ANL-8000 ANL-8000

# The SPEAKEASY-3 Reference Manual

樂語

U of C-AUA-USAEC

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

Prepared for the U.S. ATOMIC ENERGY COMMISSION
under Contract W-31-109-Eng-38

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.

**NOTICE**

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights.

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22151
Price: Printed Copy $7.60; Microfiche $1.45

a-

ANL-8000
Mathematics and Computers

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

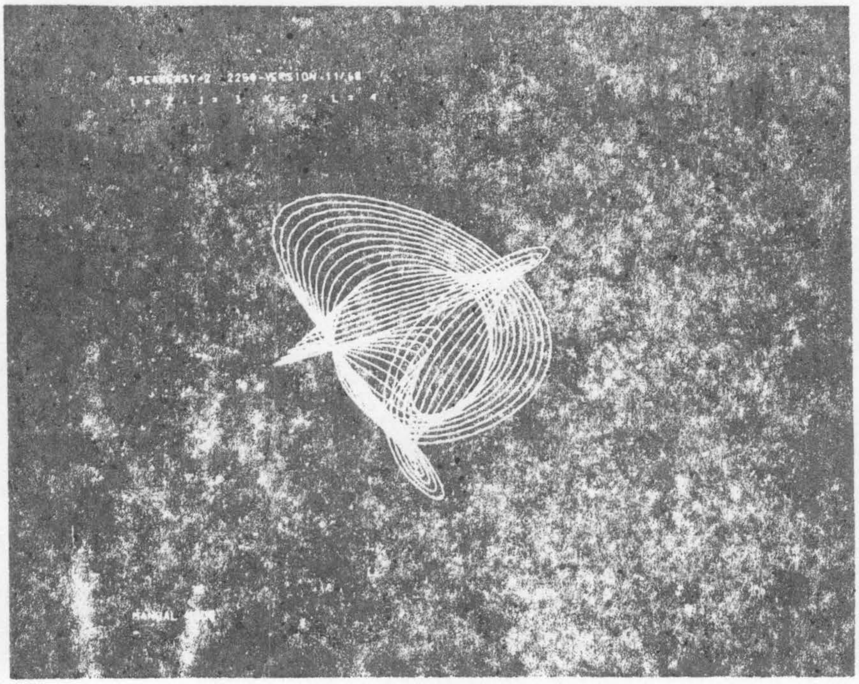
# The SPEAKEASY-3 Reference Manual

Compiled by

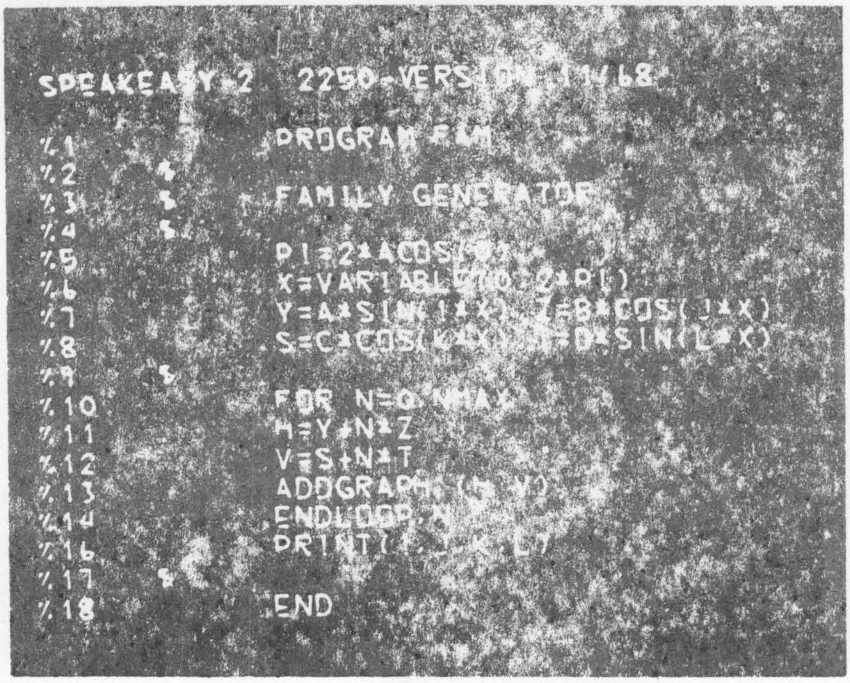
Stanley Cohen

Physics Division

May 1973



```
SPEAKEASY-2  2250-VERSION 11/68
I = 2 , J = 3 , K = 2 , L = 4
```



```
SPEAKEASY 2  2250-VERSION 11/68

%1          PROGRAM FAM
%2     $
%3     $    FAMILY GENERATOR
%4     $
%5          PI=2*ACOS(0)
%6          X=VARIABLE(0,2*PI)
%7          Y=A*SIN(I*X),Z=B*COS(J*X)
%8          S=C*COS(K*X),T=D*SIN(L*X)
%9     $
%10         FOR N=0,NMAX
%11         H=Y+N*Z
%12         V=S+N*T
%13         ADDGRAPH (H,V)
%14         ENDLOOP N
%16         PRINT(I,J,K,L)
%17    $
%18         END
```

## TABLE OF CONTENTS

TABLE OF CONTENTS

## TABLE OF CONTENTS

TABLE OF CONTENTS

The SPEAKEASY-3 Reference Manual

## ABSTRACT

SPEAKEASY is a computer language designed to provide access to information stored in a computer. Ease of use, natural notation, and built-in capabilities for growth are important features of SPEAKEASY. This book, consisting of four sections, combines most of the available documentation of the language. The first section is an introductory specification of the language accompanied by a large number of illustrative examples of its use in normal batch processing. The second section describes important extensions to the language. This section emphasizes facilities available for interactive usage. The third section describes the means by which the language itself is extended. Although not necessary for the casual user of the language, this section provides the information necessary to meet his later needs. The last section is a complete printout of the documentation internally available to users through the HELP processor.

This manual is intended as the primary information source for users of SPEAKEASY. It should enable such users to make effective use of the computer for their day-to-day calculations.

# PART ONE

# AN INTRODUCTION TO SPEAKEASY

by

S. Cohen and C. M. Vincent

# 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 section describes the basic language and shows how this subset of the language can be used in the so-called manual mode. The third section describes the use of stored programs. The last section describes the diagnostic features of the language. Each section 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 Part. Since this appendix 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 | $1.057 \times 10^{-5}$ |
| 236E+26 | means | $2.36 \times 10^{28}$ |
| 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 dimension) Class [A1]

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

e) Arrays (2 dimension) 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 named variable are not fixed quantities and may vary dynamically during a calculation.

Each named 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 named 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 not 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

$(0, \Delta, 2\Delta, \cdots, 10 - \Delta, 10)$

where $\Delta$ 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 |
| ** | 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, pp. 59—62, 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
```

means 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)<br>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.

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.

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 that are enclosed in periods

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.

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 some SPEAKEASY processors. For those versions the following represent some of the general control features available. Most are for CALCOMP output. Only certain are usable for other graphical devices. The descriptions are for CALCOMP plots.

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 where $Y \leqslant 10$

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.

## 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.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* PROGRAM ONE
* X=1
* EXECUTE TWO
* PRINT X Y
* END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* PROGRAM TWO
* Y=X+8
* RETURN
* END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...EXECUTE ONE
       X =  1 Y =  9
```

## D. SAMPLE PROGRAMS

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* PROGRAM LOOK
* MAXX=MAX(X);MINX=MIN(X);NOELSX=NOELS(X)
* AV=AVERAGE(X)
* RMS=SQRT(SUM((X-AV)**2)/NOELSX)
* PRINT (MAXX,MINX,NOELSX,AV,RMS)
* END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...X=1 2 3 4 5
INPUT...EXECUTE LOOK
MAXX =  5 MINX =  1 NOELSX =  5 AV =  3 RMS =  1.4142
INPUT...X=ARRAY(:X,X)
INPUT...EXECUTE LOOK
MAXX =  5 MINX =  1 NOELSX =  10 AV =  3 RMS =  1.4142
INPUT...X=X+1
INPUT...EXECUTE
MAXX =  6 MINX =  2 NOELSX =  10 AV =  4 RMS =  1.4142
```

Other examples of programs are shown on pp. 93 and 137.

## 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 calçulation, 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.

When SPEAKEASY is used in an interactive environment the DUMP option is modified to provide the user with the names of currently defined objects. He may then selectively print the information of interest.

## 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

ENDAUTOPRINT

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

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

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
CONTINUE
DATA
DO
END
ENDLOOP
EXECUTE
FIN
FOR

FREE
FUNCTION
GLOBAL
GOTO
IF
LOADDATA
LOCAL
NEWPAGE
PRINT

PROCEDURE
PROGRAM
RETURN
RUN
SPACE
SUBROUTINE
USE
WHERE
WHEREVER

*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. 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
VFAM
MFAM

i) Graphics (page 30)

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

j) Input/Output (page 25)

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

k) Commands (page 24)

FREE, DOMAIN, ACCURACY, CLEARDATA

ℓ) Program Mode (pages 36-39)

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

m) Others

| | |
|---|---|
| AUTOPRINT | (page 45) |
| DUMP | (page 44) |
| ONERROR | (page 46) |

### 3. 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 meaningful; all others are ignored.

The numbers beside the keywords refer to the schedule containing a description or 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.

| | |
|---|---|
| ABS | 7 |
| ACOS | 7 |
| ACOT | 7 |
| ACCURACY | (24) |
| ADDGRAPH | 36 |
| AFAM | 13 |
| AND | 24 |
| ARRAY | 2 |
| ARRAY2D | 2 |
| ASIN | 7 |
| ASMAT(ASYMMAT) | 1 |
| ASYMMAT | 1 |
| ATAN | 7 |
| AUTOPRINT | 25 |
| AUTOTAB | (29) |
| CALL | |
| CLEARDATA | (24) |
| COLMAX | 9 |
| COLMIN | 9 |
| COLWIDTH | (29) |
| CONJ(CONJUGATE) | 7 |
| CONJUGATE | 7 |
| CONTINUE | (36) |
| COS | 7 |
| COSH | 7 |
| COT | 7 |

| | |
|---|---|
| DATA | 35 |
| DERIV | 10 |
| DET | 11 |
| DIAGELS | 11 |
| DIAGMAT | 1 |
| DO | |
| DOMAIN | 14 |
| DMAT(DIAGMAT) | 1 |
| DUMP | (44) |
| EIGENVALS | 11 |
| EIGENVECS | 11 |
| END | 37 |
| ENDLOOP | 37 |
| EXECUTE | 37 |
| EXP | 7 |
| FIN(ENDLOOP) | |
| FOR | 37 |
| FRACPART | 7 |
| FREE | (24) |
| FUNCTION | |
| GAMMA | 7 |
| GLOBAL | |
| GOTO | 37 |
| GRAPH | 36 |
| GRID | 2 |

## APPENDIX II. Card-input SPEAKEASY

### 1. <u>Card-Input Conventions</u>

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 <u>Job-control Cards</u>

In order to run a SPEAKEASY job on the Argonne 360/195 the deck shown below should be placed in front of your SPEAKEASY cards.*

```
// jbname  JOB (badge,,,), CLASS=C, REGION=260K
           Your account card

//         EXEC    SPEAKEZ
           Your SPEAKEASY deck
```

*This form of deck is usable as of May 1973.

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          UNIT=2400—7, ID=(780300, RING, SAVE, NL), DDNAME=PLOTTAPE
//               EXEC       SPEAKEZ, VERSION=GRAPHEZ
//PLOTTAPE       DD  UNIT = TAPE7TRK, DISP = (,PASS), LABEL = (,NL),
//                VOL = SER = 780300
                        Your SPEAKEASY deck
```

The last SPEAKEASY card should read ENDDRAW(0).

## 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 $\ell$-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 $\ell$ 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($e_1$, $e_2$, . . . , $e_\ell$) | 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: $e_1$, $e_2$, . . . , $e_\ell$) | Defines a 1-dimensional array with the first $\ell$ elements set to $e_i$. The last (n - $\ell$) elements are zero. | e ≤ n |
| ARRAY(n, m:)<br>or<br>ARRAY2D(n, m) | Defines a 2-dimensional n × m array with all elements set to zero. | Note colon. (See note above.) |
| ARRAY(n, m: $e_1$, $e_2$, . . . , $e_\ell$) | 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<br>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, $\ell$) | Defines a 1-dimensional array with elements containing the integers from n to m in steps of $\ell$. | All n, m, and $\ell$ are integers. |

Class of Right Operand

| Operator ± | | | M Family | | A Family | |
|---|---|---|---|---|---|---|
| | | S | V(n) | M(n,m) | A1(n) | A2(n,m) |
| | S | $A = L \pm R$<br>Class S | $A_i = L \pm R_i$<br>Class V(n) | $A_{ij} = L\delta_{ij} \pm R_{ij}$<br>$n = m$<br>Class M(m,m) | $A_i = L \pm R_i$<br>Class A1(n) | $A_{ij} = L \pm R_{ij}$<br>Class A2(n,m) |
| Class of Left Operand<br>M Family | V(n) | $A_i = L_i \pm R$<br>Class V(n) | $A_i = L_i \pm R_i$<br>Class V(n) | $A_{ij} = L_i\delta_{ij} \pm R_{ij}$<br>$n = m$<br>Class M(m,m) | | |
| | M(n,m) | $A_{ij} = L_{in} \pm R\delta_{ij}$<br>$m = n$<br>Class M(n,n) | $A_{ij} = L_{ij} \pm R_i\delta_{ij}$<br>$m = n$<br>Class M(n,n) | $A_{ij} = L_{ij} \pm R_{ij}$<br>Class M(n,m) | | |
| A Family | A1(n) | $A_i = L_i \pm R$<br>Class A1 | | | $A_i = L_i \pm R_i$<br>Class A1(n) | $A_{ij} = L_i \pm R_{ij}$<br>Class A2(n,m) |
| | A2(p,n') | $A_{ij} = L_{ij} \pm R$<br>Class A2(p,n') | | | $A_{ij} = L_{ij} \pm R_j$<br>$n' = n$<br>Class A2(p,n) | $A_{ij} = L_{ij} \pm R_{ij}$<br>$p = n, n' = m$<br>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 | |
|---|---|---|---|---|---|---|
| Class of Left Operand | | | V(n) | M(n,m) | A1(n) | A2(n,m) |
| | S | $A = L * R$ <br> Class S | $A_i = L * R_i$ <br> Class V(n) | $A_{ij} = L * R_{ij}$ <br> Class M(n,m) | $A_i = L * R_i$ <br> Class F1(n) | $A_{ij} = L * R_{ij}$ <br> Class F2(n,m) |
| M Family | V(n) | $A_i = L_i * R$ <br> Class V(n) | $A = \sum_i L_i * R_i$ <br> Class S <br> inner product | $A_i = \sum_j L_j * R_{ji}$ <br> Class V(m) | | |
| | M(p,n) | $A_{ij} = L_{ij} * R$ <br> Class M(p,n) | $A_i = \sum_j L_{ij} * R_j$ <br> Class V(p) | $A_{ij} = \sum_k L_{ik} * R_{kj}$ <br> Class M(p,m) | | |
| A Family | A1(n) | $A_i = L_i * R$ <br> Class A1(n) | | | $A_i = L_i * R_i$ <br> Class A1(n) | $A_{ij} = L_i * R_{ij}$ <br> Class A2(n,m) |
| | A2(p,n') | $A_{ij} = L_{ij} * R$ <br> Class A2(p,n') | | | $A_{ij} = L_{ij} * R_j$ <br> if n' = n <br> Class A2(p,n) | $A_{ij} = L_{ij} * R_{ij}$ <br> if p = n, n' = m <br> Class A2(n,m) |

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

| Operator / | | S | M Family | | A Family | |
|---|---|---|---|---|---|---|
| | | | V(n) | M(n,n)‡ | A1(n) | A2(n,m) |
| S | | A = L/R<br>Class S | | $A_{ij} = L * I_{ij}$<br>Class M(n,n) | $A_i = L/R_i$<br>Class A1(n) | $A_{ij} = L/R_{ij}$<br>Class A2(n,m) |
| Class of Left Operand — M Family | V(n) | $A_i = L_i/R$<br>Class V(n) | | $A_i = \sum_k L_k * I_{ki}$<br>Class V(n) | | |
| | M(p,n) | $A_{ij} = L_{ij}/R$<br>Class M(p,n) | | $A_{ij} = \sum_k L_{ik} * I_{kj}$<br>Class M(p,n) | | |
| A Family | A1(n) | $A_i = L_i/R$<br>Class A1(n) | | | $A_i = L_i/R_i$<br>Class A1(n) | $A_{ij} = L_i/R_{ij}$<br>Class A2(n,m) |
| | A2(p,n') | $A_{ij} = L_{ij}/R$<br>Class F2(n') | | | $A_{ij} = L_{ij}/R_j$<br>if n' = n<br>Class A2(p,n) | $A_{ij} = L_{ij}/R_{ij}$<br>if p = n, n' = m<br>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 | | | |
|---|---|---|---|---|---|---|
| Operator | | | M Family | | A Family | |
| ** | | S | V(n) | M(m,n) | A1(n) | A2(n,m) |
| Class of Left Operand | S | $A = L^R$<br>Class S | | | $A_i = L^{R_i}$<br>Class A1(n) | $A_{ij} = L^{R_{ij}}$<br>Class A2(n,m) |
| M Family | V(m) | | $A_{ij} = L_i * R_j$<br>Class M(m,n)<br>outer product | | | |
| | M(m,n) | R th power of L<br>Class M(m,n) ‡ | | | | |
| A Family | A1(n) | $A_i = L_i^R$<br>Class A1(n) | | | $A_i = L_i^{R_i}$<br>Class A1(n) | $A_{ij} = L_i^{R_{ij}}$<br>Class A2(n,m) |
| | A2(p,n') | $A_{ij} = L_{ij}^R$<br>Class A2(p,n') | | | $A_{ij} = L_{ij}^{R_{ij}}$<br>n' = n<br>Class A2(p,n) | $A_{ij} = L_{ij}^{R_{ij}}$<br>p = n, n' = m<br>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 | | |
| EXP(X) | Exponent $e^X$ | Real $X \leqslant 170$, $\lvert \text{imag } X \rvert < 5 \times 10^6$ | |
| LOG(X) | Natural logarithm | $X \neq 0$ | |
| SIN(X) | Sine | $\lvert X \rvert < 10^{15}$ | |
| COS(X) | Cosine | | |
| TAN(X) | Tangent | | |
| COT(X) | Cotangent | | |
| ASIN(X) | Arc sine | $\lvert X \rvert \leqslant 1$ | Real X only |
| ACOS(X) | Arc cosine | | |
| ATAN(X) | Arc tangent | | |
| ACOT(X) | Arc cotangent | | |
| SINH(X) | Hyperbolic sine | $\lvert X \rvert < 170$ | |
| COSH(X) | Hyperbolic cosine | | |
| GAMMA(X) | Γ function | $10^{-50} < X < 56$ | |
| 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 member of the family of X |
| SUMSQROWS(X) | $\sum_j (X_{ij})^2$ | |
| PRODROWS(X) | $\prod_j X_{ij}$ | |
| SUMCOLS(X) | $\sum_i X_{ij}$ | Answer is a 1-dimensional member of the family of X |
| SUMSQCOLS(X) | $\sum_i (X_{ij})^2$ | |
| 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)<br>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) | $\int_{x_1}^{x_i}$ Fdx an integral with a variable upper limit | Numerical integration. (Answer is an array.) |
| TOTALINT(F:X) | $\int_{x_1}^{x_n}$ Fdx 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 ascending values |
| EIGENVECS(X) | Unitary matrix whose columns 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<br>Note: 1/X is also the inverse |
| TRACE(X) | Sum of the diagonal elements |
| TRANSPOSE(X) | Transpose of the object |

**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.<br>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)<br>or<br>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
INPUT...$                 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)
        -1.6056  5.6056  9

INPUT...     EIGENVECS(X)

       EIGENVECS(X)  (A 3 BY 3 MATRIX)
        -.8105     .098784  .57735
         .3197    -.7513    .57735
         .4908     .65252   .57735

INPUT...$
INPUT...$      THE EIGENVECTORS ARE THE COLUMNS OF THIS MATRIX.
INPUT...$      THE VECTOR CORRESPONDING TO THE SMALLEST 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 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...       ENDAUTOPRINT
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
INPUT...$                       FUNCTIONS
INPUT...$
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)
INPUT...      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...      DOMAIN COMPLEX
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...      DOMAIN REAL
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);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
```

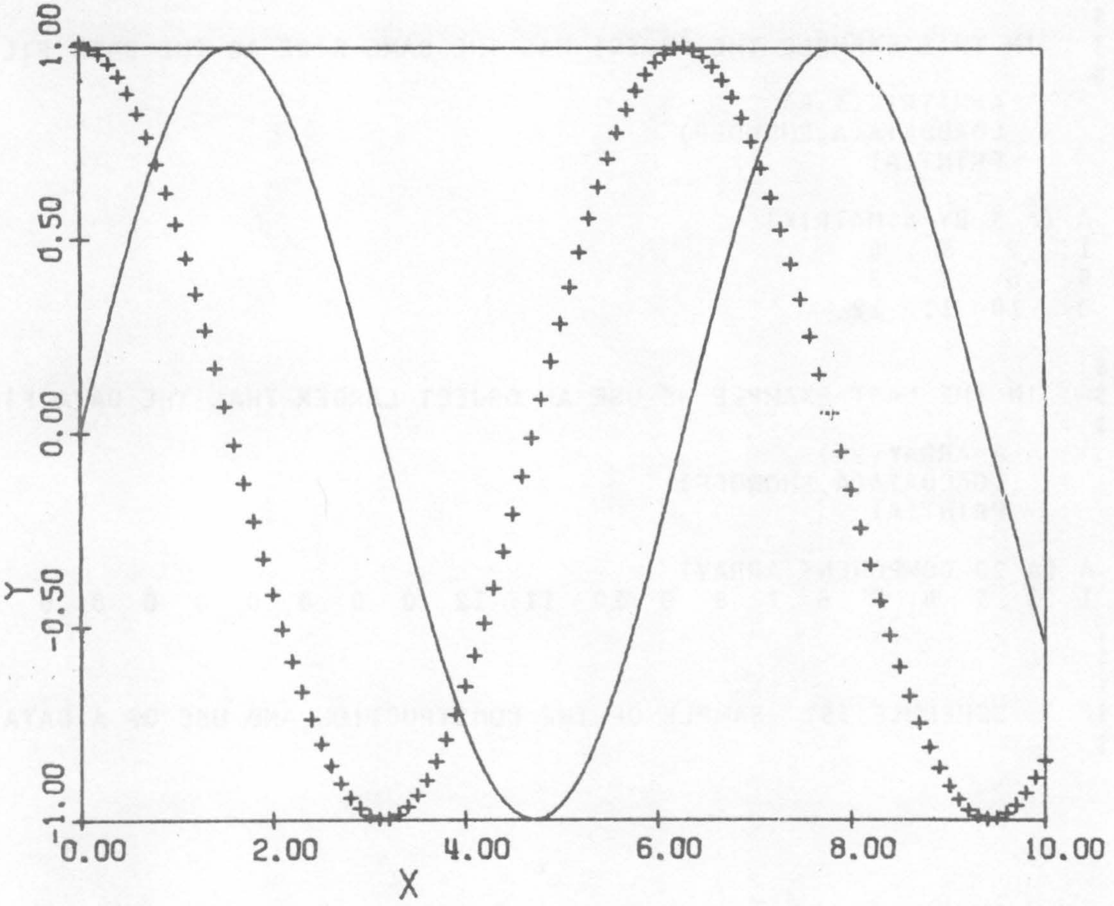
```
INPUT...$
INPUT...$         SCHEDULE 31.  SAMPLE TRANSFAMILY OPERATIONS
INPUT...$
INPUT...      X=ARRAY(:1,2,3)
INPUT...      X;VFAM(X)

         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 -.1874    .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 -1.16E-14  -1.157E-14 -1.157E-14   6.283
```

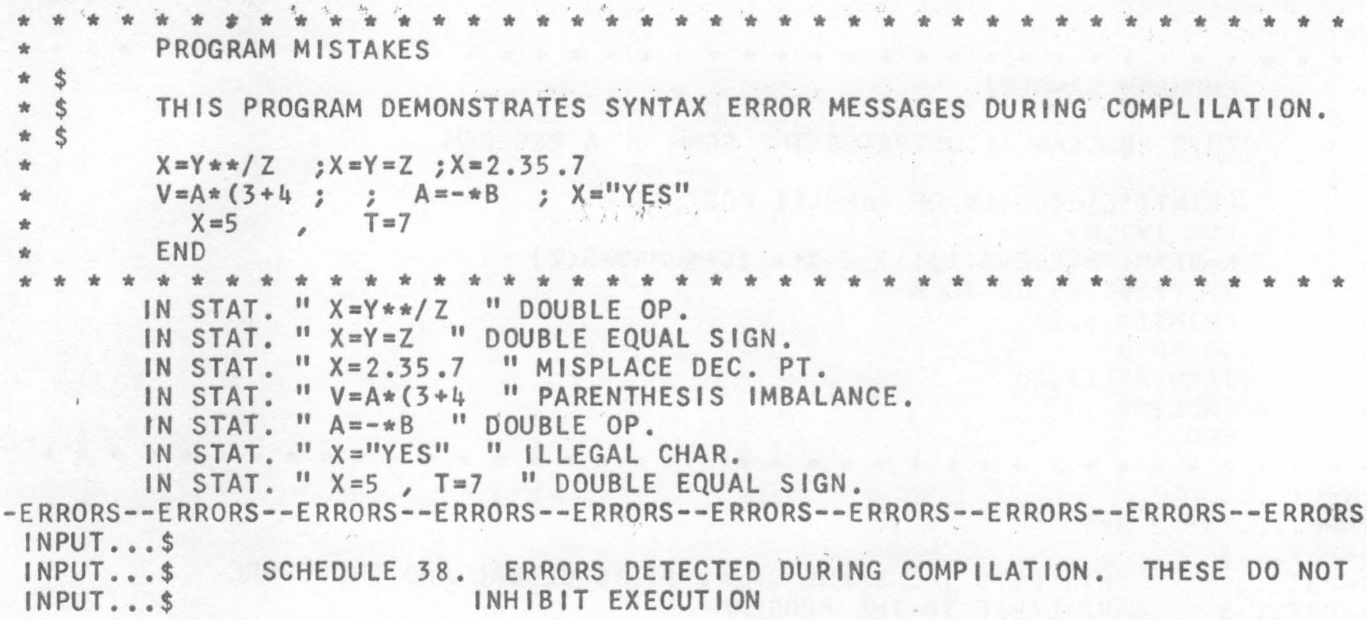
```
INPUT...$
INPUT...$       SCHEDULE 33.  SAMPLE OF A FUNCTION OF 2 VARIABLES.
INPUT...$
INPUT...      X=GRID(-1,1,.25); Y=GRID(0,2, .20)
INPUT...$
INPUT...$     FIRST CONSTRUCT A PAIR OF TWO DIMENSIONAL ARRAYS CONTAINING
INPUT...$     THE DESIRED VALUES OF X AND Y IN THEIR ROWS AND COLUMNS,
INPUT...$     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
       1.288  .9715  .7549  .6319  .5985  .6529  .7956  1.029  1.359
       1.575  1.38   1.26   1.201  1.197  1.243  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   2.436
       2.438  2.607  2.774  2.909  2.992  3.014  2.978  2.897  2.795
       2.726  3.016  3.279  3.479  3.591  3.605  3.523  3.364  3.154
       3.014  3.425  3.784  4.048  4.189  4.195  4.069  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   4.764  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
INPUT...$       PREVIOUS 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
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
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)
INPUT...       ENDDRAW(0)
```



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       PROGRAM SAMPLE1
* $
* $     THIS PROGRAM ILLUSTRATES THE FORM OF A PROGRAM
* $
*       PRINT('EXECUTION OF SAMPLE1 FOR',N)
*       FOR I=2,N
*       X=VFAM(INTEGERS(1,I));Z=X**X;S=SUMROWS(Z)
*       IF (I.GT.2) GO TO A
*       PRINT(X,Z,S)
*       GO TO B
*     A:TABULATE(X,S)
*     B:ENDLOOP I
*       END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
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

INPUT...$
INPUT...$     SCHEDULE 37.  A SAMPLE PROGRAM AND ITS EXECUTION
```



