...       X**2

           X**2  (A 3 BY 3 MATRIX)
            29  24  28
            24  45  12
            28  12  41
```

```
INPUT...$
INPUT...$       SCHEDULE 18.  MATRIX/VECTOR OPERATIONS.
INPUT...$
INPUT...      X=VECTOR(1,2,3,4);Y=VECTOR(3,1,2)
INPUT...$
INPUT...$     FORM THE OUTER PRODUCT OF X AND Y
INPUT...$
INPUT...      Z=X**Y ; PRINT(X,Y,Z)

          X (A VECTOR WITH 4 COMPONENTS)
          1  2  3  4

          Y (A VECTOR WITH 3 COMPONENTS)
          3  1  2

          Z (A 4 BY 3 MATRIX)
          3   1   2
          6   2   4
          9   3   6
          12  4   8
INPUT...      X*Z

         X*Z  (A VECTOR WITH 3 COMPONENTS)
          90  30  60
INPUT...$     THE INNER PRODUCT IS
INPUT...      X*X
         X*X  =  30
INPUT...      Z*Y

         Z*Y  (A VECTOR WITH 4 COMPONENTS)
          14  28  42  56
INPUT...      T=X**X;PRINT(T)

          T (A 4 BY 4 MATRIX)
          1   2   3   4
          2   4   6   8
          3   6   9   12
          4   8   12  16
INPUT...      T+1

         T+1  (A 4 BY 4 MATRIX)
          2   2   3   4
          2   5   6   8
          3   6   10  12
          4   8   12  17
INPUT...      T+X

         T+X  (A 4 BY 4 MATRIX)
          2   2   3   4
          2   6   6   8
          3   6   12  12
          4   8   12  20
```

```
INPUT...$
INPUT...$       SCHEDULE 19.  OPERATIONS ON 1-DIMENSIONAL ARRAYS
INPUT...$
INPUT...      X=ARRAY(5:1,2,3,4,5) ; Y=X+3 ; Z=X*Y
INPUT...      PRINT (X,Y ,Z)

          X (A 5 COMPONENT ARRAY)
          1  2  3  4  5

          Y (A 5 COMPONENT ARRAY)
          4  5  6  7  8

          Z (A 5 COMPONENT ARRAY)
          4   10  18  28  40
INPUT...      X/Z

         X/Z  (A 5 COMPONENT ARRAY)
          .25     .2      .16667  .14286  .125
INPUT...      X/3

         X/3  (A 5 COMPONENT ARRAY)
          .33333  .66667  1       1.3333  1.6667
INPUT...      X**X

         X**X  (A 5 COMPONENT ARRAY)
          1     4     27    256   3125
INPUT...      X**3

         X**3  (A 5 COMPONENT ARRAY)
          1     8     27    64    125
```

```
INPUT...$
INPUT...$       SCHEDULE 20.  OPERATIONS ON 2-DIMENSIONAL ARRAYS
INPUT...$
INPUT...      X=ARRAY(2,2:1,2,3,4) ; Y=X+1 ; Z=X*Y
INPUT...      PRINT (X,Y,Z)

          X (A 2 BY 2 ARRAY)
          1  2
          3  4

          Y (A 2 BY 2 ARRAY)
          2  3
          4  5

          Z (A 2 BY 2 ARRAY)
          2   6
          12  20
INPUT...      X/Y

          X/Y  (A 2 BY 2 ARRAY)
           .5       .66667
           .75      .8
INPUT...      X**Y

          X**Y  (A 2 BY 2 ARRAY)
           1       8
           81      1024
INPUT...      X**3

          X**3  (A 2 BY 2 ARRAY)
           1   8
           27  64
```

```
INPUT...$
INPUT...$        SCHEDULE 21.  OPERATIONS BETWEEN 1- AND 2-DIMENSIONAL ARRAYS
INPUT...$                      1D ARRAYS OPERATE FROM THE LEFT ON ROWS, FROM THE
INPUT...$                    RIGHT ON COLUMNS.
INPUT...$
INPUT...       X=ARRAY(3:1,2,3);Y=ARRAY(3,2:1,2,3,4,5,6);Z=ARRAY(2:1,2)
INPUT...       PRINT(X,Y,Z)

            X (A 3 COMPONENT ARRAY)
            1  2  3

            Y (A 3 BY 2 ARRAY)
            1  2
            3  4
            5  6

            Z (A 2 COMPONENT ARRAY)
            1  2
INPUT...       X*Y ;Y*Z

           X*Y    (A 3 BY 2 ARRAY)
            1   2
            6   8
            15  18

           Y*Z   (A 3 BY 2 ARRAY)
            1   4
            3   8
            5   12
INPUT...       X+Y;Y+Z

           X+Y    (A 3 BY 2 ARRAY)
            2  3
            5  6
            8  9

           Y+Z   (A 3 BY 2 ARRAY)
            2  4
            4  6
            6  8
INPUT...       X**Y;Y**Z

           X**Y    (A 3 BY 2 ARRAY)
            1    1
            8    16
            243  729

           Y**Z   (A 3 BY 2 ARRAY)
            1   4
            3   16
            5   36
```

```
INPUT...$
INPUT...$        SCHEDULE 22.  SIMPLE INDEX OPERATIONS.
INPUT...$
INPUT...       X=ARRAY(3,3:1,2,3,4,5,6,7,8,9); PRINT (X)

            X (A 3 BY 3 ARRAY)
            1  2  3
            4  5  6
            7  8  9
INPUT...        X(3,3);X(2);X(,2)
          X(3,3)    =  9

          X(2)    (A 3 COMPONENT ARRAY)
           4  5  6

          X(,2)   (A 3 COMPONENT ARRAY)
           2  5  8
INPUT...        X(3)=X(,2) ; PRINT(X)

           X (A 3 BY 3 ARRAY)
           1  2  3
           4  5  6
           2  5  8
INPUT...        X(3)=1 ; PRINT(X)

           X (A 3 BY 3 ARRAY)
           1  2  3
           4  5  6
           1  1  1
INPUT...        X(,3)=1 ; PRINT(X)

           X (A 3 BY 3 ARRAY)
           1  2  1
           4  5  1
           1  1  1
INPUT...$
INPUT...$        AUTOMATIC EXTENSION IS ILLUSTRATED BY
INPUT...$
INPUT...        X(4,5)=1 ; PRINT(X)

           X (A 4 BY 5 ARRAY)
           1  2  1  0  0
           4  5  1  0  0
           1  1  1  0  0
           0  0  0  0  1
```

```
INPUT...$
INPUT...$        SCHEDULE 23.  EXAMPLES OF STRUCTURED INDEX OPERATIONS
INPUT...$
INPUT...       A=ARRAY(3,3:1,2,3,4,5,6,7,8,9) ; I=ARRAY(1,2) ; PRINT(A,I)

          A (A 3 BY 3 ARRAY)
           1  2  3
           4  5  6
           7  8  9

          I (A 2 COMPONENT ARRAY)
           1  2
INPUT...       A(I) ; A(,I) ; A(I,I) ; A(I,I+1)

         A(I)     (A 2 BY 3 ARRAY)
          1  2  3
          4  5  6

         A(,I)     (A 3 BY 2 ARRAY)
          1  2
          4  5
          7  8

        A(I,I)     (A 2 BY 2 ARRAY)
          1  2
          4  5

        A(I,I+1)   (A 2 BY 2 ARRAY)
          2  3
          5  6
```

```
INPUT...$
INPUT...$        SCHEDULE 24.  LOGICAL AND RELATIONAL OPERATIONS
INPUT...$
INPUT...$
INPUT...        A=ARRAY(0,2,0,1) ; B=ARRAY(1,2,0,0)
INPUT...        AORB=A OR B
INPUT...        AANDB=A AND B
INPUT...        NOTA=NOT A
INPUT...        TABULATE (A,B,AORB,AANDB,NOTA)

            A   B   AORB AANDB NOTA

            0   1   1    0     1
            2   2   1    1     0
            0   0   0    0     1
            1   0   1    0     0
INPUT...$
INPUT...        A=ARRAY(0,1,2,3,4,5);B=6-A
INPUT...        AGTB=A GT B
INPUT...        ALTB=A LT B
INPUT...        AEQB=A EQ B
INPUT...        AGEB=A GE B
INPUT...        ALEB=A LE B
INPUT...        ANEB=A NE B
INPUT...        TABULATE(A,B,AGTB,ALTB,AEQB,AGEB,ALEB,ANEB)