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       PROGRAM MISTAKES
* $
* $     THIS PROGRAM DEMONSTRATES SYNTAX ERROR MESSAGES DURING COMPILATION.
* $
*       X=Y**/Z  ;X=Y=Z ;X=2.35.7
*       V=A*(3+4 ;  ;  A=-*B  ; X="YES"
*         X=5   ,    T=7
*       END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
       IN STAT. " X=Y**/Z  " DOUBLE OP.
       IN STAT. " X=Y=Z  " DOUBLE EQUAL SIGN.
       IN STAT. " X=2.35.7  " MISPLACE 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 EQUAL SIGN.
-ERRORS--ERRORS--ERRORS--ERRORS--ERRORS--ERRORS--ERRORS--ERRORS--ERRORS--ERRORS
 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)
       IN STAT. " X=SQRT(-2)  " ENTERED COMPLEX DOMAIN.
```

# PART TWO

## SPEAKEASY-3: The SPEAKEASY System

by

S. Cohen

# I. INTRODUCTION

The capabilities of the current (1972) production versions of SPEAKEASY are greatly enhanced over those available at the time Part One was written (1968). From the viewpoint of the users, the growth has been in the direction of increases in the available facilities and an enlarged vocabulary. Actually there have been several major revisions in the structure of the processor during this period, but none of these have been directly apparent to the user community.

This report is intended to formally describe some of the features now available. In part this means describing the new words in the language. A section of this Part therefore deals with increases in the vocabulary. As will be seen, many of the features of the language now reflect the coordination of broad-based library facilities with the processor. The evolution of SPEAKEASY into a library-directed processor has had profound impact on its growth. Part of this report therefore deals with the library facilities now available and discusses their importance.

No new computational capability is apparent in the increased vocabulary described here; such improvements are now added to the so-called LINKULE libraries. These libraries of compiled FORTRAN subroutines are the means by which all new computational features and most other features are added to the processors. The LINKULES are discussed only briefly here. Part Three deals specifically with their capabilities.

Other libraries of the system, such as the documentation libraries and the libraries of stored SPEAKEASY decks, are dealt with in general. Specific information about the contents of these libraries can be obtained by use of the features that are available in the language and are described in this Part.

A change in the specifications of the SPEAKEASY language is being introduced in Sec. IV. This modification of the notation for logical and relational operators eliminates some previously restricted words and therefore will benefit all users. During a long transition period in which either notation can be used, existing programs can be converted to the new specifications. This conversion is not likely to be difficult, however, since the change is being introduced to eliminate conflicts that only a few users have encountered.

The use of SPEAKEASY in interactive sessions is growing in popularity now that TSO (the Time Sharing Option) is available to many institutions. Section V therefore explains the use of the facilities of this language that are particularly adapted to this operation and also describes the EDIT mode of operation.

The various processors currently available at Argonne are described. The choice of the default parameters for each processor is detailed. This serves both as a guide to the users of the particular processors and as an indication of the possible choice for other installations.

## II. NEW OPERATIONAL FEATURES

This chapter describes some of the more important words and concepts that have been added to SPEAKEASY since its original documentation. In writing SPEAKEASY documentation, the general approach has always been to describe the particular operation clearly and concisely. The operation is then illustrated by using the actual processor to display the effect. It is felt that this form of documentation lends itself to development of the language and provides users with some direct contact with the applications of the language.

As with all documentations, it is difficult to describe an operation concisely and at the same time indicate in detail the applications to which that operation is particularly suited. This report makes no attempt to do the latter; a separate report illustrating applications of the language and some of the tricks that have been developed is planned.

Each of the sections that follow is headed by a major keyword now available in SPEAKEASY. The operation of that word is described, and the section concludes with a sample of the use of the facility in an illustrative run with one of the currently available processors. In most cases the normal batch version is used because the format clearly labels the input.

Many of the words described in this section are members of the LINKULE library of the system. Some are at present built into the basic processor. Most words now are gradually being shifted from the processor to the libraries. Users of SPEAKEASY should be unaware of such changes except when specific inquiries are made about the contents of the libraries.

## A. SIZE

SPEAKEASY has a built-in dynamic-storage allocator that carefully manages the space available for information defined during execution. The SPEAKEASY processors themselves are designed so that parts are dynamically brought into the computer as dictated by demands of the particular application program. Unfortunately, it is not yet possible to automatically divide the space available in the computer between these two uses. While default sizes for each of the above are normally supplied, it is desirable that the space allocated to each be based on the actual need.

The first card in the SPEAKEASY deck can be used to inform the processor of the amount of data space needed in the run. The rest of the allocated space for the job is then available for other uses. This first card will be interpreted as a size-selecting card if it carries any of the following words: SIZE, MAIN, LCS, or AUTOCORE. If none of these occur, the default size selection for that particular processor is assumed and the first card is interpreted as a normal SPEAKEASY statement.

If the size of data storage is being specified, the card should read

SIZE = n
SIZE = n, MAIN
SIZE = n, LCS
or AUTOCORE

Here n is the number of kilobytes of storage to be set aside for SPEAKEASY data (1 kilobyte is approximately 120 user-defined numbers). For most applications the card will read

SIZE = n

For machines with LCS (large-core storage) the data area may be placed in LCS by including the word LCS in the size selection card. Alternatively, in such machines the selection of LCS space can be automatically determined by the space allocation on the job card by use of the word AUTOCORE. This word means that the largest amount of LCS that is available is to be used for this job. The word MAIN is a default if SIZE is specified and has been included here only for completeness of notation.

Initially users will not know how large a data area to assign to a specific job. The default values should be used in such cases (i.e., the user should omit the size-selection card entirely).

At the completion of each job, the last line of oucput gives three pieces of information: the current amount of space being used (labeled NOW), the peak amount needed (labeled PEAK), and the amount of space allocated (labeled ALLOCATED). These numbers are the approximate numbers of kilobyte involved. The next run of this same job should be given a SIZE specification some 10% larger than the indicated peak value. (The extra space provides for some increase in efficiency of operation.)

Part of some actual SPEAKEASY runs are shown below. These indicate the use of SIZE and shows the form of the final line of output.

```
         SPEAKEASY 3D BETA  7:55 PM  7/20/72
 INPUT...SIZE=10
 INPUT...MARGINS(1,80)
 INPUT...X=MATRIX(5,5:INTEGERS(1,25))
 INPUT...Y=X*X
 INPUT...PRINT Y

   Y (A 5 BY 5 MATRIX)
   215   230   245   260   275
   490   530   570   610   650
   765   830   895   960   1025
   1040  1130  1220  1310  1400
   1315  1430  1545  1660  1775
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  CORE USED  1 K NOW,  1 K PEAK,      ALLOCATED  10 K
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         SPEAKEASY 3D BETA  8:00 PM  7/20/72
 INPUT...SIZE=15
 INPUT...MARGINS(1,80)
 INPUT...X=GRID(0,10)
 INPUT...Y=SIN(X)*COS(X)+X*EXP(-X)+2*X*X
 INPUT...AVERAGE(Y)
 AVERAGE(Y)  =  67.116
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  CORE USED  2 K NOW,  5 K PEAK,      ALLOCATED  16 K
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

## B. VOCABULARY

The single word VOCABULARY asks the processor to list the currently defined words in its vocabulary. Most of the words are described in Part One or are added synonyms for these words. Others are described in this Part. The rest have not been documented except internally in the system. A brief description of any of these words can be obtained by use of the special word HELP (described in the next subsection).

Changes in response to the word VOCABULARY can be expected. Eventually it is planned that the combination of this word and the word HELP will be the means by which users are led to all of the available information in the system. One form of the response given on the next page is typical of current processors.

```
:_vocabulary
   _         _         _         _         _         _         _
   ABS       COSH      FOR       LENGTH    MYPROCS   RANKER    SUMCOLS
   ACCURACY  COT       FRAC      LIBINDEX  NAMES     RATIONAL  SUMPROD
   ACOS      CREATE    FRACPART  LIBNAMES  NE        READ      SUMROWS
   ACOT      CREATEME  FREE      LIBRARIE  NEUMANN   REAL      SUMS
   ADDGRAPH  CUMPROD   GAMMA     LIBRARYN  NEWGRAPH  REALPART  SUMSQ
   ADJOINT   CUMSUM    GE        LINKLIB   NEWPAGE   REAL4     SUMSQCOL
   AFAM      DATA      GEIGEN    LINKULES  NEWS      REAL8     SUMSQROW
   AMAT      DEBUGGIN  GO        LIST      NOCOLS    RECLASS   SYMBOLS
   AND       DEC       GOTO      LISTHEAD  NOECHO    RESTRICT  SYMMAT
   ANGLES    DEFINEA1  GRAPH     LISTMEMB  NOELS     RESUME    TABULATE
   ARRAY     DEFINEA2  GRAPHICS  LISTPROG  NORATION  RETURN    TAN
   ARRAYS1   DELETE    GRID      LOADDATA  NOROOTS   ROOTS     TIME
   ARRAYS2   DERIV     GT        LOC       NOROWS    ROWARRAY  TOTALINT
   ARRAY2D   DERIVATI  HELP      LOCMAX    NOT       ROWMAT    TOTINT
   ASIN      DET       HENCEFOR  LOCMIN    NOZEROS   ROWMAX    TRACE
   ASYMMAT   DIAGELS   HIERARCH  LOCS      NUMBERS   ROWMIN    TRANSFAM
   ATAN      DIAGMAT   HIGHWIDE  LOG       OBJECT    RUN       TRANSP
   AUTOCORE  DMAT      HIWIDE    LOGGAMMA  OBJECTS   SELECT    TRANSPOS
   AUTOPRIN  DOCUMENT  HLABEL    LOGIC     OMITCLAS  SETGAUSS  TRIG
   AUTOTAB   DOMAIN    HSCALE    LOWERTRI  ONEDIMFU  SETINFIN  TUTORIAL
   AVERAGE   DONTLIST  HSIZE     LT        ONERROR   SETJACOB  TWODIMFU
   A1D       DOUBLEFA  IF        MARGINS   OR        SETLAGUE  UMAT
   A2D       DUMP      IMAG      MAT       ORDERED   SETLEGEN  UNITMAT
   BESSEL    ECHO      IMAGPART  MATH      ORDERER   SETLIB    UPPERTRI
   BESSELK   EDIT      INOUT     MATRICES  OTHERS    SETNULL   USE
   BUGS      EDITMODE  INPUT     MATRIX    OUTPUT    SETPLOT   USEMEMBE
   CGAMMA    EIGENSYS  INPUTS    MATRIXDE  PAUSE     SIGN      VARIABLE
   CLEAR     EIGENVAL  INSERT    MATRIXOP  PLOTSYMB  SIGNIFIC  VEC
   CLEARDAT  EIGENVEC  INTEG     MAX       PLOTTITL  SIMEQ     VECTOR
   COLARRAY  ELEMENTA  INTEGERS  MAXOFCOL  PRINT     SIN       VECTORDE
   COLMAT    ELLIPE    INTEGRAL  MAXOFROW  PRINTCLA  SINGLEVA  VECTORS
   COLMAX    ELLIPK    INTEGRAT  MELD      PROCLIB   SINH      VERSIONS
   COLMIN    END       INTERP    MFAM      PROD      SIZE      VFAM
   COLWIDTH  ENDAUTOP  INTERPOL  MIN       PRODCOLS  SMAT      VLABEL
   COMMANDS  ENDLOOP   INTPART   MINOFCOL  PRODROWS  SORT      VOCABULA
   COMPILE   EONE      INTS      MINOFROW  PRODUCTS  SPACE     VSCALE
   CONJ      EQ        INT2      MISCELLA  PROGRAM   SPECIAL   VSIZE
   CONJUGAT  ERF       INT4      MOVE      PROGRAMM  SPHBES    WHERE
   CONSTRAI  ERFC      INVERSE   MYDOCS    PROGRAMS  SPHBESN   WHEREVER
   CONTINUE  ERRORS    KEEP      MYHELP    PUNCH     SQRT      WHOLE
   CONVERT   EXECUTE   KEPT      MYKEEP    QUIT      STOP      WRITE
   COPY      EXP       LABEL     MYKEPT    RANDOM    STRUCTUR  ZEROS
   COS       FIN       LE        MYLINKS   RANKED    SUM
```

## C. HELP

The vocabulary of SPEAKEASY is constantly being augmented. While documentation such as this report can be used to communicate new features, it is highly desirable to provide a less formal and more rapidly altered means of describing specific words in the language. In interactive usage, it is also extremely important that both the vocabulary and the documentation for each word be easily accessible.

The basic vocabulary of the processor is printed out by the command VOCABULARY described previously. Each word in this vocabulary and other words in the language are described in a concise operational form and are available on demand by use of the key word HELP. The input statement

```
HELP xxxx
```

requests that the processor print out a brief description of the word xxxx. The documents available in this manner are not restricted to those listed in the vocabulary but are actually a separate document library that is appended to the various processors (see Part Four). By using a standard sequence of SPEAKEASY statements, one can obtain a catalog of this library. That is, the names of the documents available by use of the word HELP will be printed in response to the input

```
CATALOG = LIBINDEX (HELP)
TABULATE (CATALOG)
```

The entire set of documents may be printed out by use of the following SPEAKEASY program:

```
PROGRAM LISTHELP
CATALOG = LIBINDEX (HELP)
FOR I = 1, NOELS (CATALOG)
SPACE(2)
```

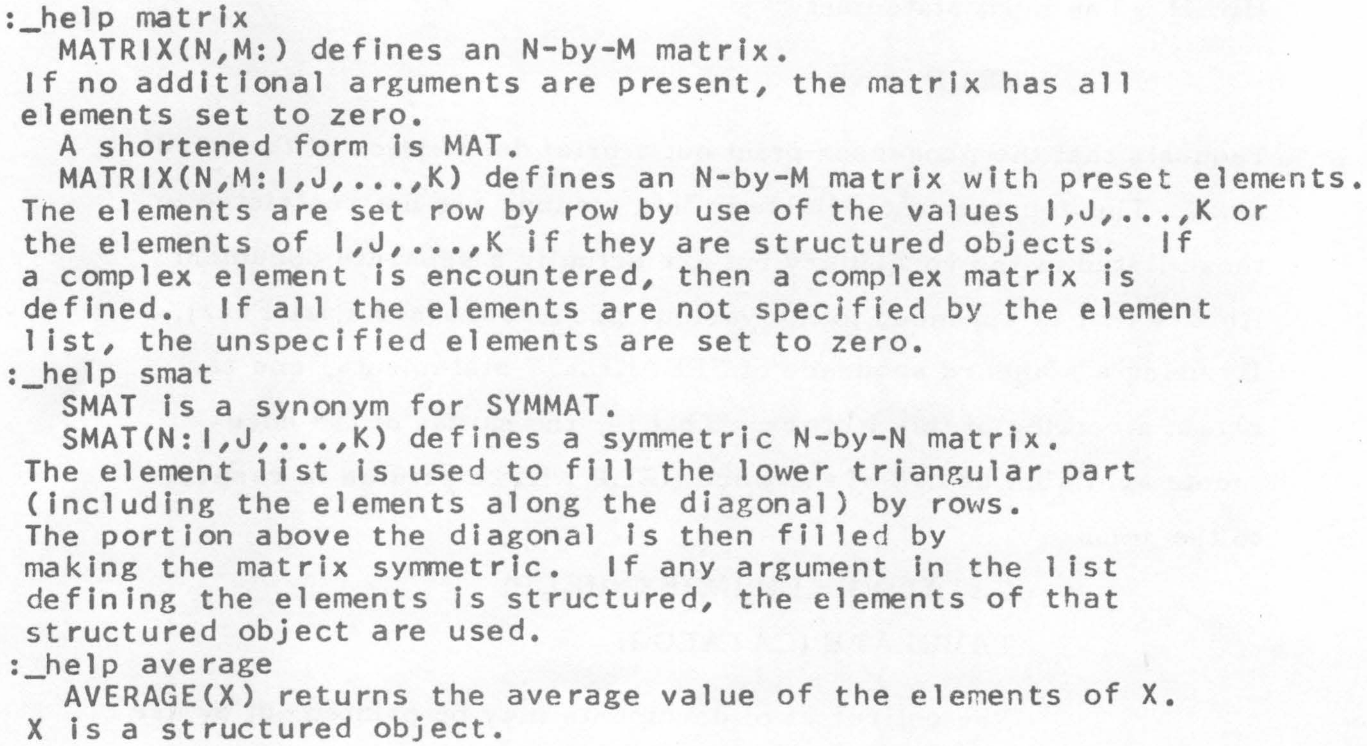
```
HELP  OBJECT  (CATALOG(I))
ENDLOOP  I
END
EXECUTE  LISTHELP
```

Although all HELP documents could be printed in this way this would be a time consuming and expensive means of obtaining such a listing. Part Four of this report contains a complete printout of the documents available at this time.

Information necessary to make use of the HELP processor itself is available by the statement

```
HELP.
```

A few of the actual HELP documents are shown below.

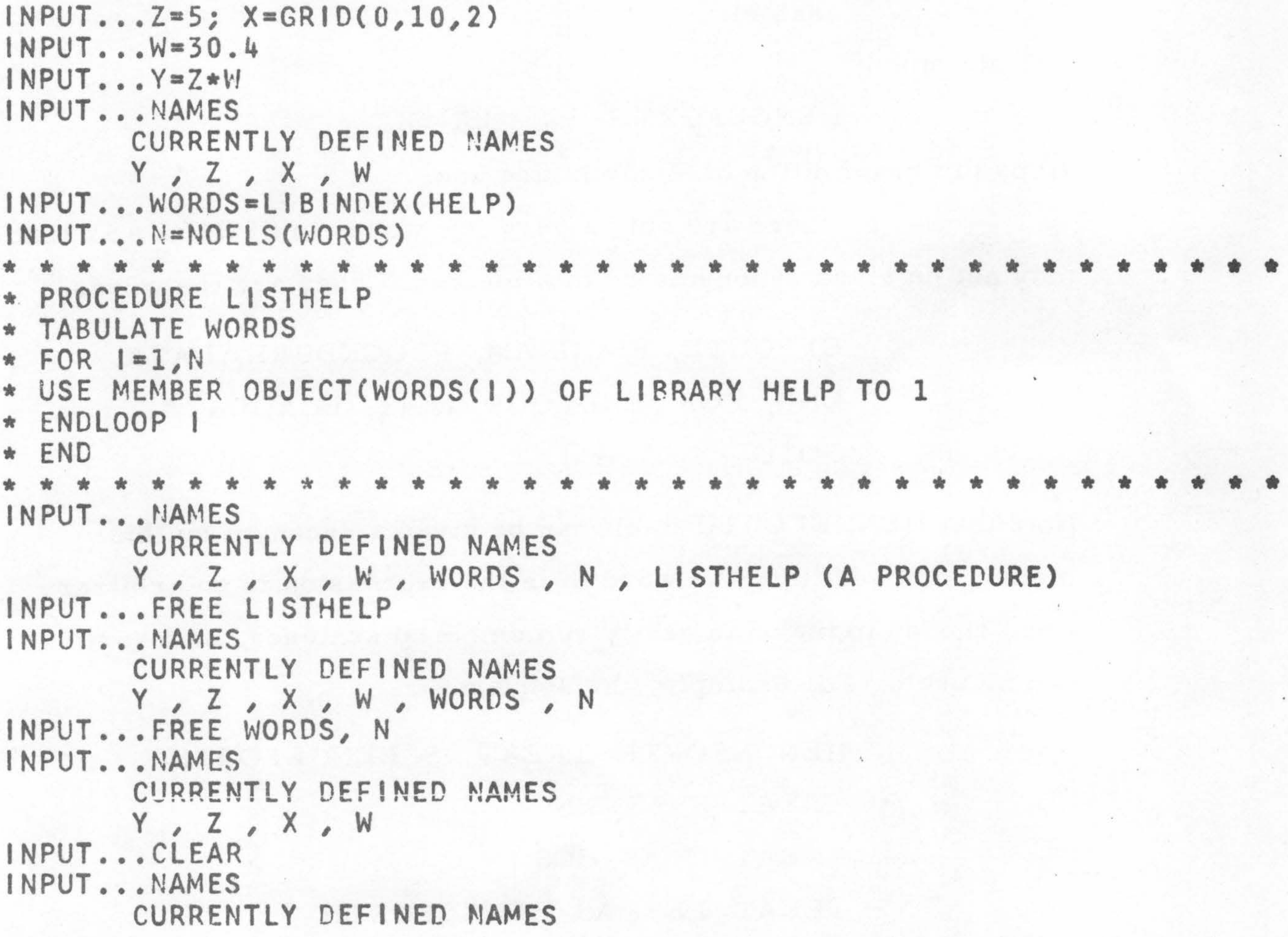


```
:_help matrix
   MATRIX(N,M:) defines an N-by-M matrix.
 If no additional arguments are present, the matrix has all
 elements set to zero.
   A shortened form is MAT.
   MATRIX(N,M:I,J,...,K) defines an N-by-M matrix with preset elements.
 The elements are set row by row by use of the values I,J,...,K or
 the elements of I,J,...,K if they are structured objects.  If
 a complex element is encountered, then a complex matrix is
 defined.  If all the elements are not specified by the element
 list, the unspecified elements are set to zero.
:_help smat
   SMAT is a synonym for SYMMAT.
   SMAT(N:I,J,...,K) defines a symmetric N-by-N matrix.
 The element list is used to fill the lower triangular part
 (including the elements along the diagonal) by rows.
 The portion above the diagonal is then filled by
 making the matrix symmetric.  If any argument in the list
 defining the elements is structured, the elements of that
 structured object are used.
:_help average
   AVERAGE(X) returns the average value of the elements of X.
 X is a structured object.
```

## D. NAMES

The single word NAMES requests the processor to print out the names of all of the currently defined SPEAKEASY objects. This is extremely useful in the interactive mode since it provides the user with the means of recollecting the names of variables that he has previously defined. Combined with the implied print statement, the user can quickly examine any currently defined object.

A section of programming in which objects are defined and freed is shown below. Between each such command the processor was requested to display the defined names.



```
INPUT...Z=5; X=GRID(0,10,2)
INPUT...W=30.4
INPUT...Y=Z*W
INPUT...NAMES
      CURRENTLY DEFINED NAMES
      Y , Z , X , W
INPUT...WORDS=LIBINDEX(HELP)
INPUT...N=NOELS(WORDS)
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* PROCEDURE LISTHELP
* TABULATE WORDS
* FOR I=1,N
* USE MEMBER OBJECT(WORDS(I)) OF LIBRARY HELP TO 1
* ENDLOOP I
* END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INPUT...NAMES
      CURRENTLY DEFINED NAMES
      Y , Z , X , W , WORDS , N , LISTHELP (A PROCEDURE)
INPUT...FREE LISTHELP
INPUT...NAMES
      CURRENTLY DEFINED NAMES
      Y , Z , X , W , WORDS , N
INPUT...FREE WORDS, N
INPUT...NAMES
      CURRENTLY DEFINED NAMES
      Y , Z , X , W
INPUT...CLEAR
INPUT...NAMES
      CURRENTLY DEFINED NAMES
```

## E. HENCEFORTH

Although every effort has been made to choose the names of SPEAKEASY functions with care, it is often desirable to provide alternatives. Previously this was done by adding synonyms (for example MAT is equivalent to MATRIX, VEC is equivalent to VECTOR, etc.). A more flexible facility has now been provided to enable users to define their own synonyms at execution time. The statement

```
HENCEFØRTH   X   IS   Y
```

means that:

> Anytime hereafter, if the word X is encountered treat that word as if it were the word Y (the word Y is still usable).

The statement

```
HENCEFØRTH   X   IS   X
```

stops the redefinition of X for future use.

There are only a very few words in SPEAKEASY that may not be given synonyms by this method. They are the words

> EXECUTE, PROGRAM, PROCEDURE, DATA,
> END, FOR, ENDLOOP, QUIT, LOADDATA,
> GOTO

Note that HENCEFORTH itself can be given a synonym by this method. In addition, the word IS in the expression is an arbitrary word chosen to make an easily remembered sentence, any word can be used. For example, the sequence

```
HENCEFORTH TREAT IS HENCEFORTH
TREAT  S  AS  SIN
TREAT  C  AS  COS
TREAT  LET  AS  HENCEFORTH
LET  M  BE  MATRIX
```

is acceptable. The statement

HENCEFORTH X IS 'SIN(X)'

is usable with the restriction that the literal quantity can have at most eight letters. Any of the normal SPEAKEASY character set except an apostrophe is allowed. This makes it possible to obtain output that is particularly neat.

HENCEFORTH is a convenient way of compacting a user's program by eliminating repetitious typing of long words. It is particularly important when combined with the operation OBJECT described later in this document.

The use of HENCEFORTH is illustrated below.

```
INPUT...HENCEFORTH M IS MATRIX
INPUT...Y=M(2,2:1,5,8 4)
INPUT...PRINT Y
  Y (A 2 BY 2 MATRIX)
  1  5
  8  4
INPUT...HENCEFORTH TREAT IS HENCEFORTH
INPUT...TREAT P AS PRINT
INPUT...TREAT LET AS TREAT
INPUT...LET V BE VECTOR
INPUT...W=V(4:1,2,3,4)
INPUT...P W
  W (A VECTOR WITH 4 COMPONENTS)
  1  2  3  4
```

## F. OBJECT

It is often desirable to be able to generate the names of objects. This is particularly true when defining large numbers of objects, each of which is dependent on particular parameters of a calculation. A general operation for generating names has made available.

It should be remembered that names in SPEAKEASY can have at most 8 characters. This restriction holds for names that are generated by the mechanism described here. Other restrictions such as the use of special characters do not exist since the names are generated internally in the processor.

The expression

OBJECT(I, J, K, · · ·)

may occur anywhere in a SPEAKEASY statement. It means that the arguments I, J, K, etc. are to be used to generate a name that is to be substituted for this expression.

Each of the arguments can be either a literal quantity or a non-negative number. For literals, the expression itself is used. For numbers, the integer part of the number is used as a literal. The various arguments are then joined together to make up the name. Thus

```
OBJECT ('A', 3, 'B')  becomes  A3B
OBJECT ('A', 3, 'X', 4)  is  A3X4
```

If X = 'XX'; Y = 'Y'; I = 4 then

```
OBJECT (X, I, 3 Y)  becomes  XX43Y
```

The use of literal quantities in expressions means that strange looking names can be generated. Thus if X = '*4' then

```
OBJECT (X, X)  becomes  *4*4
```

The power provided by the use of the word OBJECT is greatly enhanced when combined with the word HENCEFORTH. The expression

HENCEFORTH X IS OBJECT ('X',I)

means that the variables X1, X2, X3, etc. are specified when the current value of the variable I is 1, 2, 3, etc. The following series of statements illustrates one of the uses of the feature to obtain a rather elegant printout which would not otherwise be possible.

```
INPUT...PI=2 ACOS(0);X=GRID(0,2 PI,PI/40)
INPUT...N=3;HENCEFORTH Y IS OBJECT('SIN(',N,'X)')
INPUT...Y=SIN(N X)
INPUT...TABULATE(X,Y)

  X        SIN(3X)          X        SIN(3X)          X        SIN(3X)

  0        0                2.1206  .078459           4.2412  .15643
  .07854  .23345            2.1991  .30902            4.3197  .38268
  .15708  .45399            2.2777  .5225             4.3982  .58779
  .23562  .64945            2.3562  .70711            4.4768  .76041
  .31416  .80902            2.4347  .85264            4.5553  .89101
  .3927   .92388            2.5133  .95106            4.6338  .97237
  .47124  .98769            2.5918  .99692            4.7124  1
  .54978  .99692            2.6704  .98769            4.7909  .97237
  .62832  .95106            2.7489  .92388            4.8695  .89101
  .70686  .85264            2.8274  .80902            4.948   .76041
  .7854   .70711            2.906   .64945            5.0265  .58779
  .86394  .5225             2.9845  .45399            5.1051  .38268
  .94248  .30902            3.0631  .23345            5.1836  .15643
  1.021   .078459           3.1416  0                 5.2622 -.078459
  1.0996 -.15643            3.2201 -.23345            5.3407 -.30902
  1.1781 -.38268            3.2987 -.45399            5.4192 -.5225
  1.2566 -.58779            3.3772 -.64945            5.4978 -.70711
  1.3352 -.76041            3.4558 -.80902            5.5763 -.85264
  1.4137 -.89101            3.5343 -.92388            5.6549 -.95106
  1.4923 -.97237            3.6128 -.98769            5.7334 -.99692
  1.5708 -1                 3.6914 -.99692            5.8119 -.98769
  1.6493 -.97237            3.7699 -.95106            5.8905 -.92388
  1.7279 -.89101            3.8485 -.85264            5.969  -.80902
  1.8064 -.76041            3.927  -.70711            6.0476 -.64945
  1.885  -.58779            4.0055 -.5225             6.1261 -.45399
  1.9635 -.38268            4.0841 -.30902            6.2046 -.23345
  2.042  -.15643            4.1626 -.078459           6.2832  0
```

## G. MELD

Some SPEAKEASY programs include a sequence such as:

```
FOR  I = 1, NOELS(X)
FOR  J = 1, NOELS(Y)
F(I, J) = some function of X(I) and Y(J)
ENDLOOP J
ENDLOOP I
```

Such looping, though logically correct, defeats many of the optimizing features of the language. Users have been warned of the consequence of doing element by element operations that do not make use of the built-in algebra for structured objects. For a pair of nested loops as illustrated, the array algebra of the language is usually sufficient to enable one to eliminate the loops easily.

The problem of multidimensional arrays with more than two independent variables is not easily dealt with. For this purpose, a new concept and a new word has been added to the SPEAKEASY language. Both the word and the concept are due to Richard Kimmel. The word MELD provides a major advance in the capabilities of the language.