            A   B   AGTB ALTB AEQB AGEB ALEB ANEB

            0   6   0    1    0    0    1    1
            1   5   0    1    0    0    1    1
            2   4   0    1    0    0    1    1
            3   3   0    0    1    1    1    0
            4   2   1    0    0    1    0    1
            5   1   1    0    0    1    0    1
INPUT...$
INPUT...        AGT0=A GT 0
INPUT...        ANE1=A NE 1
INPUT...        AMBNE0=A-B NE 0
INPUT...        TABULATE(A,B,AGT0,ANE1,AMBNE0)

            A   B   AGT0 ANE1 AMBNE0

            0   6   0    1    1
            1   5   1    0    1
            2   4   1    1    1
            3   3   1    1    0
            4   2   1    1    1
            5   1   1    1    1
```

```
INPUT...$
INPUT...$       SCHEDULE 25.  EXAMPLES OF WHERE AND IF STATEMENTS.
INPUT...$
INPUT...      X=ARRAY(1,2,3,4,5,6,7) ; Y=ARRAY(7) ; PRINT(X,Y)

          X (A 7 COMPONENT ARRAY)
          1  2  3  4  5  6  7

          Y (A 7 COMPONENT ARRAY)
          0  0  0  0  0  0  0
INPUT...      AUTOPRINT(Y)
INPUT...$
INPUT...$     AUTOPRINT IS USED IN HERE FOR AUTOMATICALLY PRINTING Y.
INPUT...$
INPUT...      WHERE (X GT 3) Y=4

          Y (A 7 COMPONENT ARRAY)
          0  0  0  4  4  4  4
INPUT...      WHERE (X GT 4) Y=X-1

          Y (A 7 COMPONENT ARRAY)
          0  0  0  4  4  5  6
INPUT...      WHERE (Y EQ 0)Y=-X

          Y (A 7 COMPONENT ARRAY)
         -1 -2 -3  4  4  5  6
INPUT...      WHERE (X+Y GT 1 AND X LT 7)Y=9

          Y (A 7 COMPONENT ARRAY)
         -1 -2 -3  9  9  9  6
INPUT...      X=7
INPUT...      IF (X LT 5) Y=Y-1
INPUT...      PRINT (Y)

          Y (A 7 COMPONENT ARRAY)
         -1 -2 -3  9  9  9  6
INPUT...      IF (X GT 5) Y=Y-1

          Y (A 7 COMPONENT ARRAY)
         -2 -3 -4  8  8  8  5
INPUT...      AUTOPRINT(0)
```

```
INPUT...$
INPUT...$       SCHEDULE 26.  SAMPLE USE OF LOCS (THE TRUTH FUNCTION).
INPUT...$
INPUT...      X=ARRAY(5,3,1,0,2,2.5,7,6.3,0)
INPUT...      LOCS(X); X(LOCS(X))

          LOCS(X)   (A 7 COMPONENT ARRAY)
           1  2  3  5  6  7  8

          X(LOCS(X))  (A 7 COMPONENT ARRAY)
           5    3    1    2    2.5  7    6.3
INPUT...      LOCS(FRACPART(X))

          LOCS(FRACPART(X))  (A 2 COMPONENT ARRAY)
           6  8
INPUT...       X(LOCS(FRACPART(X)))

          X(LOCS(FRACPART(X)))  (A 2 COMPONENT ARRAY)
           2.5  6.3
INPUT...      LOCS(X GT 2)

          LOCS(X GT 2)  (A 5 COMPONENT ARRAY)
           1  2  6  7  8
INPUT...      X(LOCS(X GT 2))

          X(LOCS(X GT 2))  (A 5 COMPONENT ARRAY)
           5    3    2.5  7    6.3
```

```
INPUT...$
INPUT...$        SCHEDULE 27.  SAMPLE OPERATIONS USING ELEMENT-BY-ELEMENT FUNCTIONS
INPUT...$
INPUT...       X=ARRAY(-2,-1.5,-1,0,2.5,7);ABSX=ABS(X);SIGNX=SIGN(X)
INPUT...         FRACX=FRACPART(X);INTX=INTPART(X);TABULATE(X,ABSX,SIGNX,FRACX,INTX)