Although MELD is a straightforward operator, it differs from others that have previously been met in SPEAKEASY since it redefines the arguments occurring in the statement. Thus

```
MELD(I, J, K)
```

in fact alters the structure of I, J, and K.

The arguments in the MELD must be 1-dimensional objects. In essence, the MELD operation is a simple and direct means of providing a revised set of objects in which every element of each of the objects in the argument list is associated with every element of every other object in that list. In fact, as will be seen, this is a simple way of describing a multidimensional space. A few examples are sufficient to show how the operator acts. (It is much easier to show the operation than to describe it in words.) If I is a 2-component 1-dimensional array and J is a 3-component array, e.g., if

$$I = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \qquad J = \begin{pmatrix} 0 \\ 5 \\ 7 \end{pmatrix},$$

then MELD (I, J) alters both I and J and produces two 6-component arrays

$$I = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \end{pmatrix}, \qquad J = \begin{pmatrix} 0 \\ 5 \\ 7 \\ 0 \\ 5 \\ 7 \end{pmatrix}.$$

Similarly if I, J, and K are

$$I = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad J = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}, \quad K = \begin{pmatrix} 8 \\ 9 \end{pmatrix},$$

then MELD (I, J, K) redefines them all to be the 12-component arrays

$$I = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad J = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 4 \\ 4 \\ 1 \\ 1 \\ 2 \\ 2 \\ 4 \\ 4 \end{pmatrix}, \quad K = \begin{pmatrix} 8 \\ 9 \\ 8 \\ 9 \\ 8 \\ 9 \\ 8 \\ 9 \\ 8 \\ 9 \\ 8 \\ 9 \end{pmatrix}.$$

Note that after melding, as in the last example, any function of I, J, and K can be written in a straightforward manner. Thus

```
F = 3*I + 2 * (I + J) - K * I * SIN (PI * K)
```

is allowed. The function is then effectively evaluated in the 3-dimensional space spanned by the three original arrays. A table of the values of I, J, K, and the resultant could be obtained by

```
TABULATE (I, J, K, F)
```

Up to ten arguments are permitted in a single MELD call. Care should be taken not to produce unreasonably large arrays by this method. For example, if ten arrays of only 3 components each are melded, then each of the resultant arrays would have $3^{10}$ elements. This would, in fact, far exceed the capacity of any available SPEAKEASY processor.

Melding combined with the use of structured indices leads to other capabilities that are often needed. If one melds the indices of an array rather than its elements, then it is possible to carry out later operations on this and correlated arrays.

The use of the MELD operation is shown below. The evaluation of the function W as a function of the three independent variables I, J and K is carried out by means of melding.

```
INPUT...I=ARRAY(5:1 2 3 4,5);J=ARRAY(3:7,8,9);K=ARRAY(2:0,6)
INPUT...TABULATE(I,J,K)
  I  J  K
  1  7  0
  2  8  6
  3  9
  4
  5
INPUT...MELD(I,J,K)
INPUT...TABULATE(I,J,K)
  I  J  K      I  J  K
  1  7  0      3  8  6
  1  7  6      3  9  0
  1  8  0      3  9  6
  1  8  6      4  7  0
  1  9  0      4  7  6
  1  9  6      4  8  0
  2  7  0      4  8  6
  2  7  6      4  9  0
  2  8  0      4  9  6
  2  8  6      5  7  0
  2  9  0      5  7  6
  2  9  6      5  8  0
  3  7  0      5  8  6
  3  7  6      5  9  0
  3  8  0      5  9  6
INPUT...W=3*I+4/J-K
INPUT...TABULATE(I,J,K,W)
  I  J  K  W               I  J  K  W
  1  7  0  3.5714          3  8  6  3.5
  1  7  6 -2.4286          3  9  0  9.4444
  1  8  0  3.5             3  9  6  3.4444
  1  8  6 -2.5             4  7  0  12.571
  1  9  0  3.4444          4  7  6  6.5714
  1  9  6 -2.5556          4  8  0  12.5
  2  7  0  6.5714          4  8  6  6.5
  2  7  6  .57143          4  9  0  12.444
  2  8  0  6.5             4  9  6  6.4444
  2  8  6  .5              5  7  0  15.571
  2  9  0  6.4444          5  7  6  9.5714
  2  9  6  .44444          5  8  0  15.5
  3  7  0  9.5714          5  8  6  9.5
  3  7  6  3.5714          5  9  0  15.444
  3  8  0  9.5             5  9  6  9.4444
```

## H. CONSTRAIN/SELECT

As has been mentioned, MELD produces correlated elements in several objects. It is often then necessary to make selections based on certain constraints between the elements of these objects. An operation is provided to carry out such selections for all of the objects and other correlated objects simultaneously. For example, the command

```
CONSTRAIN (A, B, C, D : (A + B.GT.C) .AND. B.NE.C)
```

means that the logical expression to the right of the colon is to be constructed. The objects to the left of the colon are truncated to leave only elements corresponding to true values in the logical expression. For example, if X, Y, and Z describe positions in 3-space, the statement

```
CONSTRAIN (X, Y, Z:X**2+Y**2+Z**2. LT. R**2)
```

would eliminate any point described by $X_i$, $Y_i$, and $Z_i$ lying outside the sphere of radius R. Note that all arguments must be 1-dimensional or scalar, and all 1-dimensional objects must have the same length.

SELECT is similar to CONSTRAIN. The effect of the operator is to truncate (or expand) several correlated 1-dimensional objects by use of a single structured index as the control array.

Thus if

$$I = \begin{pmatrix} 1 \\ 3 \\ 5 \\ 7 \end{pmatrix}$$

and A, B, and C are all 1-dimensional arrays, each with at least seven components, then

```
SELECT (A, B, C : I)
```

redefines A, B, C to be 4-component objects with the 1st, 3rd, 5th, and 7th elements of the original objects. Note that in this operation A, B, and C need not be the same length.

Samples of the operations CONSTRAIN and SELECT are shown below.

```
INPUT...I=INTEGERS(3,10)
INPUT...J=INTEGERS(20,13)
INPUT...L= 1 3 4
INPUT...TABULATE I J
  I   J
  3   20
  4   19
  5   18
  6   17
  7   16
  8   15
  9   14
  10  13
INPUT...SELECT(I,J:L)
INPUT...TABULATE I J
  I  J
  3  20
  5  18
  6  17
```

```
INPUT...X=GRID(0,1,.1);Y=X;MELD(Y,X)
INPUT...CONSTRAIN(X Y:X**2+2*Y**2 LT .5)
INPUT...TABULATE X Y
  X    Y         X   Y          X   Y
  0    0         .4  .1         .2  .3
  .1   0         .5  .1         .3  .3
  .2   0         .6  .1         .4  .3
  .3   0         0   .2         .5  .3
  .4   0         .1  .2         0   .4
  .5   0         .2  .2         .1  .4
  .6   0         .3  .2         .2  .4
  .7   0         .4  .2         .3  .4
  0    .1        .5  .2         .4  .4
  .1   .1        .6  .2         0   .5
  .2   .1        0   .3
  .3   .1        .1  .3
```

## I. ECHO/NOECHO

One feature of SPEAKEASY that has proved desirable in batch operation is that of printing the actual input information along with the results. This "echoing" of the input means that the user can clearly see what it was that he asked and can see the response immediately below it.

In other applications, particularly in interactive usage, repeating the input information would be redundant since it would follow immediately below the typed line. The ability to control the echoing therefore has been added to the language. If the statement NOECHO is encountered during execution, the echoing of input data is suppressed. ECHO reinstates it.

One application of this feature in normal batch jobs is to produce more publishable results. Extraneous commands to the processor can be done in the NOECHO mode without their operations appearing on the output.

## J. LISTPROG/DONTLIST

The listing of SPEAKEASY programs can be controlled in much the same way as that of the ECHO facility in the MANUAL MODE. DONTLIST indicates that such programs should not be printed in the output. LISTPROG means that they should. If the options DONTLIST and NOECHO are selected, only the actual results produced during execution will be printed—as in the operation of conventional programs such as FORTRAN.

In the interactive operation, DONTLIST and NOECHO were selected as the preferred mode of operation. Batch processing, on the other hand, normally uses LISTPROG and ECHO.

## K. MARGINS

This is a facility that enables a user to control the width and position of his printed output. This is of particular importance when devices other than line printers are used for output. For instance a teletype has only 72 characters on a line and would not supply the output in the proper format for a printer.

The user specifies the left-hand and right-hand limits of the printout by executing a statement of the form

MARGINS (n, m)     where $n < m$

which means that output should be restricted to columns starting at n and ending with m. Alternatively

MARGINS (m)     is equivalent to     MARGINS (1, m).

All printed output normally has a so-called carriage-control character in its first position. This is not printed but is used to control vertical spacing and to position output at the top of a new page. In some applications it is necessary to suppress such control functions. The statement

MARGINS (0, m)     is equivalent to     MARGINS (1, m)

except that the carriage-control characters are eliminated.

The SPEAKEASY processor will readjust all of its printout to conform to the specified margins.

The next page shows how the word MARGINS can be used to control the format of output.

```
INPUT...A=AMAT(3:1,2,3)
INPUT...PRINT A

         A (A 3 BY 3 MATRIX)
         0 -1 -2
         1  0 -3
         2  3  0

INPUT...S=SMAT(3:1,2,3,4,5,6)
INPUT...MARGINS(20,40)
INPUT...PRINT S

                     S (A 3 BY 3 MATRIX)
                     1  2  4
                     2  3  5
                     4  5  6

INPUT...MARGINS(30,120)
INPUT...AS=A*S
INPUT...PRINT ('THE PRODUCT IS 'AS)
                             THE PRODUCT IS

                               AS (A 3 BY 3 MATRIX)
                              -10 -13 -17
                              -11 -13 -14
                               8  13  23
```

## III. LIBRARY FACILITIES

The SPEAKEASY-3 processors were developed with the intent of making extensive use of libraries that could be attached to the processors at execution time. Once the means of accessing such libraries was clearly defined, the processors and the libraries could be developed independently rather than in concert.

The consequences of the separation cannot be overemphasized. The processor developments have by and large been in the direction of providing very basic facilities; their objective is always a smaller executable module with as few specialized features as possible. This is in marked contrast to the developments in the libraries, in which as many new features as possible are sought. One wants the greatest possible capabilities in general, but for each specific application one wants to avoid the difficulties associated with such generality. Libraries answer both needs admirably.

Detached libraries have many additional direct advantages. Users can freely put information into private libraries without fear of altering the operations of the processors. Similarly, new words for the language can be tested and validated by use of standard processors without the usual problems associated with new releases. The transfer of information from private to communal libraries is not a major disturbance to the system. The growth of the over-all language since the introduction of attachable libraries has been dramatic. (Most of the words described in this Part are in the libraries of the system—some were in fact added to fill gaps in this writeup itself.)

The growth capabilities of the SPEAKEASY system now rests in the libraries attached to the processors. To a large extent, the processor can now be viewed as an interface between the various libraries as well as between the users and the libraries.

This section describes the various libraries in the system. To some extent these descriptions tend to overemphasize their distinctions. Information (i. e., computational techniques or data) can be entered into the system in a variety of ways. The choice of a particular library or combination of libraries for the storing of information is somewhat arbitrary and may reflect personal whims. The system is sufficiently broad-based to accept several alternatives. There should therefore be no need to be concerned about following rigid rules in selecting the mode of operation.

Actually there is a growing interconnection between the libraries that involve the SPEAKEASY processor only as a communication module. The exciting aspects of the system at this stage in the growth is in this interconnectivity. Each new feature added to the system enhances the over-all capabilities of the system, not only because of direct contribution but even more by the ways in which it can be interconnected with other facilities already in the system. The potential power of the SPEAKEASY system rests in this limitless growth capability.

Each user community exposed to SPEAKEASY can develop its own library. It can at the same time draw on the other libraries easily. The major problem being faced is not the construction nor the operations of the libraries themselves but rather in the communication between users and between user communities.

This report itself represents a crude method of communicating facilities, some of which have been available within the system for a long time. Other major facilities will no doubt become available before this report is actually distributed. It is for this reason that the current effort in development is directed towards techniques for information retrieval within the system, with emphasis on information available about the system itself. Only a few of the tools are available at present. They are in the process of being expanded to provide users with the necessary information.

The types of libraries can be divided into two sets, those that could in principle be read directly by people and those that represent stored information that is usable only by the computer. In the former class one can put the various forms of documentation that are associated with this system or that are stored within the system for other reasons. In addition, stored decks of SPEAKEASY statements represent information that is readable by people although it is also intended to be used directly by the computer.

The latter type of library (i.e., computer-readable information) for the most part represents stored compiled computer programs. These are specially designed routines that are compiled independently from the processors and are placed in accessible libraries. These libraries, in which the specialized tools intended to carry out major functions are stored, are the so-called LINKULE libraries.

LINKULES represent the real operational capability of the over-all SPEAKEASY system. They usually are efficient routines for carrying out specific mathematical operations. They are of use, however, only if there are documents that can explain how the operation is used and what it does. For each member in the communal libraries of the system, there must exist documentation. Such documentation is in one of the other libraries of the system and is thus readily available to users of the system.

It is obviously necessary to provide some method for finding the contents of specific libraries. The ability to ask the system for such information is essential so that exploratory searches can be made. Processors to enable the user to carry out such searches have been built into the LINKULE library. These have already been alluded to in the discussion of HELP.

Finally it is desirable that a user needing specific information should be able to carry out systematic searches through all the libraries of the system. This can in fact be done by connecting the facilities of the system described above together by a SPEAKEASY program such as the one described in the HELP section. Thus one sees that the documentation libraries, the LINKULE libraries, and the libraries of stored SPEAKEASY statements can be used together even at this very simple level.

Probably a single LINKULE could be written to carry out all of these functions. However, it would not have the flexibility inherent in the highly modular interconnected approach outlined above.

There are still problems of communication. A user of the language processor still can be unaware of how to get to a specific piece of information, even though that information exists within the system. To attempt to inventory all of the information available in the system, even as it exists today, would be a formidable task. It would surely swamp the user with undesired information. Lest there be a misunderstanding, it is not felt that this is an unsolvable problem. It is just that simple techniques of indexing and of report generation are not satisfactory solutions, and more powerful facilities will have to be added. A major effort in this direction has now been started.

The sections that follow describe each of the types of libraries currently considered part of the system. Others will be added as the need arises. Each section is a rather general over-all description of the purpose of the library. More specific information about the individual members of the libraries are to be found within the system itself. A set of operators for this purpose is described at the end of this section.

All libraries in the system are, in IBM terminology, partitioned data sets (PDS). Each such data set is a collection of members referred to by distinct member names. Although the data sets themselves are named, in this application we are interested only in the name used to refer to the data set in the SPEAKEASY run. This name by which we refer to the library is called the DDNAME in standard IBM Job-Control Language[*] and the FILE in TSO usage.[†]

The SPEAKEASY processor assumes that certain libraries are attached and checks to see if other special libraries are available for this run. The specific library names and their contents are as follows.

1) Libraries that are assumed to be attached are:

| | |
|---|---|
| LINKULES | libraries of operations (compiled FORTRAN routines) |
| PROCLIB | library of stored SPEAKEASY statements |
| HELP | library of brief HELP documents |
| DOCUMENT | library of larger documents |

2) Optionally attached libraries:

| | |
|---|---|
| MYLINKS | private versions of LINKULES |
| MYPROCS | private library of stored SPEAKEASY statements |
| MYHELP | private HELP words |
| MYDOCS | private documents |

Any additional libraries can be attached to the system to supplement those listed above. However, it is necessary to

[*] For standard batch jobs, the cards which have the form

//name DD · · ·

are called DD cards. The first field (i. e., the word name above) is the DDNAME referrred to and is called the library name in this report.

[†] In TSO runs, the statement

ALLOCATE FILE(name) DA(· · ·)

the equivalent way to define the library name.

explicitly communicate the names of such libraries to the processor. For example, adding additional LINKULES can be done by an explicit statement of the form

LINKLIB = 'XXX'

where XXX is an additional library of LINKULES. Similarly, document libraries are addressed by indicating the library name in the reference statement.

## A. The LINKULE Libraries

These libraries contain packages of compiled FORTRAN subroutines. Each of the members of such a library is available to the processors. If in the execution of any SPEAKEASY statement a word is encountered that has not been previously defined, then the system library is searched for a member with that name. If one is found, then that routine is brought into the computer and control is transferred to it. The calling sequence for these subroutines is designed to enable complete information transfer between processor and the individual LINKULES. The form of this calling sequence and the method of communication to the processor is described in Part Three.

The user library MYLINKS is of the same form as the system library but represents personal routines. If MYLINKS is an attached library, then it will be searched for a given member prior to the search of the LINKULES library.

## B. The PROCEDURE Libraries

Instead of distributing listings and copies of commonly used SPEAKEASY decks, a library named PROCLIB has been created for them. This library is always attached to the SPEAKEASY processors. The procedures in this library are directly available as input to the processor and can be used in any program by inserting a card of the form

USE MEMBER NAME

where NAME is the name of the particular procedure desired.

A user may append his own library of such statements to the system by assigning it the name MYPROCS. His library is then available for use within the run and is considered as part of PROCLIB for that run.

Since members of the procedure library are to be considered part of the generally available resources in SPEAKEASY, a description of the use of each procedure is also included in the HELP library.

### C. The HELP Library

The members of the HELP library (see Part Four) are concise documents describing words and features available to the SPEAKEASY processor. These documents are oriented towards the interactive user who is interested only in making use of the facility and not in a detailed description of its internal workings. The intent is to enable the user to quickly find out about a feature so that he may use it in the calculation currently before him.

It is intended that a HELP document will exist for every word used in the processor and for every member attached to it in a system library.

If the user attaches his own library of such brief documents, he should give it the name MYHELP. In this case, all of his documents are also available during that run.

Any specific HELP document can be obtained by an input statement

```
HELP XXX
```

where XXX is the name of the desired document.

## D. The DOCUMENT Library

In many cases the brief HELP documents described above are too concise to explain details about particular words. A larger document library is available for more lengthy descriptions. A member of this library is obtained by use of the statement

```
DOCUMENT XXX
```

where XXX is the name of the desired document. The user may attach his own library of documents. It should be given the library name MYDOCS. The library name for the system documents is DOCUMENT.

## E. LIBINDEX

The statement

```
LIBINDEX (name)
```

defines a literal 1-dimensional array with the members of the named library as components of the array. This array can be used in many ways, the simplest one being to tabulate it. The statements

```
LINKS = LIBINDEX (LINKULES)
TABULATE (LINKS)
```

will produce a table containing the names of all the LINKULES in the system library. Similarly

```
HELPNMS = LIBINDEX (HELP)
TABULATE (HELPNMS)
```

will list the names of all of the available HELP documents.

## F. LIST

The statement available to list members of a library has a generalized keyword format. Each keyword encountered is a signal that the next information is to be associated with the keyed option. The keywords, their default values, and their functions are

| KEYWORD | DEFAULT | Meaning |
|---|---|---|
| MEMBER | INDEX | Select member to be listed |
| LIBRARY | PROCLIB | Select library to search for member |
| FROM | 1 | Start listing from line # |
| TO | 90000 | End listing at line # |

Any words that are not keywords are ignored. Thus

```
LIST MEMBER MOON FROM 3 TO 7
```

will produce a printed copy of lines 3 through 7 of member MOON of PROCLIB.

## G. USE

Since decks of SPEAKEASY statements can-be stored in libraries, a method must be provided to make these decks available to the processor. Execution of the statement

```
USE MEMBER memname OF LIBRARY libname
```

causes the deck of the designated name from the library to be used as input to the processor. If the library reference is omitted, PROCLIB is assumed. If a library named MYPROCS is attached, it will be searched before looking in PROCLIB for this particular member. When the member has been completely read in, the input will be taken from the normal input device.

Such items as SPEAKEASY programs, series of HENCEFORTH statements, and notes to be printed to the user can all be in such libraries. The use of this facility in private libraries is to supply commonly used constants and SPEAKEASY programs to the processor in a simple way.

### H. CREATE

It is possible to create new members of documentation and procedure libraries while running in SPEAKEASY. This is done by the simple command

```
CREATE MEMBER memname OF LIBRARY libname
```

This command indicates that the lines that follow are to be used to define a new library member. The processor itself is passive in this operation. The creation of the new member is terminated by a single word

```
ENDCREATE
```

in the input data. All information between those two statements is saved as the newly created member.

## IV. LOGICAL AND RELATIONAL OPERATOR NOTATION

In the original specifications for SPEAKEASY, a special set of restricted keywords were used as logical and relational operators. This has on occasion caused difficulties because these words cannot be used as names of objects. For this reason, the language specifications have been changed. The introduction of SPEAKEASY 3E (October 1972) began a transition period to a logical operator notation similar to the FORTRAN conventions. During the transition period, both the old and new notations will be accepted.

In order to benefit from the new conventions, however, it is necessary to provide users with a means of eliminating the older restricted words. During the period of transition, therefore, a SPEAKEASY statement of the form

```
.NEW.
```

will deactivate the restricted words for logical operators. The statement

```
.OLD.
```

will reactivate them if it is necessary. The default condition will be .OLD. —at least during the early part of the transition period. The transition period will be a long one—users writing new programs should make use of the new notation and should gradually replace statements using the older words. The logical and relational operators in SPEAKEASY are listed in the following table.

| Older form<br>(being phased out) | Newer form<br>(now acceptable) | Meaning |
|---|---|---|
| LT | .LT. | Less than |
| LE | .LE. | Less than or equal to |
| NE | .NE. | Not equal to |
| EQ | .EQ. | Equal to |
| GE | .GE. | Greater than or equal to |
| GT | .GT. | Greater than |
| AND | .AND. | And |
| OR | .OR. | Or |
| NOT | .NOT. | Not |

## V. INTERACTIVE SPEAKEASY

An interactive version has been available since the inception of SPEAKEASY. The 2250 version of the language has been used for a variety of calculations. With the introduction of TSO, an interactive version of SPEAKEASY is now becoming available for use by a large community. TSO SPEAKEASY differs from conventional SPEAKEASY only in the interactive capabilities. Users in the MANUAL MODE of operation are able to direct the processor step by step through a calculation. They may examine the information, make corrections to it, and thus proceed directly through the steps to the completed results. In this mode of operation, the system can be viewed as a super desk calculator. Operations on whole arrays of elements can be carried out with a single command. All of the large sets of capabilities of SPEAKEASY are literally at the users' fingertips.

The program mode of operation is similar to the normal batch operation except that the results are instantaneously available. Small SPEAKEASY procedures that will be repeated several times can be programmed during the session at the terminal and used immediately.

Errors occurring in the manual mode are repaired by merely retyping the correct input. In the program mode, it is necessary to edit previously entered information. The EDIT mode is available for this purpose.

### A. The EDIT Mode

In batch processing, there is no need to edit the statements of a SPEAKEASY program since they can be altered only after the job has been completed. In interactive usage, on the other hand, such editing is of great importance. Facilities for this purpose are provided in SPEAKEASY. All processors include these features, but they are not normally used except in an interactive environment.

The EDIT mode is entered automatically when the word PROGRAM, PROCEDURE, or DATA is encountered in the input stream. In the EDIT mode, statements are assigned successive integer reference numbers starting with the number 1. The program mode is left if the single word END is encountered. At this time, the program is compacted and the statements are individually examined for syntax errors. The stored program is then defined as a single object whose name is the name of the program.

While the processor is in the EDIT mode, certain control functions are activated. All such functions are selected by a % in the first field in the input card. The functions are

| | |
|---|---|
| %LIST | List the entire edit file |
| %LIST N | List the statement with line number N |
| %LIST N, M | List statements with line numbers between N and M |
| %N Statement | Assign this statement the number N |
| %INSERT N | Insert the statements that follow at N, N + 1, · · · |
| %INSERT N(i) | Insert the statements that follow at N, N + i, N + 2i, · · · |
| % | Stop the insert |
| %DELETE N | Delete statement number N |
| %DELETE N, M | Delete statements N through M |
| %MOVE N | Move statement N to the last position |
| %MOVE N, M | Move statement N through M to the last positions |

| | |
|---|---|
| %MOVE N, M, K | Move statements N through M to K, K + 1, K + 2, · · · |
| %MOVE N, M, K(i) | Move statements N through M to K, K + i, K + 2i, · · · |
| %COPY N | Copy statement N into the last position |
| %COPY N, M | Copy statements N through M into the last positions |
| %COPY N, M, K | Copy statements N through M into K, K + 1, K + 2, · · · |
| %COPY N, M, K(i) | Copy statements N through M into K, K + i, K + 2i, · · · |

Although integer values are automatically assigned to statement numbers, these numbers can have smaller incremental values. During normal editing, numbers with increments as small as 0.01 are allowed. Thus a statement of the form % COPY 5, 10, 18.9 (.01) is acceptable.

If the process of editing generates a statement number that is identical to a previous one, then the old one is replaced. Care should be taken to protect previous information when performing multiple insertions.

For interactive processing, a second copy of the program is maintained. This copy contains statements in their original form and with the associated statement numbers. In such cases, the processor can be returned to the EDIT mode and a previously defined program can be activated by the statement

EDIT xxxx

where xxxx is the name of the program to be edited. This program can then be corrected by replacing, deleting, or inserting statements.

After it is satisfactorily corrected, the single word COMPILE will return SPEAKEASY to the manual mode. The word RUN is equivalent to the word COMPILE followed by the EXECUTE command.

In the interactive operation of SPEAKEASY, the user is informed of changes in the mode of operation as they occur. Once again, this information is suppressed for normal batch operations. In contrast, the program listing is normally printed in batch operation and is available only on command in interactive operation.

The following sample of the operation of the EDIT mode is a run carried out with the interactive version of SPEAKEASY operating under TSO.

```
:_program fit
 EDIT MODE
:_a=array(noels(x),nfit:) ; i=integers(1,nfit)
:_a=mfam((x+a)**(i-1)) ; y=vfam(y)
:_afit=1/(transpose(a)*a)*transpose(a)*y
:_print('    the best fit is ',afit)
:_end
 MANUAL MODE
:_x= 1 2 3 4
:_y=3*x**2+4*x+1
:_nfit=3
:_execute fit
 EXECUTION STARTED
  THE BEST FIT IS
   AFIT (A VECTOR WITH 3 COMPONENTS)
   1  4  3
:_edit
 O.K.-EDIT MODE
:_%list
 %1       PROGRAM FIT
 %2       A=ARRAY(NOELS(X),NFIT:) ; I=INTEGERS(1,NFIT)
 %3       A=MFAM((X+A)**(I-1)) ; Y=VFAM(Y)
 %4       AFIT=1/(TRANSPOSE(A)*A)*TRANSPOSE(A)*Y
 %5       PRINT('    THE BEST FIT IS ',AFIT)
 %6       END
:_%5.5 yfit=a*afit ;tabulate x y yfit
:_%run
 EXECUTION STARTED
  THE BEST FIT IS
   AFIT (A VECTOR WITH 3 COMPONENTS)
   1  4  3
   X  Y   YFIT
   1  8   8
   2  21  21
   3  40  40
   4  65  65
:_x=1.1 2.03 3.34 4.43 5.16
:_y=2.78 4.4 10.7 19.4 27.1
:_execute
 EXECUTION STARTED
  THE BEST FIT IS
   AFIT (A VECTOR WITH 3 COMPONENTS)
   3.8691 -2.4795  1.3526
   X      Y      YFIT
   1.1    2.78   2.7782
   2.03   4.4    4.4095
   3.34   10.7   10.676
   4.43   19.4   19.429
   5.16   27.1   27.087
```

## B. INPUT/PAUSE/STOP

In interactive usage of the program mode of SPEAKEASY, it is often desirable to interrupt the computation at specific places but to retain the capability of resuming at that point. Three statements are available in the program mode for this purpose. All are identical in operation; the choice between them is purely subjective. If the first word on a SPEAKEASY statement encountered in the execution mode is INPUT, PAUSE, or STOP, then the entire statement is printed out and the execution of the program is interrupted. The system is put into a mode referred to as the HOLDING mode.

This mode is in fact the MANUAL mode with the added capability of resuming the execution of the SPEAKEASY program at a later time. All of the facilities of the manual mode are available. New objects can be defined, old results can be examined, etc. Whenever the objectives of the interrupt have been met, then the execution can be resumed by entering a statement with one of the words RESUME, CONTINUE, or GO. If one desires to terminate the HOLDING mode, the statement MANUAL places the system in the true manual mode. In operation a statement of the form

```
INPUT A, B AND C PLEASE
```

encountered during execution would print out

```
INPUT A, B AND C PLEASE
```

and the user might then type in

```
A = 4; B = 7; C = 27.48*W; RESUME
```

Note that the operations in the HOLDING mode are completely general and need not be restricted to the implied requests. The only restriction is that the EXECUTION mode itself may not be used. If it is used, then one loses the ability to resume from this point at some later time.