          X     ABSX SIGNX FRACX INTX

         -2     2    -1      0    -2
         -1.5   1.5  -1     -.5   -1
         -1     1    -1      0    -1
          0     0     0      0     0
          2.5   2.5   1      .5    2
          7     7     1      0     7
INPUT...       X=ARRAY(0,1+1I,2-3I,4I)
INPUT...       REALX=REALPART(X);IMAGX=IMAGPART(X);CONJX=CONJUGATE(X)
INPUT...       TABULATE(X,REALX,IMAGX,CONJX)

          X      REALX IMAGX CONJX

          0      0      0     0
          1+1I   1      1     1-1I
          2-3I   2     -3     2+3I
         +4I     0      4    -4I
INPUT...       X=VECTOR(1,2,3,4)
INPUT...       SQRTX=SQRT(X);SINX=SIN(X);SINHX=SINH(X);GAMMAX=GAMMA(X)
INPUT...       TABULATE(X,SQRTX,SINX,SINHX,GAMMAX)

          X  SQRTX    SINX     SINHX    GAMMAX

          1  1        .84147   1.1752   1
          2  1.4142   .9093    3.6269   1
          3  1.7321   .14112   10.018   2
          4  2       -.7568    27.29    6
```

```
INPUT...$
INPUT...$        SCHEDULE 28.  SAMPLES OF SUM AND PRODUCT FUNCTIONS.
INPUT...$
INPUT...       X=MATRIX(2,3:1,2,3,4,5,6);PRINT X

          X (A 2 BY 3 MATRIX)
           1  2  3
           4  5  6
INPUT...       SUM(X);SUMSQ(X);PROD(X)
          SUM(X)    =  21
          SUMSQ(X)  =  91
          PROD(X)   =  720
INPUT...       SUMROWS(X); PRODROWS(X)

          SUMROWS(X)    (A VECTOR WITH 2 COMPONENTS)
           6   15

          PRODROWS(X)   (A VECTOR WITH 2 COMPONENTS)
           6   120
INPUT...       SUMCOLS(X); SUMSQCOLS(X)

          SUMCOLS(X)    (A VECTOR WITH 3 COMPONENTS)
           5  7  9

          SUMSQCOLS(X)  (A VECTOR WITH 3 COMPONENTS)
           17  29  45
INPUT...       PROD(INTEGERS(1,10))
          PROD(INTEGERS(1,10))  =  3628800
INPUT...       SUM(INTEGERS(1,20))
          SUM(INTEGERS(1,20))   =  210
```

```
INPUT...$
INPUT...$        SCHEDULE 29.  USE OF BUILT-IN STRUCTURE FUNCTIONS.
INPUT...$
INPUT...       X=MATRIX(2,3:-1,7,-2,4,1,0);PRINT(X)

          X (A 2 BY 3 MATRIX)
         -1  7 -2
          4  1  0
INPUT...       MIN(X);ROWMIN(X);COLMIN(X)
         MIN(X)    = -2
         ROWMIN(X)    =  1
         COLMIN(X)  =  3
INPUT...       MAX(X)
         MAX(X)  =  7
INPUT...       NOELS(X)
         NOELS(X)  =  6
INPUT...       NOCOLS(X)
         NOCOLS(X)  =  3
```

```
INPUT...$        SCHEDULE 30.  EXAMPLES OF THE USE OF RANKING FUNCTIONS.
INPUT...$
INPUT...$
INPUT...       X=ARRAY(1,2,-1,-7,4,-3);Y=X**2;PRINT(X,Y)

          X (A 6 COMPONENT ARRAY)
          1  2 -1 -7  4 -3

          Y (A 6 COMPONENT ARRAY)
          1   4   1   49  16  9
INPUT...       RANKED(X)

          RANKED(X)  (A 6 COMPONENT ARRAY)
          -7 -3 -1  1  2  4
INPUT...       RANKER(X)

          RANKER(X)  (A 6 COMPONENT ARRAY)
           4  6  3  1  2  5
INPUT...       X(RANKER(X))

          X(RANKER(X))  (A 6 COMPONENT ARRAY)
          -7 -3 -1  1  2  4
INPUT...       Y(RANKER(X))

          Y(RANKER(X))  (A 6 COMPONENT ARRAY)
           49  9   1   1   4   16
```

```
INPUT...$
INPUT...$       SCHEDULE 31.  SAMPLE TRANSFAMILY OPERATIONS
INPUT...$
INPUT...       X=ARRAY(1,2,3)
INPUT...       X;VFAM(X)

          X (A 3 COMPONENT ARRAY)
          1  2  3

         VFAM(X)  (A VECTOR WITH 3 COMPONENTS)
          1  2  3
INPUT...      X**X

         X**X  (A 3 COMPONENT ARRAY)
          1   4   27
INPUT...$
INPUT...      AFAM(VFAM(X)**VFAM(X))

         AFAM(VFAM(X)**VFAM(X))  (A 3 BY 3 ARRAY)
          1  2  3
          2  4  6
          3  6  9
```

```
INPUT...$       SCHEDULE 32.  SAMPLES OF THE USE OF SPECIAL OPERATIONS FOR
INPUT...$                     FUNCTIONS OF ONE VARIABLE.
INPUT...$
INPUT...       PI=2*ACOS(0);X=GRID(0,2*PI)
INPUT...       NOROOTS(COS(X));ROOTS(COS(X):X)
        NOROOTS(COS(X))    =  2

        ROOTS(COS(X):X)  (A 2 COMPONENT ARRAY)
         1.5708  4.7124
INPUT...       COSX=COS(X)
INPUT...       DCOSX=DERIV(COSX:X)
INPUT...       ICOSX=INTEGRAL(COSX:X)
INPUT...       SIGNIFICANCE(4)
INPUT...$      SELECT EVERY 4TH ELEMENT
INPUT...       I=INTEGERS(1,NOELS(X),4);X=X(I);COSX=COSX(I)
INPUT...       DCOSX=DCOSX(I)
INPUT...       ICOSX=ICOSX(I)
INPUT...       TANX=TAN(X)
INPUT...       SINX=SIN(X)
INPUT...       TABULATE (X,COSX,DCOSX,ICOSX,SINX,TANX,X)
```

X	COSX	DCOSX	ICOSX	SINX	TANX	X
0	1	-6.197E-5	0	0	0	0
.2513	.9686	-.2485	.2486	.2487	.2568	.2513
.5027	.8763	-.4814	.4816	.4818	.5498	.5027
.754	.729	-.6841	.6843	.6845	.9391	.754
1.005	.5358	-.8438	.8441	.8443	1.576	1.005
1.257	.309	-.9504	.9507	.9511	3.078	1.257
1.508	.06279	-.9974	.9977	.998	15.89	1.508
1.759	-.1874	-.9816	.982	.9823	-5.242	1.759
2.011	-.4258	-.9042	.9045	.9048	-2.125	2.011
2.262	-.6374	-.77	.7703	.7705	-1.209	2.262
2.513	-.809	-.5874	.5876	.5878	-.7265	2.513
2.765	-.9298	-.3679	.368	.3681	-.3959	2.765
3.016	-.9921	-.1253	.1253	.1253	-.1263	3.016
3.267	-.9921	.1253	-.1253	-.1253	.1263	3.267
3.519	-.9298	.3679	-.368	-.3681	.3959	3.519
3.77	-.809	.5874	-.5876	-.5878	.7265	3.77
4.021	-.6374	.77	-.7703	-.7705	1.209	4.021
4.273	-.4258	.9042	-.9045	-.9048	2.125	4.273
4.524	-.1674	.9816	-.982	-.9823	5.242	4.524
4.775	.06279	.9974	-.9977	-.998	-15.89	4.775
5.027	.309	.9504	-.9507	-.9511	-3.078	5.027
5.278	.5358	.8438	-.8441	-.8443	-1.576	5.278
5.529	.729	.6841	-.6843	-.6845	-.9391	5.529
5.781	.8763	.4814	-.4816	-.4818	-.5498	5.781
6.032	.9686	.2485	-.2486	-.2487	-.2568	6.032
6.283	1	6.197E-5	-1.153E-14	-1.168E-14	-1.168E-14	6.283