# VI. VERSIONS

SPEAKEASY is a general processing language. The modes of operation, even at Argonne, are rather diverse. Card-input, remote-job-entry, and remote-job-output facilities all imply slightly different optimal forms of operation. In contrast, in the truly interactive mode (e.g., on the IBM-2250 console or the newly available TSO version) the user may need a quite different form. Instead of attempting to construct specialized processors for each application, the approach taken has always been to include as many diverse capabilities in the basic processor as possible. Each use can then select the available ones that are most clearly desirable in the application. Specialized input and output requirements are met by isolating them in two or three replaceable modules. These can easily be adapted to special devices such as the 2250.

Several different versions of the SPEAKEASY processor are now available. The computational logic is identical in various versions. They differ primarily in their space requirements and in their efficiency of operation. It is expected that some of these versions will be combined in the near future. The following versions are available.

## A. STANDARD

This is the standard production version of the language for batch processing. It is a non-overlayed version that occupies 260K[*] of core. This version is the fastest running version available

[*] The size given is a nominal one. It was selected on the assumption that the size of the LINKULES does not exceed 10 K.

but does require the largest amount of computer core. The default settings can be viewed as

SIZE = 40, MAIN
MARGINS (1, 128)
LIST
ECHO

B. BABY

This is a heavily overlayed version of the above processor and is designed to provide rapid batch turnaround at the sacrifice of execution efficiency. The default settings are

SIZE = 40, MAIN
MARGINS (1, 128)
LIST
ECHO

This processor requires about 160K to operate.

C. GRAPHEZ

The graphical facilities of the language (for use with the CALCOMP 780 device) are maintained only in this special version. The version is identical to the standard version described above except for these additional features. This version requires 280K to operate.

D. CONSOLE

This version of the language is adapted to the 2250 display console. It is a heavily overlayed version that is designed for export. The operation of this version is similar to that of the TSO version except that graphical output is directly available. Since the display is on an oscilloscope, a monitor copy of the input and output is also produced on a line printer for later reference. The defaults are

SIZE = 8
MARGINS (1, 70)
LIST
ECHO

E. SPEK2250

This is the production console version of the language for use at Argonne. It is a non-overlayed version, but it is designed for use with LCS as the primary core storage. This tailored version is similar to the CONSOLE version except for the use of LCS.

F. TSO

The adaption of SPEAKEASY to operation with TSO is relatively new. The major difficulty has been one of trimming the processor to a size acceptable for use in TSO regions of normal size. Since each installation chooses this size to meet its specific needs, no single generally acceptable size has yet been established. In decreasing the size of the standard version, the already small version BABY has been further overlayed. The version that is now available will operate in a 120K TSO region. Further decreases in this size are expected; but since further decreases become more and more difficult, it is unlikely that a much smaller version will be available soon. The default options for this version are

```
SIZE = 4
MARGINS (1, 128)
NOLIST
NOECHO
```

## ACKNOWLEDGMENTS

SPEAKEASY is a language developed to serve its user community. Complaints and praise by the users, though apparently ignored in the short term, have gradually influenced the structure of the language. It is through discussions with users, particularly in the Physics Division, that defects and desirable facilities are first noted. The form and variety of the features available are therefore mostly due to the users themselves. I wish to thank those who have expressed their needs and desires and who have thus influenced this development. In particular, special thanks are due to Joanne Fink, Harvey Z. Kriloff, Steve Pieper, Keith Rich, Frank Serduke, and Martin Vincent, who have been generous with their time and who have been directly involved in many of the major discussions that have influenced the development of the language. Since the language continues to evolve, it is hoped that such direct influences will continue.

# PART THREE

## SPEAKEASY-3: Linkules and Interfaces

# I. LINKULES FOR SPEAKEASY-3

by

S. Cohen, F. J. D. Serduke, and K. Rich*

## A. Introduction

One of the most powerful features of the SPEAKEASY-3 processors is their ability to operate with attached libraries. By far the most important of these libraries is the so-called linkule library that contains individually compiled FORTRAN program packages that can be selectively used by SPEAKEASY during execution. The importance of such libraries becomes clear each time new applications are found for the SPEAKEASY processors. New words may be added to the linkule libraries to meet the particular requirements of these applications. The gradual growth of the basic systems is by the inclusion of new well-tested linkules into the system libraries. Each such addition becomes available to the entire user community and thus provides a more powerful processor for everyone.

It should be clearly understood that the linkule libraries are not part of the basic processors. The introduction of new linkules in no way alters the processors. Although a newly added linkule may produce erroneous results, the existence of that linkule will in itself not affect programs that do not address it. This means that each of the modules which provide the SPEAKEASY operational capabilities are independently correctable without fear of any subtle interconnections. Individual linkules can be added, altered, or removed from the overall system without affecting other parts of the system.

Users may have private libraries of linkules that contain operations that are either of very limited application or are not yet considered trustworthy. Such private libraries function in exactly the same manner as the communal libraries and are considered part of them during the computer runs in which they are attached to the system. When a private linkule is transferred to a communal library there is no change in the operation of either the linkule or the processor.

*National Accelerator Laboratory.

This document describes how linkules are written and attached to the system. The primary purpose in writing this report is to supply a user community with the means of constructing a SPEAKEASY processor with a vocabulary tailored to the needs of that group. This document is intended to contain all of the information necessary to construct such a new vocabulary. This is not an easy task since several levels of detailed understanding are needed. It is hoped that the following information contains enough redundancy to enable a competent FORTRAN programmer to learn to add words to SPEAKEASY.

The first section of this report describes how SPEAKEASY searches for specific names and how this process leads to a particular linkule. This is in essence a description of the search heirarchy of the processor.

The next section deals with the form of the argument list in a linkule and gives a detailed description of each piece of information transferred to the linkule for its use. A major part of this description deals with the form of objects defined in the SPEAKEASY processor. The means of defining a new object is also discussed.

The third section deals with the process of returning control from the linkule to the processor and explains the method by which error messages are transmitted.

Several examples of linkules are given to aid in the understanding of specific details.

## B. How a Linkule Is Activated

During execution of a SPEAKEASY program, each word encountered is examined for defined meaning in the following sequence of questions.

1. Is it one of the few restricted keywords in SPEAKEASY, e.g., PROGRAM, EXECUTE, etc.?
2. Is it a currently defined object, i.e., is it a defined variable, program, or data file?

3. Is it one of the "standard" words of the language such as PRINT, TABULATE, SQRT, etc. ?

If none of the above is true then the same questions are asked for the particular linkule libraries attached to the system for this run. These libraries are identified by their DDNAMES ("library names"). The libraries LINKULES and MYLINKS are automatically searched for a member with a name corresponding to the word being sought. Additional libraries may also be attached by defining a SPEAKEASY object called LINKLIB with the library names of additional libraries to be searched. The statement

LINKLIB = 'XXX'

will cause the library named XXX to be included in the search. If the word being sought is not located in any attached library, then an error message indicating that it is not defined is generated and the search process is terminated.

If the word is found in any of the libraries, then the member is brought into core and control is transferred to it. This is done by use of the standard IBM-supplied LINK macro. The linkule carries out its operation and returns control to SPEAKEASY by executing a normal RETURN statement.

After returning control, the core space used by the linkule is available for later use. The operating system attempts to provide for efficient reuse of the linkule by retaining it in core for possible later use, but it will make use of the space if necessary. In order for the operating system to operate in this manner, it is necessary that linkules be designed and marked REUSABLE (as explained in Sec. F).

The logical form in which control is passed to a linkule is exactly the same as that in which a normal FORTRAN function routine is called. The operating system in essence performs the bookkeeping necessary to locate the arguments and to pass them to the linkule. From the user viewpoint, therefore, a linkule is a standard FORTRAN function routine with a specified argument list. This routine in turn can call any other routines necessary to carry out its operation. For any existing FORTRAN routine, a linkule can be constructed by writing an interface routine by which the argument list of information supplied by SPEAKEASY is matched to that used by the FORTRAN routine.

A linkule is an entirely independent program package. Any subroutines used by the linkule must be contained within that linkule (certain exceptions will be explained later). This is one reason that the design of the linkule is of importance. If, for example, any usual input or output is attempted, then the entire package of routines involved in formatting information must be included within the linkule. If one is not careful in the use of routines the size of individual linkules will force the use of unreasonable core allocations. Moreover, the structure of SPEAKEASY implies that linkules should represent clean mathematical operations. This means that, for the most part, the linkules can and should be small packages carrying out very specific operations.

## C. The Argument List of a Linkule

A linkule is an entirely detached programming package. All information to be transferred between the SPEAKEASY processor and the linkule must be carried through the argument list used in invoking the linkule. As will be seen, this list is a long one and is complete in the sense that all information necessary to write any possible linkule is available. For any particular operation, there is therefore an overabundance of information.

When a linkule is given control, each argument in the SPEAKEASY statement that invoked the linkule must be described completely. It should be understood that much of the original

SPEAKEASY statement may already have been evaluated. If a linkule called JONES is called from SPEAKEASY as a result of the statement

X = JONES (A, B+7, B*X+AVERAGE(Y))

then the linkule is invoked as if the statement were

X = JONES(A, temp 1, temp 2)

there temp 1 and temp 2 are objects whose description will be passed to the linkule. If B had not been defined, then an error would have been detected before attempting the call to the linkule since it was involved in an algebraic statement. On the other hand, if A had not been defined, this fact would be conveyed to the linkule. All expressions such as those above would be conveyed to the linkule and are evaluated before attempting to locate the linkule.

It has been said the argument list of a linkule is a long one. The first cards of a linkule should read

```
      FUNCTION LINKUL
     1(ANS,IGNORE,NOARGS,ICOL,ICOM,IDOM,ACC,ARG,VAL,VALI,IVAL,KIND,
     2KLASS,NROWS,NCOLS,NWORDS,LOC,ALLOC,ICLRES,IQUERY,IFREE,IQURES)
C
      IMPLICIT REAL*8 (A-H,O-Z)
C
      DIMENSION ARG(1),VAL(1),VALI(1),IVAL(1),KIND(1),KLASS(1),
     1          NROWS(1),NCOLS(1),NWORDS(1),LOC(1),ALLOC(1)
```

These cards can be used in exactly this form for all linkules. Each of the arguments will now be explained.

ANS — This is an eight byte word that contains the name that is to be used to define the result of this call (if there is a result). This is a name generated by the SPEAKEASY processor.

IGNORE — This is an argument useful for multiple-entry forms of LINKULES. Few linkules make use of this argument. It is therefore being ignored in this report.

NOARGS This parameter is an integer specifing the number of arguments appearing in the SPEAKEASY statement that caused this call for the LINKULE.

ICOL This indicates the location of the first colon that appeared in the argument list. If ICOL is zero, no colon appeared. The integer is the number of the argument appearing immediately after the colon (1 means the colon is the first field).

ICOM If this is zero, there were no complex or imaginary arguments. If it is 1 then at least one such argument was encountered.

IDOM If zero, the current domain is set to REAL. It it is 1, the domain is COMPLEX.

ACC This is the current accuracy setting of the processor. This number is used whenever a decision involves a comparison between two numbers.

The next ten items are one-dimensional arrays. Each of them describes some property of an argument in the SPEAKEASY statement. The Ith element of each array corresponds to the property of the Ith argument. These properties are

ARG(I) The name appearing as the Ith argument in the SPEAKEASY statement.

VAL(I) The value of the real part of the first element of the Ith argument.

VALI(I) The value of the imaginary part of the first element of the Ith argument.

IVAL(I) The value of the integer part of the real part of the first element of the Ith argument.

KIND(I) The kind of the Ith argument.

For SPEAKEASY users, the following KINDs are normally all that are encountered.

| KIND | |
|---|---|
| 0 | Not defined. |
| 2 | A real object. |
| 3 | An imaginary object. |
| 4 | A complex object. |
| 6 | A name array (literal 8 byte). |
| 9 | A literal array (literal 1-byte objects). |

If KIND is negative, the argument is a so-called "in place definition," i. e., it occurred as an explicit constant in the statement. For example, linkule BUD (3, 4, 5+6I) has arguments that are in-place definitions.

KLASS(I) Indicates the class of the Ith argument. The class describes the structure of the object. The ones encountered in SPEAKEASY are:

| | |
|---|---|
| 0 | A scalar. |
| 1 | A vector. |
| 2 | A matrix. |
| 5 | A 1-dimensional array. |
| 6 | A 2-dimensional array. |

Many other kinds and classes are possible, but these are omitted here because they will not normally be used in writing linkules.

NROWS(I) Indicates the number of rows or the length of the Ith argument.

NCOLS(I) Indicates the number of columns or the width of the Ith argument.

If either NROWS(I) or NCOLS(I) is not applicable, it is set equal to 1.

NWORDS(I) Indicates the number of words in the Ith argument. It is normally the product of the number of rows by the number of columns for this object.

The next two items are used to address the information in the object as a whole. Each structured object is to be viewed as located in the array specified by these two items.

ALLOC The name of the array.

LOC(I) The location of the first element of the Ith argument in the array ALLOC.

The final four items in the argument list of the linkule are four subroutines that are used to define, locate, or free objects in SPEAKEASY. These are used as follows:

LO = ICLRES(ANAME, KIND, KLASS, NROWS, NCOLS) Used to define a new object with the name contained in ANAME (an 8-byte literal) and with the properties of the object described by the rest of the arguments. These are identical in meaning to the definitions above. The functional value (in this case LO) is a location in ALLOC of the newly defined object. All elements of this object are set equal to zero. The location returned is that of the first element of the object. Successive elements (by rows for 2 dimensional objects) are in successive locations of ALLOC.

CALL IFREE(ANAME) Used to free or undefine any object. The name of the object to be freed is contained in the word ANAME

LO = IQUERY(N, ANAME, KIND, KLASS, NROWS, NCOLS) Used to locate and obtain the description of some currently defined object with the name contained in ANAME. The KIND, KLASS, and dimensions NROWS and NCOLS are returned by this routine. The location of the object is returned as a functional value and also put into LOC(N) where LOC is the previously described array. Note: If N is identical to a previously used locator, it will overwrite the information that was previously in that locator. N must be an integer less than 30 and normally is chosen to be larger than the number of arguments in the linkule.

LO = IQURES (N, ANAME, KIND, CLASS, NROWS, NCOLS) This routine combines ICLRES and IQUERY. It defines the object with the specified name and structure and places its location into the nth position in LOC.

The items listed above complete the specifications of the various arguments in the list that a linkule has available through the calling sequence. To be sure that the information entries are understood, a few examples of their forms are given here. Let us assume that the linkule is called JONES. Then the sequence

```
X = 9
Y = MATRIX(3, 3 : 1, 2, 3, 4, 5, 6, 7, 8, 9)
Z = ARRAY(7 : 2, 3, 4, 5, 6, 7, 8)
T = JONES(7, X : Y, Z, X*Z+9)
```

will result in the linkule being called with the following information:

| | | |
|---|---|---|
| NOARGS | = | 5 |
| ICOL | = | 3 |
| ICOM | = | 0 |

| I | ARG | KIND | KLASS | NROW | NCOL | VAL | IVAL |
|---|---|---|---|---|---|---|---|
| 1 | — | -2 | 0 | 1 | 1 | 7.0 | 7 |
| 2 | X | 2 | 0 | 1 | 1 | 9.0 | 9 |
| 3 | Y | 2 | 2 | 3 | 3 | 1.0 | 1 |
| 4 | Z | 2 | 5 | 7 | 1 | 2.0 | 2 |
| 5 | — | 2 | 5 | 7 | 1 | 27.0 | 27 |

VALI will be zero in all cases.

Similarly the sequence

```
DOMAIN COMPLEX
X = ARRAY(4 : 1, 2, 3, 4)
Y = 3I*X
Z = MATRIX(3, 2 : 2+3I, 3+4I)
T = 'ALPHA'
FREE(W)
TT = JONES(3, 5I : X, Y, Z, T, W)
```

causes the following to be passed to the linkule "JONES."

```
NOARGS = 7
ICOL = 3
ICOM = 1
IDOM = 1
```

| I | ARG | KIND | KLASS | NROW | NCOL | LOC | VAL | VALI | IVAL |
|---|---|---|---|---|---|---|---|---|---|
| 1 | — | -2 | 0 | 1 | 1 | 0 | 3.0 | | 3 |
| 2 | | -3 | 0 | 1 | 1 | 0 | 6.0 | 5 | 0 |
| 3 | X | 2 | 5 | 4 | 1 | non-zero | 1 | — | 1 |
| 4 | Y | 3 | 5 | 4 | 1 | non-zero | — | 3 | 0 |
| 5 | Z | 4 | 2 | 3 | 2 | non-zero | 2 | 3 | 2 |
| 6 | T | 6 | 5 | 1 | 1 | non-zero | 'ALPHA' | — | — |
| 7 | W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

For SPEAKEASY users who have any questions about the form of the elements displayed for any particular situation, a special linkule in the system is available to resolve test cases by example. Thus a SPEAKEASY statement of the form

```
SHOWLINKAGE (A : B, C, D, E, · · ·)
```

will cause a printout similar to those used above to display the consequences of particular calls. Before attempting to write linkules, a new user is advised to make a series of runs to learn the consequences of specific calls.

The above descriptions explain how information describing particular objects in the argument list are conveyed to the linkules and how new objects can be defined. A very important consequence of being able to define objects at execution time (a very fundamental concept of SPEAKEASY) is that defined objects can be moved to meet the demands of the operating system. With this in mind, the next section must be read and understood clearly when defining objects in linkules.

## D. On Motion of Defined Data

Each time a new piece of data is defined by use of ICLRES or IQURES or if new data is located by IQUERY, there is the possibility that other defined information will move. (This is the nature of the storage scheme used by SPEAKEASY.) The locations in the array LOC are always the correct current locations of the corresponding objects. In practice it is therefore necessary that the locations of objects be obtained from the LOC array after any of the above routines have been made.

If only a single object is being defined in the LINKULES (as in the case if only the resultant ANS is to be defined), then only ICLRES need be used. The location returned is the proper one. All other locations are then obtained from the LOC array. These locations must be ascertained after the ICLRES routine is called.

If several objects are to be defined, it is necessary to ensure that the system will keep track of their current locations. This is done by using the routine IQURES and specifying a value of N not currently in use. This will mean that LOC(N) is to be maintained at the current location of this object.

## E. Returning from a Linkule

To return control to the SPEAKEASY processor, execute the RETURN statement in the logically top routine in the linkule. All information needed by the linkule is communicated through the argument list. On return, these same arguments are used to communicate with the processor. The only other channels of communication between the two is the functional value of the linkules.

The information to be returned includes the answer (if there is one) defined as shown in Secs. III and IV. In addition, it is necessary to tell the processor whether or not such an answer was created. The presence of an answer is indicated by the functional value of the routine: if it is 1, then an object with the name contained in ANS was defined; if it is -1, then no such resultant is to be expected by the SPEAKEASY processor.

Err6rs are indicated by setting the values of the arguments ICOL and ICOM. ICOL is always set zero or positive on input. ICOM is either 0 or 1. If either of these is set to another value on exit, then the processor is aware that an error was detected and prints an error message.

The errors messages are controlled as follows. (1) If ICOM is not negative and ICOL is -N, then the error message corresponding to N in the table in Table 1 will be generated. The message will repeat the input line and then will give the error message. To indicate that a specific argument is involved in this error, the user may set NOARGS equal to -M to indicate the <u>M</u>th argument. (2) If ICOL is positive and equal to N and ICOM is positive and greater than 1 then the statement

ARGN AND ARGM ARE INCOMPATIBLE FOR OPERATION

is generated. Here ARGN is the name of the <u>N</u>th argument and ARGM is the name of the <u>M</u>th argument. (Note that ICOM is never preset to any number larger than 1. If two arguments are incompatible, set ICOM to the index of the second. (3) If ICOM is -N, then the statement

ARGN IS NOT DEFINED

```
E   1 ILLEGAL CHAR.
E   2 DOUBLE OP.
E   3 MISPLACE DEC. PT.
E   4 NUMERIC OVERFLOW.
E   5 PARENTHESIS IMBALANCE.
E   6 STATEMENT TOO LONG.
E   7 DOUBLE EQUAL SIGN.
E   8 REAL OBJECTS ONLY
E   9 ILLEGAL LOGICAL OPERAND.
E  10 MISPLACED '?'.
E  11 IMPLIED MULT. ?
E  12 ENTERED COMPLEX DOMAIN.
E  13 TRANSLATION ERROR
E  14 OPERATOR SEQUENCE?
E  15 IS NOT A SQ. MATRIX.
E  16 DIVISION BY ZERO.
E  17 IS A SING. MATRIX.
E  18 NON-REAL LOGICAL OPERATION.
E  19 INDEX OUTSIDE BOUNDS.
E  20 COMPLEX INDEX.
E  21 PROGRAM IDENT. MISSING.
E  22 ALPHABETIC LEFT SIDE.
E  23 RESTRICTED OPERATION.
E  24 TOO MANY ARGS.
E  25 A BAD ARGUMENT.
E  26 ARGUMENT IS NOT DEFINED.
E  27 FILE NOT DEFINED.
E  28 FILE PREVIOUSLY DEFINED.
E  29 PROGRAM NOT DEFINED.
E  30 WRONG NUMBER OF ARGS.
E  31 OPS (COMPLEX) ARE LIMITED.
E  32 ZERO OR NEG. LENGTH. DEF.
E  33 SYSTEM FAILURE (DATA MAY BE LOST).
E  34 EXCEEDED CORE SIZE.
E  35 AN IMPROPER ARGUMENT.
E  36 WAIT.
E  37 BRANCH IS NOT DEFINED.
E  38 NO PROGRAM CARD.
E  39 NO FILE NAME.
E  43 ARG. OUTSIDE ALLOWED BOUNDS.
E  44 NO UPPER FOR BOUND.
E  45 LOOP NOT ACTIVE.
E  46 MAX FOR DEPTH IS 10.
E  47 EITHER: INDEX IS ALREADY IN USE OR REAL PART OF A NON NUMBER.
E  48 INCR. WRONG SIGN.
E  49 DELTA IS ZERO.
E  50 INTERP ERROR IN LOADDATA.
E  51 REAL COMPLEX CONFLICT.
E  62 DEFINITION LARGER THAN 8000 WORDS.
E  63 PROG. TOO BIG.
E  64  NON REAL ARG.
E  65 TWO DIM. HORIZ ARG.
E  67 NON REAL ARG.
E  68 NO. ELS. DIFFER IN ARG LIST.
E 100 NON REAL ARGUMENT.
E 102 LENGTH CONFLICT BETWEEN ARGS.
```

Table 1. SPEAKEASY Error Messages

is generated. Again ARGN is the name of the Nth argument.

Once control is returned to the processor, the linkule is logically disconnected from the processor. If this particular linkule is used again, a fresh copy may be brought in. One should therefore not assume that information set within a routine in one call will be there on subsequent calls to the same linkule. If information is to be retained, it should be stored in an area defined by use of the ICLRES statement.

## F. On Reusability

The dynamic link process used in transferring control to a linkule is a standard IBM facility. In attempting to transfer control by the so-called link method, the routines previously linked are searched to see if the desired routine is already available and usable. If it is not found, then a copy is loaded into the machine from the appropriate disk. This loading is a time-consuming operation. Calling a linkule in a loop in SPEAKEASY could therefore be extremely expensive if the linkule has to be repeatedly loaded. To avoid this, the link-edit step should include marking each such linkule "reusable." Unless a load module is specifically marked as being reusable, it must be reloaded for each use.

A reusable module can include no previously link-edited parts that were not reusable. This is a somewhat bothersome feature since Argonne and all other institutions thus far surveyed have marked their FORTRAN library routines as being nonreusable. For this reason (and others), a special set of FORTRAN library routines, called CONS.LOAD in the examples, has been made available for constructing linkules. The members of this library are marked "reusable."

The library also serves to hold linkules to a manageable size. FORTRAN library subprograms are designed to produce error messages for badly defined arguments. Such error messages require

*Many of the subroutines in this library were supplied by Dr. R. K. Nesbit.

that the complete FORTRAN input/output package be included in the linkule. The result of including such a package is to increase the minimum size of a linkule from 1K bytes to 25K bytes. For the reason alone, the special library (designed to eliminate error messages) should be used when constructing linkules.

F. Sample Linkules

This section iscompleted by showing a few of the linkules currently in the SPEAKEASY library. They should be helpful in understanding some of the specifics about how a linkule is actually written.

```
//SCUMPROD JOB (F88888,1,1,1),CLASS=A,REGION=210K
//   EXEC SOS,LIB=PHYSICS
//KEEP DD DSN=PHYSICSP.LOAD,DISP=OLD
/COMPILE=H
      FUNCTION MYWORD (ANS,ITH,NOARGS,ICOL,ICOM,IDOM,ACC,
     1ARG,VAL,VALI,IVL,KI,KL,NR,NC,NW,LOC,ALLOC,ICLRES)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION VAL(1),VALI(1),IVL(1),ARG(1),KI(1),KL(1),NR(1),
     1NC(1),NW(1),LOC(1),ALLOC(1)
C
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C  SPEAKEZ LINKULE FOR THE OPERATOR "CUMPROD"
C       Y = CUMPROD(X)
C  RETURNS AN OBJECT  Y  OF THE SAME STRUCTURE AS  X
C  AND WHOSE N-TH ELEMENT IS THE CUMULATIVE PRODUCT
C  OF THE FIRST N ELEMENTS OF X.  TWO-DIMENSIONAL
C  OBJECTS ARE TREATED ROW-BY-ROW.
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      MYWORD = 0
CAN ONLY HANDLE REAL OBJECTS IN THIS IMPLEMENTATION...
      IF ( ICOM .NE. 0 ) GO TO 9000
CHECK THAT THE INPUT OBJECT IS DEFINED...
      IF ( LOC(1).EQ.0 .AND. KI(1).GE.0 ) GO TO 9001
CONFIRM THAT THERE WAS ONLY ONE INPUT ARGUMENT...
      IF ( NOARGS .NE. 1 ) GO TO 9002
CHECKS PASS...SET LOOP LIMIT AND CLEAR SPACE FOR ANSWER...
      NLIMIT = NW(1)
      Y = 1.0
      LOCY = ICLRES(ANS,KI(1),KL(1),NR(1),NC(1))
      LOCX = LOC(1)
CALCULATE THE CUMULATIVE PRODUCT...
      DO 10 N = 1,NLIMIT
      Y = Y*ALLOC(LOCX+N-1)
   10 ALLOC(LOCY+N-1) = Y
      MYWORD = 1
      RETURN
CAUGHT SOME INVALID INPUT...SET THE ERROR CODE...
 9000 ICOL=-31
COMPLEX OPERATIONS NOT AVAILABLE
      RETURN
 9001 ICOM=-1
CRUMMY ARGUMENT...IT IS NOT DEFINED
      RETURN
 9002 ICOL=-24
CUMPROD TAKES ONLY ONE ARGUMENT...WAS FED MORE THAN ONE...
      RETURN
      END
/KEEP CUMPROD 'LIST,MAP,REUS,NCAL,LET'
```

```
//SCEONE JOB (F88888,1,1,1),CLASS=C,REGION=160K,MSGLEVEL=1,PRTY=L
// EXEC FTHCEP,OPTIONS='OPT=2,MAP',
//      EDTOPTS='LIST,MAP,LET,REUS',LSIZE='(150K,50K)'
//FTH.SYSIN DD *
      INTEGER FUNCTION E1LINK
     &     (ANS,ITH,NOARG,ICOL,ICOM,IDOM,ACC,ARG,VAL,VALI,IVL,KI,
     &           KL,NR,NC,NW,LOC,ALLOC,ICLRES)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION VAL(30),VALI(30),IVL(30),ARG(30),KI(30),KL(30),NR(30),
     &     NC(30),NW(30),LOC(30),ALLOC(30)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C     THIS IS THE SPEAKEASY LINKULE FOR THE EXPONENTIAL INTEGRAL
C
C        EONE(X) = INTEGRAL(X,INFINITY)  DT EXP(-T) / T
C
C     THIS ROUTINE EMPLOYS THREE SEPARATE RATIONAL APPROXIMATIONS
C     FOR EONE(X)  FOR X IN THE RANGES  0<X<1 , 1<X<4 AND 4<X<170 .
C      IF X>170 THIS ROUTINE SETS EONE(X)=0 WITH NO ERROR MESSAGE
C          THE SPEAKEASY WORD IS     EONE(X)   WHERE X IS A REAL
C     VARIABLE OF ANY STRUCTURE.
C     THE RATIONAL APPROXIMATIONS WERE DEVELOPED BY W.J. CODY OF
C     THE ARGONNE NATIONAL LABORATORY APPLIED MATH DIVISION.
C       REFERENCE:  W.J. CODY AND H.C. THATCHER JR.
C          "RATIONAL CHEBYSHEV APPROXIMATIONS FOR THE EXPONENTIAL
C          INTEGRAL E1(X)"  MATH COMP 22   (1968)
C                        ARNIE OSTEBEE AND FRANK SERDUKE  8/22/72
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C --- DATA STATEMENTS FOR CONSTANTS IN THE RATIONAL APPROXIMATIONS
      DATA  A0/ZC093C467E37DB0C8/ , A1/Z40C110E996FE3178/
      DATA  A2/Z40213DCCA0D570ED/ , A3/Z3F629544F2457AA6/
      DATA  A4/Z3E569011E23A2DC8/ , A5/Z3D44F80EBCEB91CF/
      DATA  B0/Z4110000000000000/ , B1/Z406D07BAC4D04469/
      DATA  B2/Z40146C6D6E730856/ , B3/Z3F22015DE897CE[illegible]E/
      DATA  B4/Z3E1FE0E4A6BA9AF7/ , B5/Z3CDB33FAB422E1DA/
      DATA  C0/Z3B174B1AD3E2E266/ , C1/Z40FFFFB4D38BC730/
      DATA  C2/Z41BD92AE16CB369D/ , C3/Z422D97D311F74081/
      DATA  C4/Z4245ED8DCFDDF7E2/ , C5/Z422A852C0E187B8C/
      DATA  C6/Z418D633281818C45/ , C7/Z4066C0AFC843D717/
      DATA  D0/Z4110000000000000/ , D1/Z41CD923363206BCC/
      DATA  D2/Z4238717FD76B3FA3/ , D3/Z426AA52AC377B0AB/
      DATA  D4/Z4259BB2A01911995/ , D5/Z421F7F4782BEC4CE/
      DATA  D6/Z413CBABC9E89B478/ , D7/Z401743F10E4FE979/
      DATA  E0/ZC0FFFFFFFFFFF884/ , E1/ZC22267FCB0C305CF/
      DATA  E2/ZC31AB885D23D00FD/ , E3/ZC395C04F986D432F/
      DATA  E4/ZC41818DA23430F9D/ , E5/ZC419B018D42BE7A2/
      DATA  E6/ZC383A13CE9D21BEA/ , E7/ZC1EE62A6F33B49AD/
      DATA  F0/Z4110000000000000/ , F1/Z422467FCB0C2ED0F/
      DATA  F2/Z431EE585685725C3/ , F3/Z43C7645BA5C7422E/
      DATA  F4/Z4428611347 6C61CF/ , F5/Z443FC42536094A4D/
      DATA  F6/Z442B8DC679374D02/ , F7/Z4394A2394EA64382/
      E1LINK = 0
C -- CHECK TO SEE THAT THERE IS ONLY ONE ARGUMENT
      IF (NOARG .NE. 1)  GO TO 9001
C -- CHECK IF ARGUMENT IS DEFINED...
      IF ( LOC(1).EQ.0 .AND. KI(1).GE.0 ) GO TO 9003
C -- AND SEE THAT THE ARGUMENT IS NOT COMPLEX
      IF ( ICOM .NE. 0 ) GO TO 9004
```

```
C -- MAKE SPACE FOR THE ANSWER
      LOCANS = ICLRES(ANS,IABS(KI(1)),KL(1),NR(1),NC(1))
      LIMIT = NR(1)*NC(1)
C -- LOOP ON THE ELEMENTS OF THE INPUT VARIABLE
      DO 500 INDEX = 1,LIMIT
      IF ( KI(1) .LT. 0 ) GO TO 50
      X = ALLOC(LOC(1) - 1 + INDEX)
      GO TO 100
50    X = VAL(1)
100   IF ( X .LE. 0.0D0 ) GO TO 9002
      IF (X.GT.1.D0) GO TO 200
C -- RATIONAL APPROXIMATION FOR  0<X<1
      EONEX=(((((A5*X+A4)*X+A3)*X+A2)*X+A1)*X+A0)/
     1      (((((B5*X+B4)*X+B3)*X+B2)*X+B1)*X+B0)
     2      -DLOG(X)
      GO TO 500
C -- FOR X > 170 SET EONE=0
C -- RATIONAL APPROXIMATION FOR 1<X<4
200   W=1.D0/X
      Y=DEXP(-X)
      IF (X.GT.4.D0) GO TO 300
      EONEX=Y*(((((((C7*W+C6)*W+C5)*W+C4)*W+C3)*W+
     1      C2)*W+C1)*W+C0)/(((((((D7*W+D6)*W+D5)*
     2      W+D4)*W+D3)*W+D2)*W+D1)*W+D0)
      GO TO 500
C -- RATIONAL APPROXIMATION FOR 4<X<170
300   IF ( X .GT. 170.0D0 ) GO TO 400
      EONEX=W*Y*(1.D0+W*(((((((E7*W+E6)*W+E5)*W+E4)
     1      *W+E3)*W+E2)*W+E1)*W+E0)/(((((((F7*W+F6)*W
     2      +F5)*W+F4)*W+F3)*W+F2)*W+F1)*W+F0)  )
      GO TO 500
400   EONEX = 0.0
500   ALLOC(LOCANS+INDEX-1) = EONEX
      E1LINK = 1
      RETURN
C ----- ERROR RETURNS
C -- TOO MANY ARGUMENTS
9001  ICOL = -24
      RETURN
C -- ARGUMENT OUTSIDE ALLOWED BOUNDS
9002  ICOL = -43
      RETURN
C -- ARGUMENT IS NOT DEFINED
9003  ICOM=-1
      RETURN
C -- COMPLEX ARGUMENT NOT ALLOWED
9004  ICOL = -31
      RETURN
      END
/*
//EDT.SYSLIB DD DISP=SHR,DSN=CONS.LOAD
//EDT.SYSPVT DD DISP=OLD,DSN=PHYSICS.LOAD,UNIT=2314,VOL=SER=DISK57
//EDT.SYSIN DD *
 ENTRY E1LINK
 NAME EONE(R)
```

## II. ONE GENERALIZED LINKULE INTERFACE

by

S. Cohen, R. N. Kimmel, and F. J. D. Serduke

Only moderate effort is required for a skilled FORTRAN programmer to learn the technique of interfacing functions to SPEAKEASY, but even this effort can be substantially reduced for certain classes of FORTRAN programs by writing interface routines. The following is a description of a LINKULE interface for the class of FORTRAN routines satisfying the following restrictions. (1) They are FUNCTION subprograms, i.e., they return a single number. (2) All arguments in the calling sequence are scalars. No dimensioned variable may appear in the argument list. (3) All transfer of information is through the argument list; no information is transferred through COMMON areas.

If all of these requirements are satisfied, an interface to such a function is almost trivial. An example of such an interface is given below. This small interface routine together with the FORTRAN program in question are compiled, link-edited with another special routine, and then saved as a LINKULE in an appropriate library. Many commonly used functions satisfy these restrictions and consequently are simple and straightforward to include in SPEAKEASY. In addition to saving labor, this process has the advantage of automatically providing the standard SPEAKEASY conventions for the handling of structured arguments that may be used in the SPEAKEASY call.

This standard treatment of structured arguments is called the HIGH-WIDE convention; it is easy for SPEAKEASY users to remember and permits the user to avoid the use of loops in calculation involving such functions. This HIGH-WIDE convention relates to the forms of acceptable combinations of the structured arguments involved in the SPEAKEASY call and the definition of the structure of the answer.

If the SPEAKEASY statement

is to invoke a LINKULE that follows the HIGH-WIDE convention, the arguments in the call are to be considered "compatible" only if (1) the height (number of rows) of each object in the argument list is either 1 or some constant value NHIGH and (2) the width (number of columns) of each object in the argument list is either 1 or some constant value NWIDE.

The resulting object defined by the statement will be an object with NHIGH rows and NWIDE columns. The values of its elements will be those corresponding to a repeated call to the FORTRAN routine with combinations of arguments generated by expanding each argument, if needed, into an NHIGH-by-NWIDE object. The structure of the resulting object is determined by the following rules. (1) If all objects are scalars, the answer is a scalar. (2) If no two-dimensional objects appear but at least one one-dimensional object appears in the argument list, the answer will be one-dimensional. Furthermore, one-dimensional arrays take precedence over vectors in the determination of the structure of the result. (3) If any two-dimensional object appears in the argument list, the answer will be of the same structure; and again two-dimensional arrays take precedence over matrices. (4) If both one- and two-dimensional arguments appear, the one-dimensional arguments are treated as column-like objects. The structure of the result will be two-dimensional and will follow rule 3.

These rules may be summarized by saying that (a) the two-dimensional form takes precedence over the one-dimensional, (b) the one-dimensional form takes precedence over the scalar, and (c) for the same number of dimensions, the array form takes precedence over a matrix or vector form.

The treatment of a given object in an argument list depends on the structure of other arguments; the "effective representation" of an argument may be viewed as an expansion of its structure so that all objects in a particular argument list have the same structure. Assume that the result is to be a HIGH-by-WIDE array. Then (1) a scalar argument is expanded into a HIGH-by-WIDE array with each element equal to that scalar, (2) a column array (or matrix) is expanded into a HIGH-by-WIDE array (or matrix) with all columns equal (and, for purposes of this expansion, the convention is that one-dimensional arrays and vectors are treated as columnar structures), (3) a row array (or matrix) is expanded into a HIGH-by-WIDE array (or matrix) with all rows equal, and (4) a two-dimensional HIGH-by-WIDE object is its own effective representation.

As a specific example, let HIGH = 2 and WIDE = 3. Then the effective representation for the scalar 8 is

$$8 \rightarrow \begin{pmatrix} 8 & 8 & 8 \\ 8 & 8 & 8 \end{pmatrix},$$

that of a column array (or 1-dimensional array) is

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix},$$

that of a row array is

$$(2\ 4\ 6) \rightarrow \begin{pmatrix} 2 & 4 & 6 \\ 2 & 4 & 6 \end{pmatrix}$$

and that of a 2-dimensional array is

$$\begin{pmatrix} 1 & 3 & 5 \\ 4 & 2 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 3 & 5 \\ 4 & 2 & 0 \end{pmatrix}.$$

As an example of a generalized linkule interface, suppose BUDDY is the name of a FORTRAN function routine whose calling sequence in FORTRAN is

```
X = BUDDY (A, B, I)
```

where A, B, and I are all numbers. Then this routine can be interfaced to SPEAKEASY by the technique shown below and can be assigned any name as its SPEAKEASY representation. Let us choose BUD as this name. Then the SPEAKEASY statement

X = BUD (3, 5, 9)

will function in exactly the same manner as the FORTRAN statement. (Only scalar arguments are used in both cases.) Similarly, the sequence

A = ROWARRAY (2 : 1, 2); B = COLARRAY (3 : 5, 6, 7); X=BUD (A, B, 8)

is equivalent to

$$X = \text{BUD}\left[\begin{pmatrix}1 & 2\\ 1 & 2\\ 1 & 2\end{pmatrix}, \begin{pmatrix}5 & 5\\ 6 & 6\\ 7 & 7\end{pmatrix}, \begin{pmatrix}8 & 8\\ 8 & 8\\ 8 & 8\end{pmatrix}\right]$$

where X will be a two-dimensional array whose elements are

$$\begin{bmatrix}\text{BUD (1, 5, 8)} & \text{BUD (2, 5, 8)}\\ \text{BUD (1, 6, 8)} & \text{BUD (2, 6, 8)}\\ \text{BUD (1, 7, 8)} & \text{BUD (2, 7, 8)}\end{bmatrix}$$

How do we actually write the interface to the FORTRAN program BUDDY and create the SPEAKEASY linkule BUD? First we write a FORTRAN program with two entries: FLINK and NUMARG. The entry NUMARG returns the valid number of arguments in the SPEAKEASY call and FLINK returns the functional value for a given set of input parameters. The floating-point and integer values of these parameters come in through the arrays X and IX. For example:

```
      FUNCTION FLINK(X,IX)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(1),IX(1)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C  X(I) AND IX(I) ARE THE FLOATING-POINT AND INTEGER
C  VALUES RESPECTIVELY OF THE I-TH ARGUMENT IN THE
C  SPEAKEASY CALL
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      FLINK = BUDDY(X(1),X(2),IX(3))
      RETURN
      ENTRY NUMARG(IDUMMY)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C  THIS ENTRY RETURNS THE VALID NUMBER OF ARGUMENTS
C  IN THE SPEAKEASY CALL
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      NUMARG = 3
      RETURN
      END
```

This routine is to be compiled with the user subroutine BUDDY. It must be link-edited with a routine called LINKIT; the resulting load module is to be saved in an appropriate library and its member name in the library is the one assigned to the SPEAKEASY operator—in this case, BUD.

Appendix I shows the form of a job to be used to create such a LINKULE in the Argonne Physics Library. This is particularly simple because of the SOS capabilities in the Argonne system. New linkules should be created in a private library such as the Physics library and fully debugged there. Before such a linkule is introduced into the communal library, it must be validated and documented.

Appendix II is a listing of the source deck for the generalized interface routine LINKIT. This routine is contained in the library CONS.LOAD, which normally is available at SPEAKEASY installations; or its source deck can be reproduced from this listing.

Appendix III is a complete listing of a job specification to create the linkule for the evaluation of $e^{-AX}$*SIN(M*X) as a function of A, M, and X. This function has been named FUN(A, M, X) for SPEAKEASY use.

Appendix IV shows the deck used to operate with the linkule created by the job shown in Appendix III. Note that the private library PHYSICS. LOAD has been attached to the system for this run by means of the //LIB card and the card defining LINKLIB in the SPEAKEASY run.

Appendix V is the output from a SPEAKEASY run using the indicated function in a variety of ways. This is the result of the job shown in Appendix IV.

## APPENDIX I. Job Specification to Create a New Linkule

A job specification to create a new linkule at Argonne should be in the form illustrated below. This linkule is created in the Physics library (not the SPEAKEASY link library). New linkules should not be placed in the link library until they are validated and documented. The job-control language for this run is particularly simple because of the SOS capabilities available in the Argonne system.

```
//MYJOB JOB (F88888,1,1,1),CLASS=A,REGION=200K
// EXEC SOS
//KEEP DD DSN=PHYSICS.LOAD,DISP=OLD
//SYSLIB DD DSN=CONS.LOAD,DISP=SHR
/COMPILE=H

---->  FORTRAN source for the FLINK program goes here

---->  FORTRAN source for the BUDDY program goes here

/KEEP BUD 'MAP,LIST,LET,REUS'
 INCLUDE SYSLIB(LINKIT)
 ENTRY LINKIT
```

## APPENDIX II. FORTRAN Source Deck for LINKIT

The FORTRAN source deck for the generalized interface routine LINKIT is listed in this appendix. It is applicable to SPEAKEASY linkules that are scalar functions of scalar variables. The HIGH-WIDE argument convention is built into this interface.

```
      FUNCTION LINKIT
     1(ANS,ITH,NOARGS,ICOL,ICOM,IDOM,ACC,ARG,PARAM,PARAMI,IPARAM,
     2KIND,KLAS,NHIGH,NWIDE,NOELS,LOC,ALLOC,ICLRES,IQUERY,IFREE)
C
      IMPLICIT REAL*8 (A-H,O-Z)
C -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
C   SPEAKEASY GENERALIZED INTERFACE FOR LINKULES THAT ARE
C   SCALAR FUNCTIONS OF SCALAR VARIABLES.  THE HIGH-WIDE
C   ARGUMENT CONVENTION IS BUILT INTO THIS LINKULE.
C     THIS ROUTINE IS USED BY WRITING A SUBROUTINE WITH
C   TWO ENTRIES  (1) FLINK(X,IX)   AND (2) NUMARG(IDUMMY)
C   NUMARG RETURNS THE NUMBER OF ARGUMENTS IN THE SPEAKEZ WORD
C   FLINK  IS A FUNCTIONAL ENTRY THAT RETURNS THE FUNCTION TO
C          BE EVALUATED AS A FUNCTION OF THE SPEAKEZ ARGS THAT
C          ARE CONTAINED IN THE ARRAYS  X  AND  IX .
C -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
C
      DIMENSION ANS(1),PARAM(1),PARAMI(1),IPARAM(1),ARG(1),KIND(1),
     1          KLAS(1),NHIGH(1),NWIDE(1),NOELS(1),LOC(1),ALLOC(1)
C
      INTEGER MAXSIZ/32000/
C     -----------------------------------------------------
      INTEGER*4 ANKLAS,HEIGHT,WIDTH,TALL(30),FAT(30)
C     -----------------------------------------------------
C
         LINKIT=0
         HEIGHT=1
         WIDTH=1
         ANKLAS=0
         INDEXA=0
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C   PHASE 0: IS THE NUMBER OF ARGUMENTS IN THE SPEAKEZ CALL CORRECT?
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      IF ( NOARGS .NE. NUMARG(0) ) GO TO 9003
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C   PHASE I:  DETERMINATION OF THE STRUCTURE OF THE ANSWER
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      DO 5 N=1,NOARGS
C
C          FIRST TEST IF N-TH ARGUMENT IS DEFINED
      IF ( LOC(N).EQ.0 .AND. KIND(N).GE.0 ) GO TO 9004
C
C         SET UP SIZE AND DETERMINE CLASS OF ANSWER
      IF ( NWIDE(N) .GT. WIDTH )   WIDTH = NWIDE(N)
      IF ( NHIGH(N) .GT. HEIGHT)   HEIGHT= NHIGH(N)
      IF ( KLAS(N)  .GT. ANKLAS)   ANKLAS= KLAS(N)
C
C          SET ROW AND COLUMN LOOP SWITCHES
      FAT(N) = 0
      IF ( NWIDE(N) .GT. 1 ) FAT(N) = 1
      TALL(N) = 0
      IF ( NHIGH(N) .GT. 1 ) TALL(N) = 1
C
    5 CONTINUE
C
```

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C   PHASE II:  ERROR CHECKS
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C       TEST IF THE RESULTING STRUCTURED OBJECT WILL BE TOO LARGE
      IF ( HEIGHT*WIDTH .GT. MAXSIZ ) GO TO 9002
C
      DO 10 N=1,NOARGS
C
C        IS THE N-TH ARGUMENT ANYTHING BUT A REAL NUMBER?
      IF ( IABS(KIND(N)) .NE. 2 ) GO TO 9000
C        TEST FOR INCOMPATIBLE DIMENSIONS
C           BUT BYPASS TEST FOR AN IN-PLACE DEFINITION
            IF(KIND(N).LT.0) GO TO 10
      IF(NWIDE(N).NE.1.AND.NWIDE(N).NE.WIDTH)  GO TO 9001
      IF(NHIGH(N).NE.1.AND.NHIGH(N).NE.HEIGHT) GO TO 9001
C
C       TEST IF BOTH MATRIX AND 1-D ARRAY INPUT
C            OUTPUT WILL BE A MATRIX
      IF ( ANKLAS.EQ.5  .AND.  KLAS(N).EQ.2 ) ANKLAS = 2
C
   10 CONTINUE
C
C-------------------------------------------------------------------
C
C  RESERVE ALLOCATOR SPACE AND DEFINE THE STRUCTURE OF THE ANSWER
      LOCANS = ICLRES ( ANS , 2 , ANKLAS , HEIGHT , WIDTH )
C
C-------------------------------------------------------------------
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C   PHASE III:  DECOMPOSITION OF STRUCTURED INPUT FOR FORTRAN CALL
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      DO 20 I=1,HEIGHT
      DO 20 J=1,WIDTH
          DO 15 N=1,NOARGS
C
C        IF N-TH ARGUMENT IS A SCALAR OR AN IN-PLACE DEFINITION
C        BYPASS INDEXING.....
             IF(KIND(N).LE.0) GO TO 15
C
C        HERE DEAL WITH STRUCTURED ARGUMENT
      INDEX=(J-1)*FAT(N)+(I-1)*TALL(N)*NWIDE(N)
      PARAM(N)=ALLOC(LOC(N)+INDEX)
C
   15     CONTINUE
C
C        IN CASE INTEGER ARGUMENTS ARE NEEDED
      DO 16 N=1,NOARGS
   16 IPARAM(N)=PARAM(N)+DSIGN(ACC,PARAM(N))
```

```
C
C     *****************************************************
C
C           CALL TO THE USER-WRITTEN FUNCTION 'FLINK'
C           WITH THE REAL AND INTEGER VALUES OF THE
C           ARGUMENTS
C
         ALLOC(LOCANS+INDEXA) = FLINK(PARAM,IPARAM)
C     *****************************************************
C
         INDEXA = INDEXA+1
C
   20 CONTINUE
C
      LINKIT=1
C
   30 RETURN
C
C
C -----------------------------------------------------
C ERROR RETURNS
C
 9000 ICOL=-100
C     SOME ARGUMENT NOT A REAL NUMBER
      GO TO 30
C
 9001 ICOL=-102
C     INCOMPATIBLE DIMENSIONS
      GO TO 30
C
 9002 ICOL=-62
C  OVERSIZE OUTPUT...ALLOCATOR CAN'T HANDLE IT
      GO TO 30
C
 9003 ICOL=-30
C  INCORRECT NUMBER OF ARGUMENTS IN SPEAKEZ CALL
      GO TO 30
C
 9004 ICOM=-N
C  N-TH ARGUMENT IS NOT DEFINED
      GO TO 30
C
C -----------------------------------------------------
C
      END
```

## APPENDIX III. Job Specification Submitted to Argonne Computer

The new linkule called FUN was introduced into the private library PHYSICS. LOAD. This linkule was designed to evaluate $e^{-AX}$ sin(MX) and to act according to the generalized rules described in this report.

```
//MYJOB JOB (F88888,1,1,1),CLASS=A,REGION=240K
/*PROCESS MAIN
/*PROCESS RSOUT10
//   EXEC SOS
//KEEP DD DSN=PHYSICS.LOAD,DISP=OLD
//SYSLIB DD DSN=CONS.LOAD,DISP=SHR
/COMPILE=H
      FUNCTION FLINK(X,IX)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(1),IX(1)
      FLINK=EVAL(X(1),IX(2),X(3))
      RETURN
      ENTRY NUMARG(IDUMMY)
      NUMARG=3
      RETURN
      END
      FUNCTION EVAL (A,M,X)
      IMPLICIT REAL*8 (A-H,O-Z)
      EVAL=DEXP(-A*X)*DSIN(M*X)
      RETURN
      END
/KEEP FUN 'MAP,LIST,LET,REUS'
 INCLUDE SYSLIB(LINKIT)
 ENTRY LINKIT
```

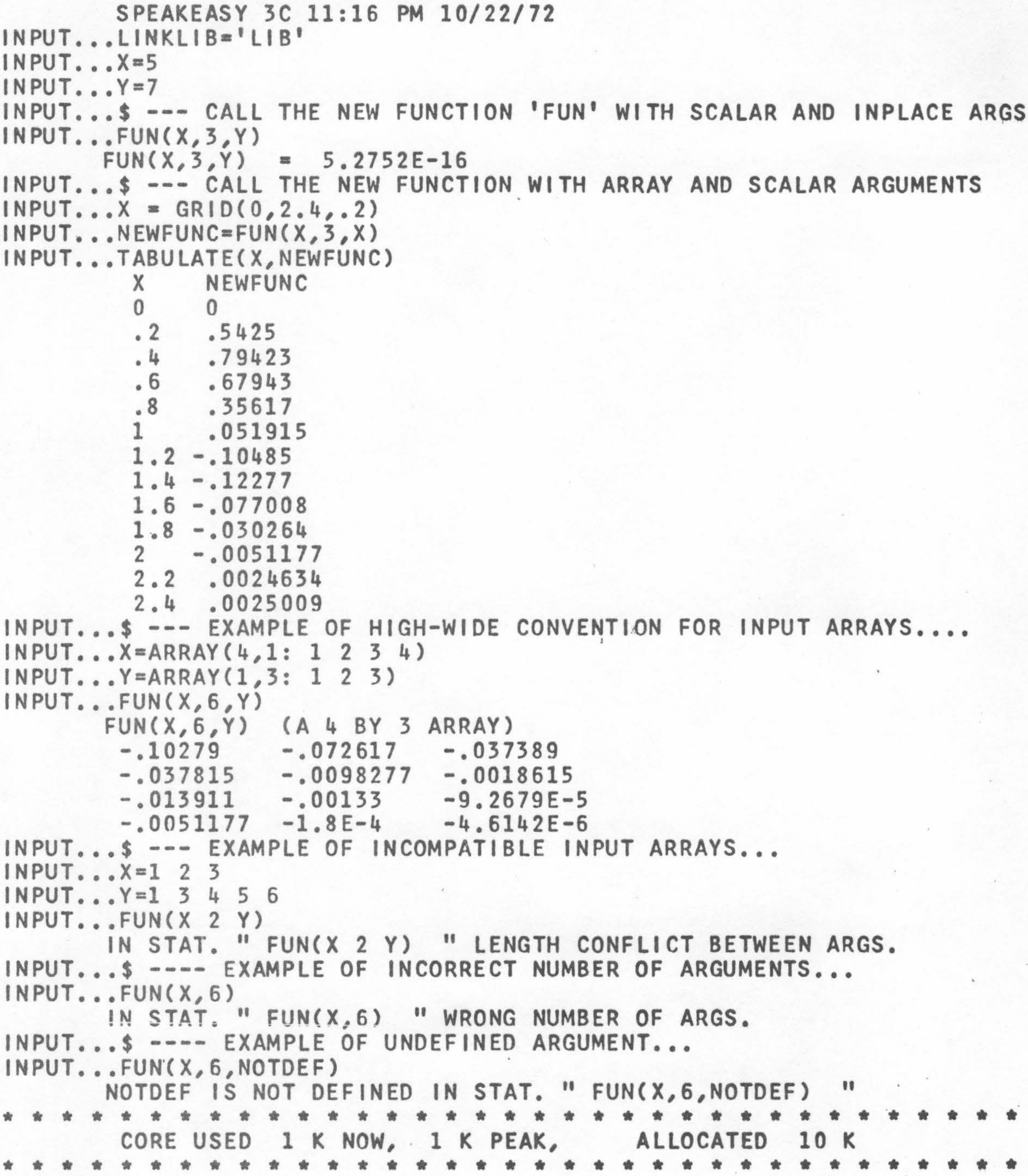
## APPENDIX IV. Deck to Operate with Routine FUN

The following is the complete deck to run SPEAKEASY and to use the linkule FUN created in Appendix III. Note the use of the private library.

```
//MYSPEAK JOB (F88888,1,1,1),CLASS=C,REGION=220K,PRTY=L
/*PROCESS MAIN
/*PROCESS RSOUT10
//   EXEC SPEAKEZ
//LIB DD DSN=PHYSICS.LOAD,DISP=SHR
SIZE=10
LINKLIB='LIB'
X=5
Y=7
$ --- CALL THE NEW FUNCTION 'FUN' WITH SCALAR AND INPLACE ARGS...
FUN(X,3,Y)
$ --- CALL THE NEW FUNCTION WITH ARRAY AND SCALAR ARGUMENTS
X = GRID(0,2.4,.2)
NEWFUNC=FUN(X,3,X)
TABULATE(X,NEWFUNC)
$ --- EXAMPLE OF HIGH-WIDE CONVENTION FOR INPUT ARRAYS....
X=ARRAY(4,1: 1 2 3 4)
Y=ARRAY(1,3: 1 2 3)
FUN(X,6,Y)
$ --- EXAMPLE OF INCOMPATIBLE INPUT ARRAYS...
X=1 2 3
Y=1 3 4 5 6
FUN(X 2 Y)
$ ---- EXAMPLE OF INCORRECT NUMBER OF ARGUMENTS...
FUN(X,6)
$ ---- EXAMPLE OF UNDEFINED ARGUMENT...
FUN(X,6,NOTDEF)
```

## APPENDIX V. Output of Run with Linkule FUN

The output generated by the deck listed in Appendix IV is shown below. This illustrates the variety of ways in which the linkule can be used and shows the automatic built-in error messages that result from using the standard interface.



```
       SPEAKEASY 3C 11:16 PM 10/22/72
INPUT...LINKLIB='LIB'
INPUT...X=5
INPUT...Y=7
INPUT...$ --- CALL THE NEW FUNCTION 'FUN' WITH SCALAR AND INPLACE ARGS...
INPUT...FUN(X,3,Y)
      FUN(X,3,Y)  =  5.2752E-16
INPUT...$ --- CALL THE NEW FUNCTION WITH ARRAY AND SCALAR ARGUMENTS
INPUT...X = GRID(0,2.4,.2)
INPUT...NEWFUNC=FUN(X,3,X)
INPUT...TABULATE(X,NEWFUNC)
        X    NEWFUNC
        0    0
        .2   .5425
        .4   .79423
        .6   .67943
        .8   .35617
        1    .051915
        1.2 -.10485
        1.4 -.12277
        1.6 -.077008
        1.8 -.030264
        2   -.0051177
        2.2  .0024634
        2.4  .0025009
INPUT...$ --- EXAMPLE OF HIGH-WIDE CONVENTION FOR INPUT ARRAYS....
INPUT...X=ARRAY(4,1: 1 2 3 4)
INPUT...Y=ARRAY(1,3: 1 2 3)
INPUT...FUN(X,6,Y)
      FUN(X,6,Y)  (A 4 BY 3 ARRAY)
       -.10279    -.072617   -.037389
       -.037815   -.0098277  -.0018615
       -.013911   -.00133    -9.2679E-5
       -.0051177  -1.8E-4    -4.6142E-6
INPUT...$ --- EXAMPLE OF INCOMPATIBLE INPUT ARRAYS...
INPUT...X=1 2 3
INPUT...Y=1 3 4 5 6
INPUT...FUN(X 2 Y)
      IN STAT. " FUN(X 2 Y)  " LENGTH CONFLICT BETWEEN ARGS.
INPUT...$ ---- EXAMPLE OF INCORRECT NUMBER OF ARGUMENTS...
INPUT...FUN(X,6)
      IN STAT. " FUN(X,6)  " WRONG NUMBER OF ARGS.
INPUT...$ ---- EXAMPLE OF UNDEFINED ARGUMENT...
INPUT...FUN(X,6,NOTDEF)
      NOTDEF IS NOT DEFINED IN STAT. " FUN(X,6,NOTDEF)  "
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
       CORE USED  1 K NOW,  1 K PEAK,     ALLOCATED  10 K
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

# PART FOUR

## The SPEAKEASY HELP Documents

by

J. K. Fink

Preceding page blank

## I. INTRODUCTION

The user of an interactive language such as TSO SPEAKEASY needs a means of quickly determining the operations available to him and of easily learning to use particular features. To fill this need, a special library named HELP has been created and attached to all SPEAKEASY processors. The HELP library consists of concise documents that describe each word of the SPEAKEASY language. The interactive user may obtain each document by a simple command from his terminal.