```
INPUT...$
INPUT...$       SCHEDULE 33.  SAMPLE OF A FUNCTION OF 2 VARIABLES.
INPUT...$
INPUT...        X=GRID(-1,1,.25); Y=GRID(0,2, .20)
INPUT...$
INPUT...$       FIRST CONSTRUCT A PAIR OF TWO DIMENSIONAL ARRAYS CONTAINING THE DESIRED
INPUT...$       VALUES OF X AND Y IN THEIR ROWS AND COLUMNS, RESPECTIVELY
INPUT...$
INPUT...        NX=NOELS(X);NY=NOELS(Y);Y=Y+ARRAY(NY,NX:) ;X=ARRAY(NY,NX:)+X
INPUT...$       NOTE THE ORDER OF THE ADDITIONS.  THE RESULTS ARE:
INPUT...        PRINT(X,Y)

          X (A 11 BY 9 ARRAY)
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1
         -1   -.75 -.5   -.25  0     .25  .5   .75  1

          Y (A 11 BY 9 ARRAY)
          0    0    0    0    0    0    0    0    0
          .2   .2   .2   .2   .2   .2   .2   .2   .2
          .4   .4   .4   .4   .4   .4   .4   .4   .4
          .6   .6   .6   .6   .6   .6   .6   .6   .6
          .8   .8   .8   .8   .8   .8   .8   .8   .8
          1    1    1    1    1    1    1    1    1
          1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2
          1.4  1.4  1.4  1.4  1.4  1.4  1.4  1.4  1.4
          1.6  1.6  1.6  1.6  1.6  1.6  1.6  1.6  1.6
          1.8  1.8  1.8  1.8  1.8  1.8  1.8  1.8  1.8
          2    2    2    2    2    2    2    2    2
INPUT...$       NOW ANY FUNCTION OF X AND Y CAN EASILY BE CONSTRUCTED:
INPUT...        FUNXY=X**2 +  3*Y*SIN(X+3/2); PRINT(FUNXY)

          FUNXY (A 11 BY 9 ARRAY)
          1       .5625  .25    .0625  0       .0625  .25    .5625  1
          1.268   .9715  .7549  .6319  .5985   .6529  .7956  1.029  1.359
          1.575   1.38   1.26   1.201  1.197   1.242  1.341  1.496  1.718
          1.863   1.789  1.765  1.771  1.795   1.834  1.887  1.963  2.077
          2.151   2.198  2.27   2.34   2.394   2.424  2.432  2.43   2.426
          2.438   2.607  2.774  2.909  2.992   3.014  2.979  2.897  2.795
          2.726   3.016  3.279  3.479  3.591   3.615  3.523  3.364  3.156
          3.014   3.425  3.784  4.048  4.189   4.195  4.068  3.83   3.514
          3.301   3.834  4.289  4.618  4.788   4.786  4.615  4.297  3.873
          3.589   4.243  4.794  5.187  5.386   5.376  5.16   4.766  4.232
          3.877   4.652  5.299  5.756  5.985   5.966  5.706  5.231  4.591
```

```
INPUT...$
INPUT...$        SCHEDULE 34   A SAMPLE OF A CRUDE CONTOUR PLOT
INPUT...$
INPUT...$     A SIMPLE CONTOUR PLOT OF THE FUNCTION DEFINED IN THE PREVIOUS
INPUT...$     SCHEDULE CAN BE PRODUCED THUS:
INPUT...      STEP=1; INTPART((FUNXY-MIN(FUNXY))/STEP)

INTPART((FUNXY-MIN(FUNXY))/STEP)  (A 11 BY 9 ARRAY)
 1  0  0  0  0  0  0  0  1
 1  0  0  0  0  0  0  1  1
 1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  2
 2  2  2  2  2  2  2  2  2
 2  2  2  2  2  3  2  2  2
 2  3  3  3  3  3  3  3  3
 3  3  3  4  4  4  4  3  3
 3  3  4  4  4  4  4  4  3
 3  4  4  5  5  5  5  4  4
 3  4  5  5  5  5  5  5  4
```

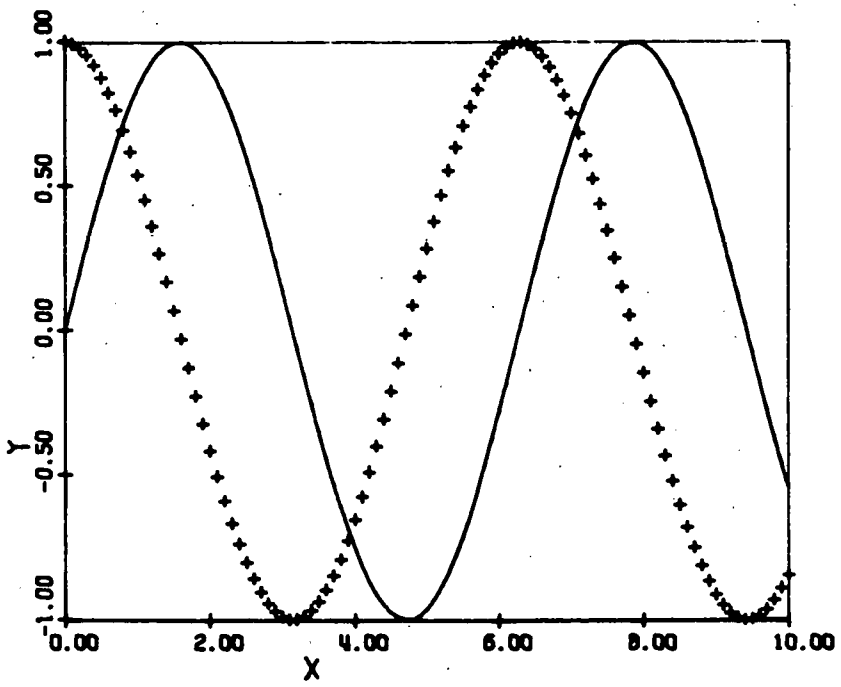
```
          DATA SHOWOFF
      1 , 2      3     4      5
             6           7  ,       8 ,9,10
      11     12
          END
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  COMPILATION TIME     0.08 SECS.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...$
INPUT...$
INPUT...$      IN THE FIRST EXAMPLE WE USE AN OBJECT SMALLER THAN THE DATA FILE.
INPUT...$
INPUT...       A=ARRAY(7)
INPUT...       LOADDATA(A,SHOWOFF)
INPUT...       PRINT(A)

        A (A 7 COMPONENT ARRAY)
        1  2  3  4  5  6  7
INPUT...$
INPUT...$      IN THIS EXAMPLE THE OBJECT HAS THE SAME SIZE AS THE DATA FILE.
INPUT...$
INPUT...       A=MATRIX(3,4)
INPUT...       LOADDATA(A,SHOWOFF)
INPUT...       PRINT(A)

        A (A 3 BY 4 MATRIX)
        1   2   3   4
        5   6   7   8
        9   10  11  12
INPUT...$
INPUT...$      IN THE LAST EXAMPLE WE USE AN OBJECT LARGER THAN THE DATA FILE.
INPUT...$
INPUT...       A=ARRAY(20)
INPUT...       LOADDATA(A,SHOWOFF)
INPUT...       PRINT(A)

        A (A 20 COMPONENT ARRAY)
        1   2   3   4   5   6   7   8   9   10   11   12   0   0   0   0   0   0   0   0
INPUT...$
INPUT...$
INPUT...$        SCHEDULE 35.  SAMPLE OF THE CONSTRUCTION AND USE OF A DATA FILE.
INPUT...$
```

```
INPUT...$
INPUT...$       SCHEDULE 36.  A SAMPLE OF THE USE OF THE GRAPHICAL FEATURES
INPUT...$                     OF THE LANGUAGE.
INPUT...$
INPUT...        X=GRID(0,10) ; Y=SIN(X)
INPUT...        D=DERIV(Y:X)
INPUT...$       THE DESIGN FEATURES ARE SETUP HERE.
INPUT...$
INPUT...$
INPUT...        HSIZE(5) ;  VSCALE(-1,1)  ;  HSCALE(MIN(X),MAX(X)) ; VSIZE(4)
INPUT...        GRAPH(Y:X)
INPUT...        SETPLOT(POINTS)
INPUT...        ADDGRAPH(D :X)
```