The growth capability of SPEAKEASY has created difficulties in formally documenting all new features as rapidly as they are added. Since the HELP documents are available to all SPEAKEASY processors and thus readily accessible to all users, they have become the primary method of documentation of new features. This report collects the currently available SPEAKEASY documents in easily read form.

Sec. II consists of an index to the words described in the HELP documents. Each entry in the HELP library is given a reference name consisting of eight characters. The reference name is usually identical to the name used in the SPEAKEASY processor. If the SPEAKEASY word being described is more than eight characters in length, the reference name consists of the first eight characters.

Short one-line definitions of the words in the HELP library are given in Sec. III. These one-line definitions are the first line of each HELP document. Each line contains the word being defined, its argument(s), and a brief definition.

The organization of the HELP documents is described in Sec. IV. It is a tree structure designed so that the user can easily learn the SPEAKEASY words related to a specific topic. The "trunk" of the tree, shown at the left in a diagram at the beginning of the section, consists of the word HELP. HELP refers the user to the first levels of classification, which form the five branches of the tree and also refers the user to the tutorial to learn how to use SPEAKEASY, to NEWS which lists new features, to BUGS which gives current errors, and tells the user how to leave the SPEAKEASY processor. The document for HELP is given alone at the beginning of the section.

The HELP tree branches into five main classes. Each of these five classes branches into subclasses (one group of five subclasses branches into further subclasses) and then the branching terminates in the SPEAKEASY words. The five major classes are INOUT, MATH, MISCELLANEOUS, OBJECTS, and PROGRAMS. The listing of the document for each class is followed immediately by the documents for the subclasses under it, etc. Each branch is followed to completion before a new branch is begun. The beginning of each new branch is on a new page so it is easy to locate major classifications and to quickly learn the subclasses.

All classes, subclasses, and SPEAKEASY words are listed in alphabetical order to facilitate finding a particular class, subclass, or word. Each subclass and SPEAKEASY word is explained briefly unless the meaning is obvious. If the user desires further information about a particular word, he can refer to the specific HELP document.

The last section consists of the HELP documents themselves. Each document begins with the word being described and gives a concise definition of the word, including all possible SPEAKEASY calls. The word being defined and its arguments are capitalized throughout the document so that they may be easily located. Most documents are structured in levels; each paragraph gives greater detail than its predecessor so an interactive user can halt the printing of a document at any point by hitting the break key (the attention key on an IBM-2741 terminal). While the documents are brief and do not go into great detail in describing a word, they give sufficient information to enable a user who is unfamiliar with a feature to employ that feature after reading the appropriate HELP document. For the user interested in further information, the HELP documents refer to longer and more detailed documents if such documentation exists and is readily available.

In the program that generated the computer printout of these documents, the subprogram PRREAD, written by Dr. Steven Pieper, was used. In addition, Dr. Stanley Cohen gave many helpful suggestions which were used in writing the HELP documents.

## II. SPEAKEASY WORDS DEFINED IN THE HELP DOCUMENTS

```
There are 300 HELP documents

 ABS        CONSTRAI   ENDAUTOP   INTEG      MATH       OMITCLAS   ROWMAT     SYMMAT
 ACCURACY   CONTINUE   ENDLOOP    INTEGERS   MATRICES   ONEDIMFU   ROWMAX     TABULATE
 ACOS       CONVERT    EONE       INTEGRAL   MATRIX     ONERROR    ROWMIN     TAN
 ACOT       COPY       EQ         INTEGRAT   MATRIXDE   OR         RUN        TIME
 ADDGRAPH   COS        ERF        INTERP     MATRIXOP   ORDERED    SELECT     TOTALINT
 ADJOINT    COSH       ERFC       INTERPOL   MAX        ORDERER    SETGAUSS   TOTINT
 AFAM       COT        ERRORS     INTPART    MAXOFCOL   OTHERS     SETINFIN   TRACE
 AMAT       CREATE     EXECUTE    INTS       MAXOFROW   OUTPUT     SETJACOB   TRANSFAM
 AND        CREATEME   EXP        INT2       MELD       PAUSE      SETLAGUE   TRANSP
 ANGLES     CUMPROD    FIN        INT4       MFAM       PLOTSYMB   SETLEGEN   TRANSPOS
 ARRAY      CUMSUM     FOR        INVERSE    MIN        PLOTTITL   SETLIB     TRIG
 ARRAYS1    DATA       FRAC       KEEP       MINOFCOL   PRINT      SETNULL    TUTORIAL
 ARRAYS2    DEBUGGIN   FRACPART   KEPT       MINOFROW   PRINTCLA   SETPLOT    TWODIMFU
 ARRAY2D    DEC        FREE       LABEL      MISCELLA   PROCLIB    SIGN       UMAT
 ASIN       DEFINEA1   GAMMA      LE         MOVE       PROD       SIGNIFIC   UNITMAT
 ASYMMAT    DEFINEA2   GE         LENGTH     MYDOCS     PRODCOLS   SIMEQ      UPPERTRI
 ATAN       DELETE     GEIGEN     LIBINDEX   MYHELP     PRODROWS   SIN        USE
 AUTOCORE   DERIV      GO         LIBNAMES   MYKEEP     PRODUCTS   SINGLEVA   USEMEMBE
 AUTOPRIN   DERIVATI   GOTO       LIBRARIE   MYKEPT     PROGRAM    SINH       VARIABLE
 AUTOTAB    DET        GRAPH      LIBRARYN   MYLINKS    PROGRAMM   SIZE       VEC
 AVERAGE    DIAGELS    GRAPHICS   LINKLIB    MYPROCS    PROGRAMS   SMAT       VECTOR
 A1D        DIAGMAT    GRID       LINKULES   NAMES      PUNCH      SORT       VECTORDE
 A2D        DMAT       GT         LIST       NE         QUIT       SPACE      VECTORS
 BESSEL     DOCUMENT   HELP       LISTHEAD   NEUMANN    RANDOM     SPECIAL    VERSIONS
 BESSELK    DOMAIN     HENCEFOR   LISTMEMB   NEWGRAPH   RANKED     SPHBES     VFAM
 BUGS       DONTLIST   HIERARCH   LISTPROG   NEWPAGE    RANKER     SPHBESN    VLABEL
 CGAMMA     DOUBLEPA   HIGHWIDE   LOADDATA   NEWS       RATIONAL   SQRT       VOCABULA
 CLEAR      DUMP       HIWIDE     LOC        NOCOLS     READ       STOP       VSCALE
 CLEARDAT   ECHO       HLABEL     LOCMAX     NOECHO     REAL       STRUCTUR   VSIZE
 COLARRAY   EDIT       HSCALE     LOCMIN     NOELS      REALPART   SUM        WHERE
 COLMAT     EDITMODE   HSIZE      LOCS       NORATION   REAL4      SUMCOLS    WHEREVER
 COLMAX     EIGENSYS   IF         LOG        NOROOTS    REAL8      SUMPROD    WHOLE
 COLMIN     EIGENVAL   IMAG       LOGGAMMA   NOROWS     RECLASS    SUMROWS    WRITE
 COLWIDTH   EIGENVEC   IMAGPART   LOGIC      NOT        RESTRICT   SUMS       ZEROS
 COMMANDS   ELEMENTA   INOUT      LOWERTRI   NOZEROS    RESUME     SUMSQ
 COMPILE    ELLIPF     INPUT      LT         NUMBERS    RETURN     SUMSQCOL
 CONJ       ELLIPK     INPUTS     MARGINS    OBJECT     ROOTS      SUMSQROW
 CONJUGAT   END        INSERT     MAT        OBJECTS    ROWARRAY   SYMBOLS
```

## III. ONE-LINE DEFINITIONS OF THE SPEAKEASY WORDS

Date: 5/25/73

ABS(X) returns the absolute value of X.
ACCURACY(VAL) specifies the smallest number not to be taken as zero.
ACOS(X) defines the arccosine of X.
ACOT(X) defines the arccotangent of X.
ADDGRAPH(I:J) plots I versus J on the previous graph.

ADJOINT(X) gives the adjoint of the matrix X.
AFAM(X) defines a member of the array family.
AMAT is a synonym for ASYMMAT.
.AND. is the logical operator "and".
ANGLES specifies whether the user is using radians or degrees.

ARRAY(N:) defines a 1-dimensional N-component array.
ARRAYS1 are words dealing with 1-dimensional arrays.
ARRAYS2 are words dealing with 2-dimensional arrays.
ARRAY2D(N,M:) defines a 2-dimensional N-by-M array.
ASIN(X) defines the arcsine of X.

ASYMMAT(N:I,J,...,K) defines an N-by-N antisymmetric matrix.
ATAN(X) defines the arctangent of X.
AUTOCORE specifies that the data storage space is in LCS.
AUTOPRINT(X,Y,...,Z) prints X,Y,...,Z each time they are redefined.
AUTOTAB specifies uniform column width for printout.

AVERAGE(X) returns the average value of the elements of X.
A1D(N:) defines a 1-dimensional array.
A2D is a synonym for ARRAY2D.
BESSEL(NU,X) calculates cylindrical Bessel fn. of the first kind.
BESSELK(NU,X) returns the modified Bessel function K of X.

BUGS are the current bugs in SPEAKEASY.
CGAMMA(X) returns the gamma function of X (X is complex).
CLEAR erases the entire active data storage.
CLEARDATA frees all previously defined numerical data.
COLARRAY(N:) defines a 2-dim. N-component column array.

COLMAT(N:) defines an N-by-1 matrix which is a column.
COLMAX(X) specifies the column with the maximum element of X.
COLMIN(X) specifies the column with the minimum element of X.
COLWIDTH(N) specifies the minimum column width to be used in printing.
COMMANDS are words that are commands in SPEAKEASY.

```
COMPILE returns the processor to the MANUAL mode from EDIT.
CONJ is a synonym for CONJUGATE.
CONJUGATE(X) returns the complex conjugate of X.
CONSTRAIN(A,B,...,C:X) constrains A,B,...,C according to condition X.
CONTINUE used in the program mode causes no operation to be performed.


CONVERT are words used for numerical conversions.
COPY N,M,K(I) copies statements N through M to K,K+I,K+2I,...
COS(X) returns the cosine of X.
COSH(X) defines the hyperbolic cosine of X.
COT(X) returns the cotangent of X.


CREATE MEMBER X ON LIBRARY Y creates a new member in a library.
CREATEMEMBER(XX,YY) creates member XX in library YY.
CUMPROD(X) defines the cumulative product of the elements of X.
CUMSUM(X) defines the cumulative sum of the elements of X.
DATA NAME marks the begining of a data file.


DEBUGGING are words useful in debugging a program.
DEC(ZA1,ZA2,...,ZAN) converts A1,A2,...,AN from hex to decimal.
DEFINEA1 are words used in defining 1-dimensional arrays.
DEFINEA2 are words used in defining 2-dimensional arrays.
DELETE N,M deletes the statements N through M.


DERIV is a synonym for DERIVATIVE.
DERIVATIVE(F:X) finds the derivative of F with respect to X.
DET(X) defines the determinant of the matrix X.
DIAGELS(X) selects the diagonal elements of X.
DIAGMAT(N:I,J,...,K) defines an N-by-N diagonal matrix.


DMAT is a synonym for DIAGMAT.
DOCUMENT is a library of larger documents.
DOMAIN specifies whether real or complex numbers are being used.
DONTLIST indicates that the program is not to be printed.
DOUBLEFACTORIAL(X) returns the doublefactorial X!!


DUMP creates an easily read complete printout of all defined objects.
ECHO prints out the input along with results.
EDIT is a mode available for editing a program.
EDITMODE are words used in the EDIT mode of TSO SPEAKEASY.
EIGENSYSTEM(SYMMAT) returns all the eigenvalues of SYMMAT.
```

EIGENVALS(X) gives the eigenvalues of the matrix X.
EIGENVECS(X) gives the eigenvectors of the matrix X.
ELEMENTAL are elemental mathematical functions.
ELLIPE(X) calculates the complete elliptical integral E(X).
ELLIPK(X) calculates the complete elliptical integral K(X).

END terminates a SPEAKEASY program or datafile.
ENDAUTOPRINT turns off AUTOPRINT.
ENDLOOP N marks the end of the FOR loop N.
EONE(X) calculates the exponential integral from X to infinity.
.EQ. is the relational operator "equal to".

ERF(X) calculates the error function of X.
ERFC(X) calculates the complementary error function of X.
ERRORS are ambiguities making it impossible to carry out a statement.
EXECUTE NAME executes the stored program, NAME.
EXP(X) defines the exponential function.

FIN is a synonym for ENDLOOP.
FOR N=START,STOP,INC designates a loop in a program.
FRAC is a synonym for FRACPART.
FRACPART(X) returns the fractional part of X.
FREE(N1,N2,...,NN) frees the definitions of the objects N1,N2,...,NN.

GAMMA(X) defines the gamma function of X.
.GE. is the relational operator "greater than or equal to".
GEIGEN(M,METRIC) finds the eigenvalues of the matrix M.
GO causes execution of a program to resume from the holding mode.
GOTO X causes execution to be transferred to the statement labeled X.

GRAPH(I:J) plots the members of I versus the object J.
GRAPHICS are words for graphical output that are in version GRAPHEZ.
GRID(I,J) defines a 1-dim. array of 101 points from I to J.
.GT. is the relational operator "greater than".
HELP explains how to use the HELP processor.

HENCEFORTH X IS Y redefines the word X to mean Y.
HIERARCHY is the order in which mathematical operations are done.
HIGHWIDE activates HIGH-WIDE arithmetic for arrays.
HIWIDE is a synonym for HIGHWIDE.
HLABEL='ANY MESSAGE' labels the horizontal scale of a graph.

```
HSCALE=(LEFT,RIGHT) specifies the horizontal limits of a graph.
HSIZE=X specifies the horizontal size of the graph.
IF (X) Y is a conditional statement for scalar operations.
IMAG(X) returns the imaginary part of X.
IMAGPART(X) returns the imaginary part of X.


INOUT are classes of words dealing with input and output.
INPUT A,B,...,C puts the system in a holding mode.
INPUTS are words used for input of data or programs.
INSERT N(I) inserts the statements that follow at N,N+I,N+2I,...
INTEG is a synonym for INTEGRAL.


INTEGERS(I,J,K) defines an array with the integers I,I+K,I+2K,...,J.
INTEGRAL(F:X) defines the integral of F with a variable upper limit.
INTEGRATION are words dealing with numerical integration.
INTERP is a synonym for INTERPOL.
INTERPOL(Y,F,X) defines the function F at the points Y.


INTPART(X) returns the integer part of X.
INTS is a synonym for INTEGERS.
INT2(X) is a function that returns an integer2 object.
INT4(X) is a function that returns an integer4 object.
INVERSE(X) defines the inverse of the matrix X.


KEEP XX saves the SPEAKEASY object XX.
KEPT(XX) retrieves the SPEAKEASY object XX.
LABEL refers to a statement label.
.LE. is the relational operator "less than or equal to".
LENGTH(X) defines a scalar that is the number of elements in X.


LIBINDEX(Y) defines the names of the members of Y.
LIBNAMES returns the names of the SPEAKEASY libraries.
LIBRARIES are words for creating and using members of libraries.
LIBRARYNAMES are the names of the libraries attached to the processor.
LINKLIB='XX' adds the LINKULE library XX to those available.


LINKULES is a library of operations that are FORTRAN subroutines.
LIST MEMBER X OF LIBRARY Y FROM I TO J lists lines I through J OF X.
LISTHEAD(N,XX,YY) lists N lines of member XX of library YY.
LISTMEMBER(XX,YY) lists member XX of library YY.
LISTPROG specifies that the program is to be listed.
```

The First Lines Of The SPEAKEASY HELP Documents     Date:  5/25/73

```
LOADDATA(A,NAME) loads A with N values from the data file NAME.
LOC is a synonym for LOCS.
LOCMAX(X) specifies the location of the maximum element of X.
LOCMIN(X) specifies the location of the minimum element of X.
LOCS(X) gives the indices of the nonzero (true) elements of X.


LOG(X) returns the natural logarithm of X.
LOGGAMMA(X) defines the natural logarithm of the gamma function of X.
LOGIC are the logical, relational, and conditional operators.
LOWERTRI(X) returns the lower triangular part of X.
.LT. is the relational operator "less than".


MARGINS enables the user to contol the margins of the output.
MAT is a synonym for MATRIX.
MATH are classes of words dealing with mathematical functions.
MATRICES are classes of words dealing with matrices.
MATRIX(N,M:) defines an N-by-M matrix.


MATRIXDEF are words used in defining a matrix.
MATRIXOPS lists words that are matrix operators.
MAX(X) specifies the value of the maximum element in X.
MAXOFCOL(X) returns the largest element in each column of X.
MAXOFROW(X) returns the largest element in each row of X.


MELD(I,J,...,K) gives an odometer ordering of elements of I,J,...,K.
MFAM(X) defines a member of the matrix/vector family.
MIN(X) specifies the value of the minimum element in X.
MINOFCOL(X) returns the smallest element in each column of X.
MINOFROW(X) returns the smallest element in each row of X.


MISCELLANEOUS are classes of words not in any other classification.
MOVE N,M,K(I) moves statements N through M to K,K+I,K+2I,...
MYDOCS is the library name for private documents.
MYHELP is the library name for private HELP documents.
MYKEEP is a library for private information.


MYKEPT is a library for previously kept private information.
MYLINKS is the library name for private linkules.
MYPROCS is a private library.
NAMES prints the names of all currently defined SPEAKEASY objects.
.NE. is the relational operator "not equal to".
```

```
NEUMANN(NU,X) returns cylindrical Bessel fns. of the second kind.
NEWGRAPH causes the next graph to be drawn on a new area of paper.
NEWPAGE causes the printer to start a new page.
NEWS is information of current interest to users.
NOCOLS(X) defines a scalar which is the number of columns in X.


NOECHO suppresses the echoing of input data.
NOELS(X) defines a scalar equal to the number of elements in X.
NORATIONALIZE stops rationalization of subsequent output.
NOROOTS(F) gives the number of roots of the function F.
NOROWS(X) defines a scalar equal to the number of rows in X.


.NOT. is the logical operator "not".
NOZEROS(F) gives the number of zeros of the function F.
NUMBERS refers to numerical data.
OBJECT(I,J,...,K) generates an inplace name.
OBJECTS are classes of words dealing with structured objects.


OMITCLASS suppresses the printing of the class.
ONEDIMFUNCTIONS are words that deal with 1-dim. objects.
ONERROR(X,Y) specifies the action to be taken after an error.
.OR. is the logical operator "or".
ORDERED(X) gives the elements of X in increasing order.


ORDERER(X) gives the indices of the ordered elements of X.
OTHERS are miscellaneous words not under a specific classification.
OUTPUT are words related to output.
PAUSE puts the system in a holding mode.
PLOTSYMB(N,M) specifies the symbols used and frequency of points.


PLOTTITLE='ANY MESSAGE' titles a graph.
PRINT(A,B,C,...,Z) specifies that A,B,C,...,Z are to be printed.
PRINTCLASS prints the class of structured objects.
PROCLIB is a library of stored SPEAKEASY statements.
PROD(X) defines the product of the elements of X.


PRODCOLS(X) multiplies the elements in each column of X.
PRODROWS(X) multiplies the elements in each row of X.
PRODUCTS are words for products of elements of objects.
PROGRAM NAME gives a name to a program.
PROGRAMMODE are words restricted to the PROGRAM mode.
```

```
PROGRAMS are classes of words that are useful in writing programs.
PUNCH(F:X) punches the object X on cards in the format F.
QUIT terminates execution of a SPEAKEASY program.
RANDOM(X) generates random numbers.
RANKED(X) gives the elements of X in increasing order.

RANKER(X) gives the indices of the ordered elements of X.
RATIONALIZE (EPS, NDIGITS) causes rationalization of output.
READ(F:X) reads data from cards punched in format F and puts
REAL is a synonym for REALPART.
REALPART(X) returns the real part of X.

REAL4(X) is a function that returns a real4 number.
REAL8(X) is a function that returns a real8 number.
RECLASS(A:B,C,...,Z) alters the structure of B,C,...,Z.
RESTRICTED are the restricted words that cannot be names of variables.
RESUME causes execution of a program to resume from the holding mode.

RETURN returns execution to the program calling the stored program
ROOTS(F:X) finds the roots of the function F.
ROWARRAY(N:) defines a 2-dim. N-component array that is a row.
ROWMAT(N:) defines a 1-by-N matrix which is a row.
ROWMAX(X) specifies the row containing the maximum element of X.

ROWMIN(X) specifies the row containing the minimum element of X.
RUN is equivalent to COMPILE followed by the command EXECUTE.
SELECT(A,B,...,C:I) truncates or expands A,B,...,C using the index I.
SETGAUSS(N,X,W,XLO,XHI) returns Gauss-Legendre coords. and weights.
SETINFINITY(VAL) specifies an upper limit to the numbers printed.

SETJACOBI(N,X,W,A,B,TX,TW) defines Gauss-Jacobi coords. and weights.
SETLAGUERRE(N,X,W,A) returns Gauss-Laguerre coords. and weights.
SETLEGENDRE(N,X,W,XLO,XHI) returns Gauss-Legendre coord. and weights.
SETLIB(XX,YY) changes the name of library XX to YY.
SETNULL(VAL) specifies a lower limit to numbers printed.

SETPLOT(X,Y,Z) specifies BOX, NOBOX; SCALES, NOSCALES; LINES, POINTS.
SIGN(X) specifies whether X is positive or negative.
SIGNIFICANCE(N) gives the number of significant figures to be printed.
SIMEQ(A,B) solves a set of simultaneous linear equations.
SIN(X) returns the sine of X.
```

```
SINGLEVAR are operations on functions of one variable.
SINH(X) defines the hyperbolic sine of X.
SIZE=N,X specifies space for data and must be the first card.
SMAT is a synonym for SYMMAT.
SORT are sorting or ranking functions.

SPACE(N) skips N lines.
SPECIAL are special mathematical functions.
SPHBES(L,X) returns the spherical Bessel fn. of the first kind.
SPHBESN(L,X) returns the spherical Bessel fn. of the second kind.
SQRT(X) defines the square root of X.

STOP puts the system in the holding mode.
STRUCTURE are functions giving information on the structure of objects.
SUM(X) sums the elements of X.
SUMCOLS(X) sums the elements in each column of X.
SUMPROD are words for sums or products of elements of objects.

SUMROWS(X) sums the elements in each row of X.
SUMS are words for sums of elements of 2-dim. structured objects.
SUMSQ(X) sums the squares of the elements of X.
SUMSQCOLS(X) sums the squares of the elements in each column.
SUMSQROWS(X) sums the squares of the elements in each row.

SYMBOLS designate mathematical operations or special cards.
SYMMAT(N:I,J,...,K) defines a symmetric N-by-N matrix.
TABULATE(A,B,...,C) prints 1-dim. objects A,B,...,C in tabular form.
TAN(X) returns the tangent of X.
TIME gives the time in seconds from which one starts.

TOTALINT(F:X) defines the definite integral of F over the array X.
TOTINT is a synonym for TOTALINT.
TRACE(X) gives the trace of the matrix X.
TRANSFAMILY are functions to go from one family to another.
TRANSP is a synonym for TRANSPOSE.

TRANSPOSE(X) defines the transpose of X.
TRIG are the trigonometric functions.
TUTORIAL teaches you about SPEAKEASY.
TWODIMFUNCTIONS are words that deal with 2-dim. objects.
UMAT is a synonym for UNITMAT.
```

The First Lines Of The SPEAKEASY HELP Documents     Date:  5/25/73

UNITMAT(N) defines an N-by-N unit matrix.
UPPERTRI(X) returns the upper triangular part of X.
USE MEMBER X OF LIBRARY Y causes the member X to be used as input.
USEMEMBER(XX,YY) causes input of the member XX of library YY.
VARIABLE(I,J) defines a 1-dim. array of 101 points from I to J,

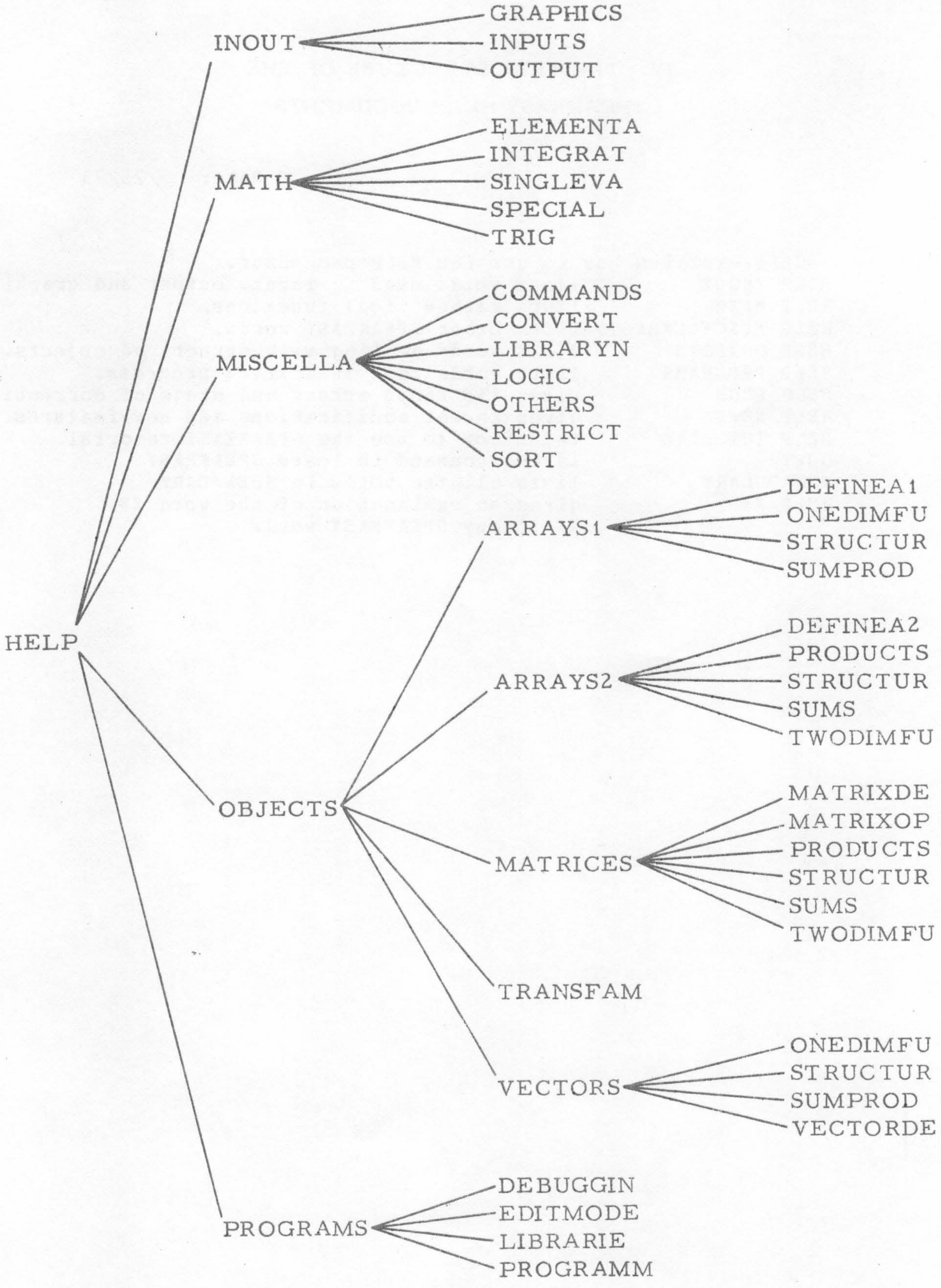
VEC is a synonym for VECTOR.
VECTOR(N:) defines a vector with N components.
VECTORDEF are words used in defining a vector.
VECTORS are classes of words dealing with vectors.
VERSIONS refers to the various versions of the SPEAKEASY processor.

VFAM(X) defines a member of the matrix/vector family.
VLABEL='ANY MESSAGE' labels the vertical scale.
VOCABULARY generates a list of all currently defined words.
VSCALE=(BOTTOM,TOP) specifies the vertical limits of a graph.
VSIZE=Y specifies the vertical size of the graph.

WHERE(X) Y is a conditional statement for array operations.
WHEREVER(X) Y is a conditional statement for array operations.
WHOLE(X) returns the integer part of X.
WRITE(F:X) prints the defined object X in the format F.
ZEROS(F:X) finds the zeros of the function F.

## IV. THE TREE STRUCTURE OF THE SPEAKEASY HELP DOCUMENTS

```
                                                    Date:  5/25/73


HELP        HELP explains how to use the HELP processor.
          HELP INOUT          lists words used in input, output and graphing.
          HELP MATH           lists mathematical functions.
          HELP MISCELLANEOUS  lists other SPEAKEASY words.
          HELP OBJECTS        lists words dealing with structured objects.
          HELP PROGRAMS       lists words used in writing programs.
          HELP BUGS           gives the known errors and stage of correction.
          HELP NEWS           gives recent modifications and new features.
          HELP TUTORIAL       tells how to use the SPEAKEASY tutorial.
          QUIT                is the command to leave SPEAKEASY.
          VOCABULARY          lists all the words in SPEAKEASY.
          HELP XX             gives an explanation of the word XX.
                              XX is any SPEAKEASY word.
```


HELP
INOUT
GRAPHICS
INPUTS
OUTPUT
MATH
ELEMENTA
INTEGRAT
SINGLEVA
SPECIAL
TRIG
MISCELLA
COMMANDS
CONVERT
LIBRARYN
LOGIC
OTHERS
RESTRICT
SORT
OBJECTS
ARRAYS1
DEFINEA1
ONEDIMFU
STRUCTUR
SUMPROD
ARRAYS2
DEFINEA2
PRODUCTS
STRUCTUR
SUMS
TWODIMFU
MATRICES
MATRIXDE
MATRIXOP
PRODUCTS
STRUCTUR
SUMS
TWODIMFU
TRANSFAM
VECTORS
ONEDIMFU
STRUCTUR
SUMPROD
VECTORDE
PROGRAMS
DEBUGGIN
EDITMODE
LIBRARIE
PROGRAMM

```
INOUT        INOUT are classes of words dealing with input and output.
         GRAPHICS (Graphical output)
         INPUTS   (Words used for input)
         OUTPUT   (Words used for output)

           To obtain the words in a given subclass SC, enter
          HELP SC


GRAPHICS     GRAPHICS are words for graphical output that are in version GRAPHEZ.
         ADDGRAPH  adds a graph to a previous graph.
         GRAPH     graphs an array vs another array.
         HLABEL    labels the horizontal scale.
         HSCALE    specifies the limits of the horizontal scale.
         HSIZE     specifies the length of the horizontal scale in inches.
         NEWGRAPH  causes the next graph to be on a new area of paper.
         PLOTSYMB  specifies the symbols to be used in graphing.
         PLOTTITLE titles the graph.
         SETPLOT   specifies box, nobox; scales, noscales; lines, points.
         VLABEL    labels the vertical scale.
         VSCALE    specifies the limits of the vertical scale.
         VSIZE     specifies the length of the vertical scale in inches.

           To obtain a description of a given word XXX, enter
          HELP XXX


INPUTS       INPUTS are words used for input of data or programs.
         CONTINUE causes execution to continue after input in the holding mode.
         DATA     begins and names a data file.
         END      terminates a data file.
         GO       causes execution to continue after input in the holding mode.
         INPUT    puts the system in the holding mode for input.
         KEPT     retrieves information from a library.
         LOADDATA puts data into an array.
         PAUSE    puts the system in the holding mode for input.
         READ     reads cards of specified format.
         RESUME   causes execution to resume after pausing for input.
         STOP     puts the system in a holding mode for input.

           To obtain a description of a given word XXX, enter
          HELP XXX
```

```
OUTPUT      OUTPUT are words related to output.
         ANGLES          specifies if angles are in radians or degrees.
         AUTOPRINT       is used to control automatic printing.
         AUTOTAB         specifies uniform column width for printing.
         COLWIDTH        specifies a minimum column width.
         DONTLIST        indicates that a program is not to be listed.
         DUMP            creates a printout of all defined objects.
         ECHO            prints input along with output.
         ENDAUTOPRINT    turns off automatic printing.
         KEEP            keeps an object in a library.
         LISTPROG        indicates that the program is to be listed.
         MARGINS         specifies the margins.
         NEWPAGE         causes the printer to start a new page.
         NOECHO          suppresses the echoing of input.
         NORATIONALIZE   turns off the rationalization of numbers.
         OMITCLASS       suppresses the printing of the class of objects.
         PRINT           specifies printing.
         PRINTCLASS      prints the class of structured objects.
         PUNCH           punches cards in a specified format.
         RATIONALIZE     causes rationalization of output.
         SETINFINITY     specifies an upper limit to numbers printed.
         SETNULL         specifies a lower limit to numbers printed.
         SIGNIFICANCE    gives the number of significant figures printed.
         SPACE           skips spaces.
         TABULATE        tabulates 1-dimensional objects.
         WRITE           prints output in a specified format.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

```
MATH       MATH are classes of words dealing with mathematical functions.
         ELEMENTAL   (Elemental mathematical functions)
         INTEGRATION (Words used for numerical integrations)
         SINGLEVAR   (Functions of one variable)
         SPECIAL     (Special functions)
         TRIG        (Trigonometric functions)

           To obtain the words in a given subclass SC, enter
          HELP SC


ELEMENTA   ELEMENTAL are elemental mathematical functions.
         ABS        finds the absolute value.
         CONJ       returns the complex conjugate.
         CONJUGATE  returns the complex conjugate.
         EXP        returns the exponential.
         FRAC       returns the fractional part.
         FRACPART   returns the fractional part.
         IMAG       returns the imaginary part.
         IMAGPART   returns the imaginary part.
         INTPART    returns the integer part.
         LOG        returns the natural logarithm.
         NUMBERS    explains how numbers are represented.
         REAL       returns the real part.
         REALPART   returns the real part.
         SIGN       returns the sign.
         SQRT       returns the square root.
         SYMBOLS    explains the mathematical symbols.
         WHOLE      returns the integer part.

           To obtain a description of a given word XXX, enter
          HELP XXX


INTEGRAT   INTEGRATION are words dealing with numerical integration.
         INTEG        defines an integral with a variable upper limit.
         INTEGRAL     defines an integral with a variable upper limit.
         SETGAUSS     gives Gauss-Legendre coordinates and weights.
         SETJACOBI    gives Gauss-Jacobi coordinates and weights.
         SETLAGUERRE  gives Gauss-Laguerre coordinates and weights.
         SETLEGENDRE  gives Gauss-Legendre coordinates and weights.
         TOTALINT     defines a definite integral.
         TOTINT       defines a definite integral.

           To obtain a description of a given word XXX, enter
          HELP XXX
```

```
SINGLEVA   SINGLEVAR are operations on functions of one variable.
         DERIV      finds the derivative of a function.
         DERIVATIVE finds the derivative of a function.
         INTEG      defines an integral with a variable upper limit.
         INTEGRAL   defines an integral with a variable upper limit.
         INTERP     does numerical interpolation.
         INTERPOL   does numerical interpolation.
         NOROOTS    gives the number of zeros or roots of a function.
         NOZEROS    gives the number of zeros or roots of a function.
         ROOTS      gives the zeros or roots of a function.
         TOTALINT   defines a definite integral.
         TOTINT     defines a definite integral.
         ZEROS      gives the zeros or roots of a function.

           To obtain a description of a given word XXX, enter
          HELP XXX


SPECIAL    SPECIAL are special mathematical functions.
         BESSEL     returns cylindrical Bessel fn. of the first kind.
         BESSELK    returns the modified Bessel function K.
         CGAMMA     returns the gamma function of a complex argument.
         DOUBLEFAC returns the doublefactorial.
         ELLIPE     returns the complete elliptical integral E.
         ELLIPK     returns the complete elliptical integral K.
         EONE       returns the exponential integral from x to infinity.
         ERF        returns the error function.
         ERFC       returns the complementary error function.
         GAMMA      returns the gamma function of a real argument.
         LOGGAMMA   returns the logarithm of the gamma function.
         NEUMANN    returns cylindrical Bessel fn. of the second kind.
         SPHBES     returns sperical Bessel fn. of the first kind.
         SPHBESN    returns spherical Bessel fn. of the second kind.

           To obtain a description of a given word XXX, enter
          HELP XXX


TRIG       TRIG are the trigonometric functions.
         ACOS
         ACOT
         ASIN
         ATAN
         COS
         COSH
         COT
         SIN
         SINH
         TAN

           To obtain a description of a given word XXX, enter
          HELP XXX
```

The Tree Structure Of The SPEAKEASY HELP Documents Date:  5/25/73

```
MISCELLA   MISCELLANEOUS are classes of words not in any other classification.
        COMMANDS       (SPEAKEASY commands)
        CONVERT        (Integer-real-hex conversions)
        LIBRARYNAMES   (SPEAKEASY libraries)
        LOGIC          (Logical, conditional, and relational operators)
        OTHERS         (Miscellaneous words)
        RESTRICTED     (Restricted words-words which may not be used as names)
        SORT           (Sorting or ranking functions)

          To obtain the words in a given subclass SC, enter
         HELP SC


COMMANDS   COMMANDS are words that are commands in SPEAKEASY.
        ACCURACY       controls the accuracy of tests.
        ANGLES         specifies if angles are in radians or degrees.
        AUTOCORE       is used to set the users space allocator.
        AUTOPRINT      is used to control automatic printing.
        CLEAR          removes all definitions.
        CLEARDATA      removes all defined numerical objects.
        DOMAIN         sets the REAL/COMPLEX domain.
        DONTLIST       indicates that the program is not to be listed.
        ECHO           prints input along with output.
        ENDAUTOPRINT   terminates automatic printing.
        EXECUTE        executes a program.
        FREE           eliminates selected defined objects.
        HIGHWIDE       activates HIGH-WIDE arithmetic for arrays.
        HIWIDE         activates HIGH-WIDE arithmetic for arrays.
        LISTPROG       indicates that the program is to be listed.
        MARGINS        controls the margins.
        NAMES          tells the currently defined names.
        NOECHO         suppresses the echoing of input.
        NORATIONALIZE  turns off RATIONALIZE.
        OMITCLASS      suppresses the printing of the class of objects.
        PRINTCLASS     prints the class of structured objects.
        QUIT           terminates a SPEAKEASY session.
        RATIONALIZE    causes rationalization of output.
        SIGNIFICANCE   controls printed output.
        SIZE           sets the user's space allocator.
        TIME           tells the time since the start of the run.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

```
CONVERT     CONVERT are words used for numerical conversions.
          DEC   converts hexidecimal to decimal form.
          INT2  converts to integer 2.
          INT4  converts to integer 4.
          REAL4 converts to real 4.
          REAL8 converts to real 8.

            To obtain a description of a given word XXX, enter
           HELP XXX

LIBRARYN    LIBRARYNAMES are the names of the libraries attached to the processor.
          DOCUMENTS
          HELP
          LINKULES
          MYDOCS
          MYHELP
          MYKEEP
          MYKEPT
          MYLINKS
          MYPROCS
          PROCLIB

            To obtain a description of a given word XXX, enter
           HELP XXX

LOGIC       LOGIC are the logical, relational, and conditional operators.
          EQ
          GE
          GT
          LE
          LT
          NE
          AND
          NOT
          OR
          IF is a conditional operator for scalars.
          WHERE is a conditional operator for arrays.
          WHEREVER is a conditional operator for arrays.

            To obtain a description of a given operator XXX, enter
           HELP XXX
```

The Tree Structure Of The SPEAKEASY HELP Documents Date: 5/25/73

```
OTHERS      OTHERS are miscellaneous words not under a specific classification.
         ERRORS     explains errors in SPEAKEASY.
         HENCEFORTH redefines words.
         HIERARCHY  gives the hierarchy of arithmetic operations.
         LOC        is a logical operator that gives indices of true elements.
         LOCS       is a logical operator that gives indices of true elements.
         NUMBERS    explains how numbers are represented.
         OBJECT     generates an inplace name.
         RANDOM     generates random numbers.
         SIMEQ      solves simultaneous linear equations.
         SYMBOLS    explains the symbols used in SPEAKEASY.
         VERSIONS   gives the versions of SPEAKEASY.
         VOCABULARY lists the words found in the HELP documents.

           To obtain a description of a given word XXX, enter
          HELP XXX


RESTRICT    RESTRICTED are the restricted words that cannot be names of variables.
         CONTINUE
         DATA
         END
         ENDLOOP
         EDIT
         EXECUTE
         FIN
         FOR
         FREE
         GOTO
         IF
         LOADDATA
         PRINT
         PROGRAM
         NEWPAGE
         RETURN
         SPACE
         USE
         WHERE
         WHEREVER

           To obtain a description of a given word XXX, enter
          HELP XXX


SORT        SORT are sorting or ranking functions.
         ORDERED gives elements of a structured object in increasing order.
         ORDERER gives the indices of ordered elements.
         RANKED  gives elements of a structured object in increasing order.
         RANKER  gives the indices of ordered elements.

           To obtain a description of a given word XXX, enter
          HELP XXX
```

```
OBJECTS     OBJECTS are classes of words dealing with structured objects.
        ARRAYS1       (1-dimensional arrays)
        ARRAYS2       (2-dimensional arrays)
        MATRICES
        TRANSFAMILY (Functions to go from one family of structured objects
                      to another)
        VECTORS

          To obtain the words in a given subclass SC, enter
         HELP SC


ARRAYS1     ARRAYS1 are words dealing with 1-dimensional arrays.
        DEFINEA1           (Functions defining 1-dim. arrays)
        ONEDIMFUNCTIONS (Functions of 1-dim. objects)
        STRUCTURE          (Fuctions which give information about the
                            structure of structured objects)
        SUMPROD            (Sums and products of elements of
                            structured objects)

          To obtain the words in a given subclass SC, enter
         HELP SC


DEFINEA1    DEFINEA1 are words used in defining 1-dimensional arrays.
        AFAM      defines a member of the array family.
        ARRAY     defines an array.
        A1D       defines a 1-dim. array.
        DIAGELS   returns the diagonal elements of a matrix.
        GRID      defines a 1-dim. array of equally spaced points.
        INTEGERS defines a 1-dim. array of integers.
        INTS      defines a 1-dim. array of integers.
        LOWERTRI gives the lower triangular part of a 2-dim. array.
        UPPERTRI gives the upper triangular part of a 2-dim. array.
        VARIABLE defines a 1-dim. array of equally spaced points.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

```
ONEDIMFU   ONEDIMFUNCTIONS are words that deal with 1-dim. objects.
        AVERAGE   returns the average value of the elements of the object.
        CONSTRAIN constrains objects according to a specified condition.
        LENGTH    gives the number of elements in the object.
        LOCMAX    gives the location of the maximum element.
        LOCMIN    gives the location of the minimum element.
        MAX       gives the value of the maximum element.
        MELD      gives an odometer ordering of elements.
        MIN       gives the value of the minimum element.
        NOELS     gives the number of elements of the object.
        SELECT    truncates or expands objects according to an index.

          To obtain a description of a given word XXX, enter
         HELP XXX


STRUCTUR   STRUCTURE are functions giving information on the structure of objects.
        LENGTH returns the number of elements.
        NOCOLS returns the number of columns.
        NOELS  returns the number of elements.
        NOROWS returns the number of rows.

          To obtain a description of a given word XXX, enter
         HELP XXX


SUMPROD    SUMPROD are words for sums or products of elements of objects.
        CUMPROD returns the cumulative product of elements.
        CUMSUM  returns the cumulative sum of elements.
        PROD    returns the product of elements.
        SUM     returns the sum of elements.
        SUMSQ   returns the sum of the squares of elements.

          To obtain a description of a given word XXX, enter
         HELP XXX


ARRAYS2    ARRAYS2 are words dealing with 2-dimensional arrays.
        DEFINEA2        (Functions defining 2-dim. arrays)
        PRODUCTS        (Products of elements of 2-dim. objects)
        STRUCTURE       (Functions which give information about the
                         structure of structured objects)
        SUMS            (Sums of elements of 2-dim. objects)
        TWODIMFUNCTIONS (Functions of 2-dim. objects)

          To obtain the words in a given subclass SC, enter
         HELP  SC
```

The Tree Structure Of The SPEAKEASY HELP Documents Date: 5/25/73

```
DEFINEA2    DEFINEA2 are words used in defining 2-dimensional arrays.
        AFAM      defines a member of the array family.
        ARRAY     defines an array.
        ARRAY2D   defines a 2-dim. array.
        A2D       defines a 2-dim. array.
        COLARRAY  defines a 2-dim. column array.
        ROWARRAY  defines a 2-dim. row array.

          To obtain a description of a given word XXX, enter
         HELP XXX


PRODUCTS    PRODUCTS are words for products of elements of objects.
        CUMPROD   defines the cumulative product of elements.
        PROD      defines the product of elements of a structured object.
        PRODCOLS  multiplies the elements in each column.
        PRODROWS  multiples the elements in each row.

          To obtain a description of a given word XXX, enter
         HELP XXX


STRUCTUR    STRUCTURE are functions giving information on the structure of objects.
        LENGTH returns the number of elements.
        NOCOLS returns the number of columns.
        NOELS  returns the number of elements.
        NOROWS returns the number of rows.

          To obtain a description of a given word XXX, enter
         HELP XXX


SUMS        SUMS are words for sums of elements of 2-dim. structured objects.
        CUMSUM     returns the cumulative sum of elements.
        SUM        returns the sum of elements.
        SUMCOLS    sums the elements in each column.
        SUMROWS    sums the elements in each row.
        SUMSQ      sums the squares of the elements.
        SUMSQCOLS  sums the squares of the elements in each column.
        SUMSQROWS  sums the squares of the elements in each row.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

```
TWODIMFU    TWODIMFUNCTIONS are words that deal with 2-dim. objects.
        COLMAX    specifies the column with the largest element.
        COLMIN    specifies the column with the smallest element.
        DIAGELS   returns the diagonal elements of a matrix.
        LENGTH    returns the number of elements.
        LOWERTRI  returns the lower triangular part of a 2-dim. object.
        HIGHWIDE  activates HIGH-WIDE arithmetic for arrays.
        HIWIDE    activates HIGH-WIDE arithmetic for arrays.
        MAX       returns the maximum element.
        MAXOFCOL  returns the maximum element in each column.
        MAXOFROW  returns the maximum element in each row.
        MIN       returns the smallest element.
        MINOFCOL  returns the smallest element in each column.
        MINOFROW  returns the smallest element in each row.
        NOELS     returns the number of elements.
        ROWMAX    specifies the row with the largest element.
        ROWMIN    specifies the row with the smallest element.
        UPPERTRI  returns the upper triangular part of a 2-dim. object.

          To obtain a description of a given word XXX, enter
         HELP XXX


MATRICES    MATRICES are classes of words dealing with matrices.
        MATRIXDEF        (Functions defining matrices)
        MATRIXOPS        (Matrix operations)
        PRODUCTS         (Products of elements of structured objects)
        STRUCTURE        (Functions which give information about
                          the structure of structured objects)
        SUMS             (Sums of elements of structured objects)
        TWODIMFUNCTIONS  (Functions of 2-dim. objects)

          To obtain the words in a given subclass SC, enter
         HELP SC
```

```
MATRIXDE   MATRIXDEF are words used in defining a matrix.
        AMAT     defines an antisymmetric matrix.
        ASYMMAT defines an antisymmetric matrix.
        COLMAT  defines a column matrix.
        DIAGMAT defines a diagonal matrix.
        DMAT    defines a diagonal matrix.
        MAT     defines a matrix.
        MATRIX  defines a matrix.
        MFAM    defines a member of the matrix/vector family.
        ROWMAT  defines a row matrix.
        SMAT    defines a symmetric matrix.
        SYMMAT  defines a symmetric matrix.
        UMAT    defines a unit matrix.
        UNITMAT defines a unit matrix.
        VFAM    defines a member of the matrix/vector family.

          To obtain a description of a given word XXX, enter
         HELP XXX


MATRIXOP   MATRIXOPS lists words that are matrix operators.
        ADJOINT     returns the transpose complex conjugate of a matrix.
        DET         returns the determinant of a matrix.
        DIAGELS     returns the diagonal elements of a matrix.
        EIGENSYSTEM returns the eigenvalues of a symmetric matrix.
        EIGENVALS   returns the eigenvalues of a matrix.
        EIGENVECS   returns the eigenvectors of a matrix.
        GEIGEN      returns the eigenvalues of a symmetric matrix.
        INVERSE     finds the inverse of a matrix.
        LOWERTRI    returns the lower triangular part of a matrix.
        TRACE       finds the trace of a matrix.
        TRANSP      returns the transpose of a matrix.
        TRANSPOSE   returns the transpose of a matrix.
        UPPERTRI    returns the uppper triangular part of a matrix.

          To obtain a description of a given word XXX, enter
         HELP XXX


PRODUCTS   PRODUCTS are words for products of elements of objects.
        CUMPROD  defines the cumulative product of elements.
        PROD     defines the product of elements of a structured object.
        PRODCOLS multiplies the elements in each column.
        PRODROWS multiples the elements in each row.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

The Tree Structure Of The SPEAKEASY HELP Documents Date: 5/25/73

```
STRUCTUR    STRUCTURE are functions giving information on the structure of objects.
        LENGTH returns the number of elements.
        NOCOLS returns the number of columns.
        NOELS  returns the number of elements.
        NOROWS returns the number of rows.

          To obtain a description of a given word XXX, enter
           HELP XXX


SUMS        SUMS are words for sums of elements of 2-dim. structured objects.
        CUMSUM      returns the cumulative sum of elements.
        SUM         returns the sum of elements.
        SUMCOLS     sums the elements in each column.
        SUMROWS     sums the elements in each row.
        SUMSQ       sums the squares of the elements.
        SUMSQCOLS   sums the squares of the elements in each column.
        SUMSQROWS   sums the squares of the elements in each row.

          To obtain a description of a given word XXX, enter
          HELP XXX


TWODIMFU    TWODIMFUNCTIONS are words that deal with 2-dim. objects.
        COLMAX    specifies the column with the largest element.
        COLMIN    specifies the column with the smallest element.
        DIAGELS   returns the diagonal elements of a matrix.
        LENGTH    returns the number of elements.
        LOWERTRI  returns the lower triangular part of a 2-dim. object.
        HIGHWIDE  activates HIGH-WIDE arithmetic for arrays.
        HIWIDE    activates HIGH-WIDE arithmetic for arrays.
        MAX       returns the maximum element.
        MAXOFCOL  returns the maximum element in each column.
        MAXOFROW  returns the maximum element in each row.
        MIN       returns the smallest element.
        MINOFCOL  returns the smallest element in each column.
        MINOFROW  returns the smallest element in each row.
        NOELS     returns the number of elements.
        ROWMAX    specifies the row with the largest element.
        ROWMIN    specifies the row with the smallest element.
        UPPERTRI  returns the upper triangular part of a 2-dim. object.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

The Tree Structure Of The SPEAKEASY HELP Documents Date:  5/25/73

```
TRANSFAM   TRANSFAMILY are functions to go from one family to another.
        AFAM    defines a member of the array family.
        MFAM    defines a member of the matrix/vector family.
        RECLASS alters the structure of objects.
        VFAM    defines a member of the matrix/vector family.

          To obtain a description of a given word XXX, enter
         HELP XXX


VECTORS    VECTORS are classes of words dealing with vectors.
        ONEDIMFUNCTIONS  (Functions of 1-dim objects)
        STRUCTURE        (Functions which give information about the
                          structure of structured objects)
        SUMPROD          (Sums and products of elements)
        VECTORDEF        (Functions defining vectors)

          To obtain the words in a given subclass SC, enter
         HELP SC


ONEDIMFU   ONEDIMFUNCTIONS are words that deal with 1-dim. objects.
        AVERAGE    returns the average value of the elements of the object.
        CONSTRAIN  constrains objects according to a specified condition.
        LENGTH     gives the number of elements in the object.
        LOCMAX     gives the location of the maximum element.
        LOCMIN     gives the location of the minimum element.
        MAX        gives the value of the maximum element.
        MELD       gives an odometer ordering of elements.
        MIN        gives the value of the minimum element.
        NOELS      gives the number of elements of the object.
        SELECT     truncates or expands objects according to an index.

          To obtain a description of a given word XXX, enter
         HELP XXX


STRUCTUR   STRUCTURE are functions giving information on the structure of objects.
        LENGTH returns the number of elements.
        NOCOLS returns the number of columns.
        NOELS  returns the number of elements.
        NOROWS returns the number of rows.

          To obtain a description of a given word XXX, enter
         HELP XXX
```

```
SUMPROD     SUMPROD are words for sums or products of elements of objects.
         CUMPROD returns the cumulative product of elements.
         CUMSUM  returns the cumulative sum of elements.
         PROD    returns the product of elements.
         SUM     returns the sum of elements.
         SUMSQ   returns the sum of the squares of elements.

            To obtain a description of a given word XXX, enter
            HELP XXX


VECTORDE    VECTORDEF are words used in defining a vector.
         MFAM   defines a member of the matrix/vector family.
         VEC    defines a vector.
         VECTOR defines a vector.
         VFAM   defines a member of the matrix/vector family.

            To obtain a description of a given word XXX, enter
           HELP XXX
```

The Tree Structure Of The SPEAKEASY HELP Documents Date: 5/25/73

```
PROGRAMS   PROGRAMS are classes of words that are useful in writing programs.
        DEBUGGING    (Words useful in debugging a program)
        EDITMODE     (Words restricted to the edit mode)
        LIBRARIES    (Library creation and use)
        PROGRAMMODE (Words restricted to the program mode)

          To obtain the words in a given subclass SC, enter
         HELP SC


DEBUGGIN   DEBUGGING are words useful in debugging a program.
        AUTOPRINT     prints words each time they are used.
        DUMP          creates a printout of all defined objects.
        ENDAUTOPRINT turns off AUTOPRINT.
        ONERROR       specifies action to be taken after an error.

          To obtain a description of a given word XXX, enter
         HELP XXX


EDITMODE   EDITMODE are words used in the EDIT mode of TSO SPEAKEASY.
        COMPILE  compiles the program.
        COPY     copies a statement or statements.
        DATA     gets one into the EDIT mode to create a datafile.
        DELETE   deletes a statement or statements.
        EDIT     gets one into the EDIT mode.
        INSERT   inserts statements at a specific location.
        LIST     lists the program or a series of statements.
        MOVE     moves a statement or group of statements.
        PROGRAM  gets one into the EDIT mode to create a program.
        RUN      compiles and executes the program.

          EDIT, DATA, and PROGRAM are words to get one into the
        EDIT mode.
          To obtain a description of a given word XXX, enter
         HELP XXX
```

```
The Tree Structure Of The SPEAKEASY HELP Documents Date:  5/25/73


LIBRARIE   LIBRARIES are words for creating and using members of libraries.
          CREATE         creates a member of a private library.
          CREATEMEMBER creates a member of a private library.
          LIBINDEX       defines the names of members of a library.
          LIBNAMES       returns the names of the SPEAKEASY libraries.
          LINKLIB        adds a linkule library to those available.
          LIST           lists a member of a library.
          LISTHEAD       lists a specified no. of lines of a member of a library.
          LISTMEMBER     lists a member of a library.
          SETLIB         changes the name of a library.
          USE            uses a member of a library as input.
          USEMEMBER      uses a member of a library as input.

            To obtain a description of a given word XXX, enter
           HELP XXX


PROGRAMM   PROGRAMMODE are words restricted to the PROGRAM mode.
          CONTINUE causes no operation in the PROGRAM mode.
          END        denotes the end of a program.
          ENDLOOP    denotes the end of a FOR loop.
          EXECUTE    executes a program.
          FIN        marks the end of a FOR loop.
          FOR        designates a loop.
          GOTO       branches to a specified statement.
          INPUT      puts the system in a holding mode for input.
          LABEL      labels a statement.
          PROGRAM    names a program and puts the system in the EDIT mode.
          PAUSE      puts the system in the holding mode for input.
          RETURN     returns execution to the program calling the subprogram.
          STOP       puts the system in a holding mode for input.

            To obtain a description of a given word XXX, enter
           HELP XXX
```

## V. THE SPEAKEASY HELP DOCUMENTS

Date: 5/25/73

ABS ABS(X) returns the absolute value of X.
ABS(X) defines an object with the same structure as X but with elements of the result equal to the absolute values of the corresponding elements of X.

ACCURACY ACCURACY(VAL) specifies the smallest number not to be taken as zero.
Whenever the equality of two numbers is checked during the execution of a program, any number less than a certain number is regarded as zero. The value of this number may be set to be any number VAL by the statement ACCURACY(VAL).
If not specified, the value of VAL is 1.E-8.