```
            PROGRAM SAMPLE1                                                    1
          $                                                                    2
          $     THIS PROGRAM ILLUSTRATES THE FORM OF A PROGRAM                 3
          $                                                                    4
            PRINT('EXECUTION OF SAMPLE1 FOR',N)                                5
            FOR I=2,N                                                          6
            X=VFAM(INTEGERS(1,I));Z=X**X;S=SUMROWS(Z)                          7
            IF (I GT 2) GO TO A                                                8
            PRINT(X,Z,S)                                                       9
            GO TO B                                                           10
          A:TABULATE(X,S)                                                     11
          B:ENDLOOP I                                                         12
            PRINT('DONE')                                                     13
            END
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     COMPILATION TIME     0.27 SECS.
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...$
INPUT...       N=3
INPUT...$
INPUT...$      THIS SETS THE VALUE OF N. IT IS GLOBAL AND THEREFORE
INPUT...$      AVAILABLE TO THE PROGRAM
INPUT...$
INPUT...       EXECUTE SAMPLE1
         EXECUTION OF SAMPLE1 FOR N =  3

          X (A VECTOR WITH 2 COMPONENTS)
          1  2

          Z (A 2 BY 2 MATRIX)
          1  2
          2  4

          S (A VECTOR WITH 2 COMPONENTS)
          3  6

          X  S

          1  6
          2  12
          3  18
         DONE
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     EXECUTION TIME     0.39 SECS.
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...$
INPUT...$      SCHEDULE 37.  A SAMPLE PROGRAM AND ITS EXECUTION
```

```
            PROGRAM MISTAKES                                                        1
        $                                                                           2
        $   THIS PROGRAM DEMONSTRATES SYNTAX ERROR MESSAGES DURING COMPLILATION.    3
        $                                                                           4
            X=Y**/Z  ;X=Y=Z ; LT=5 ;X=2.35.7                                        5
            V=A*(3+4 ;  ;  A=-*B  ; X="YES"                                         6
            T=3(4+5) ; X=5   ,    T=7                                               7
            END                                                                     8
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     IN STAT.$ X=Y**/Z    $   DOUBLE OP.
     IN STAT.$ X=Y=Z    $   DOUBLE = SIGN.
     IN STAT.$ LT=5    $   DOUBLE OP.
     IN STAT.$ X=2.35.7    $   MISPLACED DEC PT.
     IN STAT.$ V=A*(3+4    $   PARENTHESIS IMBALANCE.
     IN STAT.$ A=-*B    $   DOUBLE OP.
     IN STAT.$ X="YES"    $   ILLEGAL CHAR.
     IN STAT.$ X=5 , T=7    $   DOUBLE = SIGN.
   COMPILATION TIME    0.16 SECS.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...$
INPUT...$      SCHEDULE 38.  ERRORS DETECTED DURING COMPILATION.  THESE DO NOT
INPUT...$                        INHIBIT EXECUTION
```

```
INPUT...$
INPUT...$          SCHEDULE 39.  EXECUTION ERROR MESSAGES (MANUAL MODE)
INPUT...$
INPUT...       X=WW*10
          WW IS NOT DEFINED IN STAT.$ X=WW*10
INPUT...       X=4*3+2*(1+
          IN STAT.$ X=4*3+2*(1+  $   PARENTHESIS IMBALANCE.
INPUT...       X=4*3(7+6)
          IN STAT.$ X=4*3(7+6)  $   IMPLIED MULT?
INPUT...       X=MATRIX(3,3)
INPUT...       Y=X(4,2)
          X IN STAT.$ Y=X(4,2)  $   INDEX OUTSIDE BOUNDS.
INPUT...       X=ARRAY(3);Y=ARRAY(4);X+Y
          IN STAT.$ X+Y  $   X AND Y ARE INCOMPATIBLE FOR OPERATION

           X (A 3 COMPONENT ARRAY)
           0  0  0

           Y (A 4 COMPONENT ARRAY)
           0  0  0  0
INPUT...       DOMAIN(REAL); X=SQRT(-2)
INPUT...$
          IN STAT.$ X=SQRT(-2)  $   ENTERED COMPLEX DOMAIN.
```