ACOS ACOS(X) defines the arccosine of X.
ACOS(X) defines an object with the same structure as X with elements that are the arccosine of the corresponding elements of X.
All the elements Xi of X must be real and satisfy the condition |Xi|<1 or the condition |Xi|=1.

ACOT ACOT(X) defines the arccotangent of X.
ACOT(X) defines an object with the same structure as X with elements that are the arccotangent of the corresponding elements of X.
All elements Xi of X must be real.

ADDGRAPH ADDGRAPH(I:J) plots I versus J on the previous graph.
ADDGRAPH has the same meaning as graph except a new area of paper is not used. Design statements except the scale statements and the label statements are reexamined prior to adding the graph. Thus the plotting format may be changed for each additional graph.
I is plotted along the vertical axis and J is along the horizontal. All objects should be 1-dimensional and real and have the same number of elements. A 2-dimensional object in I is treated as several 1-dimensional objects each consisting of a row of the original 2-dimensional object. Thus a 2-dimensional object in I must have as many columns as J has elements.
If no previous graph has been drawn, ADDGRAPH acts as GRAPH.
ADDGRAPH is available only in version GRAPHEZ.

ADJOINT ADJOINT(X) gives the adjoint of the matrix X.
The adjoint of X is the transpose complex conjugate of X.

AFAM AFAM(X) defines a member of the array family. The result has the same structure as X but is an array. If X is a vector or a 1-dimensional array, the result is a 1-dimensional array. If X is a matrix or a 2-dimensional array, the result is a 2-dimensional array.

AMAT AMAT is a synonym for ASYMMAT.

AMAT(N:I,J,...,K) defines an N-by-N antisymmetric matrix. The element list is used to fill the lower triangular part (excluding the diagonal elements which will be set to zero) by rows. The portion above the diagonal is then filled by making the matrix antisymmetric. If any object in the list defining the elements is structured, the elements of that structured object are used.

AND .AND. is the logical operator "and". The periods must appear on both sides of the operator. A .AND. B = 1 if both A and B are not equal to zero; otherwise A .AND. B = 0. A and B may be any real numbers.

ANGLES ANGLES specifies whether the user is using radians or degrees. RADIANS is default.

ANGLES IN DEGREES specifies that angles are in degrees. The word IN may be omitted in the SPEAKEASY calling sequence. DEGREES may be abbreviated by DEG or DEGS.

ANGLES IN RADIANS specifies that angles are in radians. The word IN may be omitted in the SPEAKEASY calling sequence. RADIANS may be abbreviated by RAD or RADS.

ARRAY

ARRAY(N:) defines a 1-dimensional N-component array. If no further arguments are given, all the components are set equal to zero.

An alternative form is A1D.

ARRAY(:I,J,...,K) defines a 1-dim. array and specifies the elements. The elements are preset by the argument list I,J,...,K. If any argument of the defining argument list is structured, the elements of the structured object are used. The size of the array is equal to the total number of elements specified. If a complex element is encountered, then a complex 1-dimensional array is defined.

ARRAY(N:I,J,...,K) defines a 1-dim. N-component array. The elements are preset by the element list I,J,...,K. If any argument in the element list is structured, the elements of the structured object are used. If a complex element is encountered, then a complex 1-dimensional array is defined. If all N-components are not specified, the unspecified components are set equal to zero.

ARRAY(N,M:) defines a 2-dimensional N-by-M array. If no further arguments are given, all elements are set equal to zero.

An alternative form is ARRAY2D.

ARRAY(N,M:I,J,...,K) defines a 2-dim. N-by-M array with preset els. The elements are preset by the element list I,J,...,K. If any argument in the element list is structured, then the elements of the structured object are used. If a complex element is encountered, then a two-dimensional N-by-M complex array is defined. All the elements not preset by the element list are set equal to zero.

ARRAY2D

ARRAY2D(N,M:) defines a 2-dimensional N-by-M array. If no further arguments are given, all the elements are set equal to zero.

An alternative form is ARRAY(N,M:).

A shortened form is A2D.

ARRAY2D(N,M:I,J,...,K) defines a 2-dim. N-by-M array. The elements are preset by the element list I,J,...,K. If any object of the defining element list is structured, the elements of the structured object are used. If a complex element is encountered, then a 2-dimensional N-by-M complex array is defined. All elements not preset by the element list are set equal to zero.

ASIN

ASIN(X) defines the arcsine of X. ASIN(X) defines an object with the same structure as X with elements that are the arcsine of the corresponding elements of X.

All the elements Xi of X must be real and satisfy the condition |Xi|<1 or the condition |Xi|=1.

ASYMMAT

ASYMMAT(N:I,J,...,K) defines an N-by-N antisymmetric matrix. The element list is used to fill the lower triangular part (excluding the diagonal elements which are set to zero) by rows. The portion above the diagonal is then filled by making the matrix antisymmetric. If any object in the list defining the elements is structured, the elements of that structured object are used.

A shortened form is AMAT.

ATAN

ATAN(X) defines the arctangent of X. ATAN(X) defines an object with the same structure as X with elements that are the arctangent of the corresponding elements of X.

All elements Xi of X must be real.

AUTOCORE

AUTOCORE specifies that the data storage space is in LCS. AUTOCORE means that the largest amount of LCS available is to be used for the job. If LCS does not exist, main core is used. CAUTION: If LCS does not exist, using AUTOCORE may result in no core being left for subsequent use by LINKULES.

If AUTOCORE is to be used, it should be the first card in the SPEAKEASY job.

AUTOPRIN

AUTOPRINT(X,Y,...,Z) prints X,Y,...,Z each time they are redefined. Each object of the name list X,Y,...,Z is printed whenever it appears on the left of an equation.

AUTOPRINT prints all objects as they are defined or altered. AUTOPRINT with no arguments prints all objects as they are defined or altered. The statement ENDAUTOPRINT turns off autoprint.

AUTOTAB

AUTOTAB specifies uniform column width for printout. The value of N for the column width is automatically set at 10 greater than the number of significant figures so that the print is uniform for all objects composed of real numbers.

AVERAGE

AVERAGE(X) returns the average value of the elements of X. X is a structured object.

A1D A1D(N:) defines a 1-dimensional array. If no further arguments are given, all the components are set equal to zero.

A1D is a synonym for ARRAY(N:).

A1D(:I,J,...,K) defines a 1-dim. array and specifies the elements. The components are preset by the argument list I,J,...,K. If any argument in the argument list is structured, then the elements of the structured object are used. The size of the array is equal to the total number of elements specified. If a complex element is encountered, then a complex 1-dimensional array is defined.

A1D(N:I,J,...,K) defines a 1-dim. N-component array. The components are preset by the element list I,J,...,K. If any argument of the element list is structured, then the elements of the structured object are used. If a complex element is encountered, then a complex 1-diminsional array is defined. If all N-components are not specified, the unspecified components are set equal to zero.

A2D A2D is a synonym for ARRAY2D.

A2D(N,M:) defines a 2-dimensional N-by-M array. If no further arguments are given, the elements are set equal to zero.

A2D(N,M:I,J,...,K) defines a 2-dim. N-by-M array with preset els. The elements are preset by the element list I,J,...,K. If any object of the defining element list is structured, the elements of the structured object are used. If a complex element is encountered, then a 2-dimensional N-by-M complex array is defined. All elements not preset by the element list are set equal to zero.

BESSEL BESSEL(NU,X) calculates cylindrical Bessel fn. of the first kind. The result is the Bessel function J sub NU of X. NU and X must be real and nonnegative. There exists a restriction in the SPEAKEASY linkule that (X+NU)<400.

BESSELK BESSELK(NU,X) returns the modified Bessel function K of X. The result is the modified Bessel function K sub NU of X. NU and X must be real and nonnegative. There exists a restriction in the SPEAKEASY linkule that (X+NU)<400.

BUGS BUGS are the current bugs in SPEAKEASY. This document will list the current errors in SPEAKEASY and their status of correction.

CGAMMA CGAMMA(X) returns the gamma function of X (X is complex). The result has the same structure as X with elements equal to the gamma function of the corresponding elements of X.

If X is real, the gamma function of X may be obtained from GAMMA(X).

CLEAR CLEAR erases the entire active data storage. After the execution of the statement CLEAR there are no user-defined objects, programs, or henceforth definitions.

CLEARDAT CLEARDATA frees all previously defined numerical data. CLEARDATA does not free programs or hencforth definitions.

COLARRAY COLARRAY(N:) defines a 2-dim. N-component column array. If no further arguments are given, all N-components are set equal to zero.

COLARRAY(:I,J,...,K) defines a 2-dim. column array with preset els. The components are preset by the element list I,J,...,K . If any argument of the element list is structured, then the elements of that structured object are used. If a complex element is encountered, then a comlex 2-dimensional column array is defined.

COLARRAY(N:I,J,...,K) defines a 2-dim. N-component column array. The components are preset by the element list I,J,...,K . If any argument of the element list is structured, then the elements of that structured object are used. If a complex element is encountered, then a complex 2-dimensional column array is defined. If all N-components are not preset by the element list the unspecified components are set equal to zero.

COLMAT COLMAT(N:) defines an N-by-1 matrix which is a column. If no further arguments are given, all N elements are set equal to zero.

COLMAT(:I,J,...,K) defines a column matrix by specifying the els. The element list I,J,...,K is used to preset the elements of the column matrix. If one of the arguments of the element list is structured, the elements of that structured object are used. If one of the elements is complex, the column matrix is complex.

COLMAT(N:I,J,...,K) defines an N-by-1 column matrix with preset els. The elements are specified by the element list I,J,...,K. If one of the arguments of the list is structured, the elements of the structured object are used. The column matrix is complex if one of the preset elements is complex. If all N elements are not preset by the element list, the unspecified elements are set equal to zero.

COLMAX COLMAX(X) specifies the column with the maximum element of X.

COLMIN COLMIN(X) specifies the column with the minimum element of X.

COLWIDTH COLWIDTH(N) specifies the minimum column width to be used in printing. This statement prevents the column width of the print routine from being less than N characters. If N is 10 larger than the number of significant figures desired, the print will be uniform for all objects composed of real numbers.

COMPILE COMPILE returns the processor to the MANUAL mode from EDIT. COMPILE is used after a program has been edited to return from the EDIT mode to the MANUAL mode.

CONJ CONJ is a synonym for CONJUGATE.
CONJ(X) returns the complex conjugate of X. CONJ(X) defines an object with the same structure as X but with elements equal to the complex conjugate of the corresponding elements of X.

CONJUGAT CONJUGATE(X) returns the complex conjugate of X. CONJUGATE(X) defines an object with the same structure as X but with elements equal to the complex conjugate of the corresponding elements of X.
A shortened form is CONJ.

CONSTRAI CONSTRAIN(A,B,...,C:X) constrains A,B,...,C according to condition X. A,B,...,C are 1-dimensional objects of the same length. X is a logical expression of the same length as the objects to the left of the colon. The objects to the left of the colon are truncated so that they contain only those elements satisfying the logical expression X.
All 1-dimensional objects must have the same length.
For an example refer to the SPEAKEASY-3 manual.

CONTINUE CONTINUE used in the program mode causes no operation to be performed. The CONTINUE statement is a nonoperational statement to which a label may be attached. CONTINUE is ignored in the manual mode.
CONTINUE causes execution of a program to resume from the holding mode. CONTINUE is used in the holding mode after statements are entered to cause execution of the program to continue from the point where it was interrupted by PAUSE, INPUT, or STOP.
Synonyms for CONTINUE are GO, RESUME, and a null line.
CONTINUE is a restricted word.

COPY

COPY N,M,K(I) copies statements N through M to K,K+I,K+2I,...
COPY N,M,K copies statements N through M to K,K+1,K+2,...
COPY N copies statement N to the last position.
COPY N,M copies statements N through M to the last positions.
COPY is a command available in the EDIT mode.

COS

COS(X) returns the cosine of X.

COS (X) defines an object with the same structure as X but the elements of the result are the cosine of the corresponding elements of X.

Each element Xi of X must satisfy the condition |Xi|<1.E15.

COSH

COSH(X) defines the hyperbolic cosine of X.

COSH(X) defines an object with the same structure as X but with elements equal to the hyperbolic cosine of the corresponding elements of X.

The elements Xi of X must be real and must satisfy the condition |Xi|<170.

COT

COT(X) returns the cotangent of X.

COT(X) defines an object with the same structure as X but the elements of the result are the cotangent of the corresponding elements of X.

Each element Xi of X must satisfy the condition |Xi|<1.E15.

CREATE

CREATE MEMBER X ON LIBRARY Y creates a new member in a library.

All input following the input CREATE MEMBER until the input ENDCREATE is encountered is put into the member with the name X in library Y.

CREATE MEMBER X creates a member X in the library MYPROCS.

CREATEME

CREATEMEMBER(XX,YY) creates member XX in library YY.

XX is the name of the member that is created.
YY is the name of the library into which XX is put.
All the input following the CREATEMEMBER input is put into the member named XX in library YY until the input ENDCREATE is encountered.

CREATEMEMBER(X) creates member X in the library MYPROCS.
An alternative form is CREATE MEMBER XX ON LIBRARY YY.

CUMPROD

CUMPROD(X) defines the cumulative product of the elements of X.

CUMPROD(X) defines an object with the same structure as X with elements that are the cumulative product of all previous elements of X. Two-dimensional objects are treated row by row.

CUMSUM

CUMSUM(X) defines the cumulative sum of the elements of X. CUMSUM(X) defines an object with the same structure as X with elements that are the cumulative sum of all previous elements of X. Two-dimensional objects are treated row by row.

DATA

DATA NAME marks the begining of a data file. NAME is the name given to the data file. The data file is terminated by the single word END. All cards between the card DATA NAME and the card END contain data that may be punched in any format. It is read into a specified array by a LOADDATA statement.

DATA is a restricted word.

DEC

DEC(ZA1,ZA2,...,ZAN) converts A1,A2,...,AN from hex to decimal. The hexidecimal numbers A1,A2,...,AN must be prefixed by a letter to prevent their conversion to numbers by SPEAKEASY. They may be 7 hexidecimal figures in length. The first character must be a letter so they are 8-character words.

The result is an array with elements that are the decimal equivalents of the hex arguments.

DELETE

DELETE N,M deletes the statements N through M.

DELETE N deletes the statement numbered N.

DELETE is one of the commands available in the EDIT mode. DELETE may be abbreviated by D.

DERIV

DERIV is a synonym for DERIVATIVE.

DERIV(F:X) finds the derivative of the function F with respect to X. X and F are a 1-dimensional arrays and so is the result. Each element of the result is dF/dXi, where Xi is the corresponding element of X.

DERIVATI

DERIVATIVE(F:X) finds the derivative of F with respect to X. F and X are a 1-dimensional arrays and so is the result. Each element of the result is dF/dXi, where Xi is the corresponding element of X.

A shortened form is DERIV.

DET

DET(X) defines the determinant of the matrix X.

DIAGELS

DIAGELS(X) selects the diagonal elements of X. The diagonal elements are given in the order in which they occur in X. X must be a square matrix or array. The result is a 1-dimensional array.

DIAGMAT DIAGMAT(N:I,J,...,K) defines an N-by-N diagonal matrix. All elements are zero except for those along the diagonal, which are defined by the element list I,J,...,K. If any object in the element list is structured, the elements of that structured object are used.

A shortened form is DMAT.

DMAT DMAT is a synonym for DIAGMAT.

DMAT(N:I,J,...,K) defines an N-by-N diagonal matrix. All elements are zero except for those along the diagonal, that are defined by the element list I,J,...,K. If any object in the element list is structured, the elements of that structured object are used.

DOCUMENT DOCUMENT is a library of larger documents. The document library contains the documents of the system other than the HELP documents.

DOMAIN DOMAIN specifies whether real or complex numbers are being used. If no domain is specified by the user, it is assumed to be the real domain. The domain may be altered at will by the user by the statements DOMAIN(REAL) or DOMAIN(COMPLEX).

DOMAIN(REAL) specifies that real numbers are to be used. Any calculation leading to an imaginary or complex result is treated as an error.

DOMAIN(COMPLEX) specifies that complex numbers are allowed.

DONTLIST DONTLIST indicates that the program is not to be printed. If not specified, DONTLIST is default for the interactive operation; LISTPROG is usually default for batch jobs.

DOUBLEFA DOUBLEFACTORIAL(X) returns the doublefactorial X!! DOUBLEFACTORIAL(X) defines an object with the same structure as X with elements that are equal to the doublefactorial of the corresponding elements of X.

DUMP DUMP creates an easily read complete printout of all defined objects. DUMP enables the user to examine all the information at a given point in a calculation. After the printout, the calculation continues.

DUMP is not available in TSO.

ECHO

ECHO prints out the input along with results. The first line of output is the user's input. Immediately following are the results or the response to the input.

ECHO is default for batch jobs; NOECHO is default for the interactive operation.

EDTT

EDIT is a mode available for editing a program. The EDIT mode is automatically entered when one of the words PROGRAM, PROCEDURE, EDIT, or DATA is encountered.

There are two modes available when using the SPEAKEASY editor. You can change from one mode to the other at any time by entering a null input line or by entering the command %FLIP.

In the EDIT COMMAND MODE, you may only issue commands. The commands available are:

LIST (may be abbreviated by L)
DELETE (may be abbreviated by D)
MOVE
COPY
NN followed by a statement to put in location NN
INSERT (may be abbreviated by I)
A null line stops an insert.

In the COMMAND MODE, you are prompted with the characters :% or % so you need not preceed your command with a %. (It is already supplied in this mode.)

In the INPUT MODE, you are prompted with the line number into which your input will go. This number will normally be incremented by 1 for each new line. While in the INPUT MODE you can issue a command by preficing it with the character %. Thus %LIST in the INPUT MODE will produce a listing of your file.

For a description of most of these commands, type
HELP COMMAND
where COMMAND is the name of the command desired.

EDIT XX returns the processor to the EDIT MODE with the program or datafile XX. XX is the name of a previously defined program or datafile that is to be edited.

The EDIT MODE is available on all processors but it is normally not used except for interactive operations.

EIGENSYS EIGENSYSTEM(SYMMAT) returns all the eigenvalues of SYMMAT. The eigenvalues are returned as a vector. If the SPEAKEASY call is Y=EIGENSYSTEM(SYMMAT), then Y is a vector containg the eigenvalues of SYMMAT.

SYMMAT must be a real symmetric matrix.

EIGENSYTEM(SYMMAT:+M) returns the M largest eigenvalues of SYMMAT. The M eigenvalues are in descending order as the M-components of a vector.

SYMMAT must be a real symmetric matrix.

EIGENSYSTEM(SYMMAT:-M) returns the M lowest eigenvalues of SYMMAT. The M lowest eigenvalues are returned in ascending order as the M-components of a vector.

SYMMAT must be a real symmetric matrix.

EIGENSYSTEM(X,EVECS) returns the eigenvalues and eigenvectors of X. If the SPEAKEASY call is Y=EIGENSYSTEM(X,EVECS), then Y is a vector containg the eigenvalues of X. The corresponding eigenvectors are stored row-wise in the N-by-N matrix, EVECS.

X must be a real symmetric matrix.

EIGENSYSTEM(X,V:+M) gives the M largest eigenvalues and eigenvecto s. The M largest eigenvalues are returned in descending order as a vector. Their corresponding eigenvectors are returned as the rows of the M-by-N matrix V.

X must be a real symmetric matrix.

EIGENSYSTEM(X,V:-M) gives the M smallest eigenvalues and eigenvector. The M smallest eigenvalues of X are returned as a vector. V is returned as an M-by-N matrix containing M eigenvectors corresponding to the M smallest eigenvalues. The eigenvectors are stored row-wise in V.

X must be a real symmetric matrix.

EIGENVAL EIGENVALS(X) gives the eigenvalues of the matrix X. The eigenvalues are listed in order of ascending values.

EIGENVALS(X:V) gives the eigenvalues and eigenvectors of X. EIGENVALS(X:V) defines a vector containing the eigenvalues of X in ascending order. V is a matrix whose columns are the eigenvectors of X.

EIGENVEC EIGENVECS(X) gives the eigenvectors of the matrix X. EIGENVECS(X) defines a matrix whose columns are the eigenvectors of X. The resulting matrix is unitary.

EIGENVECS(X:V) gives the eigenvectors and eigenvalues of X. EIGENVECS(X:V) defines a matrix whose columns are the eigenvectors of X. V is a vector containing the eigenvalues in ascending order.

ELLIPE ELLIPE(X) calculates the complete elliptical integral E(X). X is restricted to real values between 0 and 1. X=0 and X=1 are included.

ELLTPK ELLIPK(X) calculates the complete elliptical integral K(X).
X is restricted to real values between 0 and 1. X=0 and X=1
are included.

END END terminates a SPEAKEASY program or datafile.
During execution in the program mode, the single word END
corresponds to a return. In the manual mode of SPEAKEASY, the word
END is ignored.

ENDAUTOP ENDAUTOPRINT turns off AUTOPRINT.

ENDLOOP ENDLOOP N marks the end of the FOR loop N.
N is the name of the FOR loop that occurs in the corresponding
FOR statement.
ENDLOOP is a restricted word.
FIN and NEXT are synonyms for ENDLOOP.

EONE EONE(X) calculates the exponential integral from X to infinity.
EONE(X) calculates the exponential integral, (exp(¬t)/t)dt
from X to infinity. X may be in the ranges 0<X<170.
If X>170, this routine sets EONE(X)=0 with no error messages.
X is a real variable of any structure.

EQ .EQ. is the relational operator "equal to".
The periods must appear on both sides of the operator.
The result of A .EQ. B is the logical value true or false.
In SPEAKEASY, these are defined as real numbers with the
values 1 and 0 respectively.

ERF ERF(X) calculates the error function of X.
ERF(X) = 2/sqrt(pi) intergral(0,X) exp(-t**2)dt
The result has the same structure as X with elements equal
to the error function of the corresponding elements of X.
The elements of X must be real.

ERFC ERFC(X) calculates the complementary error function of X.
ERFC(X)=1-ERF(X)
The result has the same structure as X with elements that
correspond to the complementary error functions of the corresponding elements of X.
The elements of X must be real.

ERRORS ERRORS are ambiguities making it impossible to carry out a statement. There are two types of errors in SPEAKEASY. They are syntax errors and execution errors. Syntax errors are those due to misuse of symbols such as parenthesis imbalence. Such errors are detected before execution of the SPEAKEASY statement. Execution errors relate to an inconsistency in the use of defined objects. Errors encountered during compilation of a SPEAKEASY program do not inhibit later execution.

EXECUTE EXECUTE NAME executes the stored program, NAME. The stored program is executed by this command and continues until a RETURN statement or END statement is encountered. These statements terminate the execution of the stored program and the statement following the one calling for the execution of the stored program is then executed.

EXECUTE NAME:LABEL executes the program NAME beginning at LABEL. EXECUTE NAME:LABEL causes the stored program with this name to begin execution at the designated internal label.

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

EXECUTE is a restricted word.

EXP EXP(X) defines the exponential function. EXP(X) defines an object with the same structure as X but with elements equal to the exponential of the corresponding elements of X.

Each element Xi of X must satisfy the conditions Real(Xi)<170 and |Imag(Xi)|<5E6.

FIN FIN is a synonym for ENDLOOP.

FIN N marks the end of the FOR loop N. N is the name of the loop appearing in the corresponding FOR statement.

FIN is a restricted word.

FOR

FOR N=START,STOP,INC designates a loop in a program. N is the name of a scalar that may appear in any context in the loop without altering its value. START, STOP, and INC are scalar expressions not involving N. START is the initial value of N. STOP is the final value of N. INC is the increment to be added to N every time the loop is repeated. START, STOP, and INC may be any real numbers. The loop is executed for N=START,START+INC, ... and is done for all values of N that are inside the closed range (START STOP) or (STOP START) if STOP<START. Note that STOP may be <START in which case INC must be <0. If INC is not specified, it is assumed to be +1 or -1, which ever is appropriate.

A FOR loop is terminated by an ENDLOOP statement or by a FIN statement.

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

Since FOR loops are neither efficient nor compact in SPEAKEASY, they should be avoided. Logical transfers into and out of FOR loops should be used with extreme caution.

FOR is a restricted word.

FRAC

FRAC is a synonym for FRACPART.

FRAC(X) returns the fractional part of X. FRAC(X) defines an object with the same structure as X but the elements of the result are the fractional part of the corresponding elements of X. The fractional part of a negative number is the negative of the digits to the right of the decimal point; ie FRAC(-1.2) = -.2

FRACPART

FRACPART(X) returns the fractional part of X. FRACPART(X) defines an object with the same structure as X but the elements of the result are the fractional part of the corresponding elements of X. The fractional part of a negative number is the negative of the digits to the right of the decimal point; ie FRACPART(-1.2) = -.2

A synonym is FRAC.

FREE

FREE(N1,N2,...,NN) frees the definitions of the objects N1,N2,...,NN. The core is then available for use by other objects. N1,N2,...,NN are names of defined objects or programs.

GAMMA

GAMMA(X) defines the gamma function of X. GAMMA(X) defines an object with the same structure as X but with elements equal to the gamma function of the corresponding elements of X.

The elements Xi of X must be real and must satisfy the condition 10E-50<Xi<56.

GE .GE. is the relational operator "greater than or equal to". The periods must be on both sides of the operator. The result of A .GE. B is the logical value true or false. In SPEAKEASY, these are defined as real numbers with the values of 1 or 0 respectively.

GEIGEN GEIGEN(M,METRIC) finds the eigenvalues of the matrix M. M is a real symmetric matrix in a non-orthogonal basis. METRIC is the positive definite matrix of the inner products of the basis states. The eigenvalues of M are returned as the components of a vector.

GEIGEN(M,METRIC,EVECS) finds the eigenvalues and eigenvectors of M. M is a real symmetric matrix in a nonorthogonal basis. METRIC is the positive definite matrix of the inner products of the basis states. EVECS is a matrix containing the eigenvectors of M stored row-wise. The eigenvalues of M are returned as the components of a vector.

GO GO causes execution of a program to resume from the holding mode. GO is used in the holding mode after statements are entered to cause execution of the program to continue from the point where it was interrupted by PAUSE, INPUT, or STOP.

Synonyms for GO are CONTINUE, RESUME, and a null line.

GO may only be used in the holding mode.

GOTO GOTO X causes execution to be transferred to the statement labeled X. The GOTO statement is used to alter the sequence of execution of statements. Logical branches are created by combining an IF statement with a GOTO statement.

GOTO is a restricted word.

An equivalent form is GO TO.

GRAPH GRAPH(I:J) plots the members of I versus the object J. I is in the vertical direction and J is along the horizontal.

All objects should be 1-dimensional and real and have the same number of elements. A 2-dimensional object in I is treated as several 1-dimensional objects, each consisting of a row of the original 2-dimensional object. Thus a 2-dimensional object in I must have as many columns as J has elements.

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

The graphical output from GRAPH in the program mode is plotted by CALCOMP.

GRAPH is available only in version GRAPHEZ.

GRID

GRID(I,J) defines a 1-dim. array of 101 points from I to J. The increment is chosen by the computer so that the points are equally spaced. Both points I and J are included.

A synonym is VARIABLES(I,J).

GRID(I,J,K) defines a 1-dim. array I,J+K,I+2K...until J is reached. The last element is equal to or less than J depending on the increment K.

GRID(I,J,K) is not equal to VARIABLE(I,J,K) unless K is such that I+NK=J.

GT

.GT. is the relational operator "greater than". The periods must be on both sides of the operator. The result of A .GT. B is the logical value true or false. In SPEAKEASY, these are defined as real numbers with the values of 1 or 0 respectively.

HENCEFOR

HENCEFORTH X IS Y redefines the word X to mean Y. After this statement, anytime the word X is encountered it will be treated as if it were the word Y.

The word 'IS' in the expression is arbitrary. Any word may be used.

Restricted words may not be given synonyms by this method.

HIERARCH

HIERARCHY is the order in which mathematical operations are done. The HIERARCHY in SPEAKEASY is:

** raising to a power,  
/,* division and multiplication,  
+,- addition and subtraction.

Operations of the same hierarchy are done left to right.

HIGHWIDE

HIGHWIDE activates HIGH-WIDE arithmetic for arrays. HIGH-WIDE arithmetic conforms to the HIGH-WIDE convention. The HIGH-WIDE convention is that 2-dimensional arrays are compatible for arithmetic operations if the height (number of rows) of each array is either 1 or some constant value H and the width (number of columns) of each array is either 1 or some constant value W. The resulting array defined by the operation has H rows and W columns.

For example, an Nx1 array times an NxM array yields an NxM array; an NxM array times a 1xM array yields an NxM array; an Nx1 array times a 1xM array yields an NxM array.

An alternate call to turn on the HIGH-WIDE convention is HIGHWIDE(ON).

HIGHWIDE(OFF) turns off the HIGH-WIDE convention for array operations.

A shortened form is HIWIDE.

HIWIDE HIWIDE is a synonym for HIGHWIDE.
HIWIDE activates HIGH-WIDE arithmetic for arrays. HIGH-WIDE arithmetic conforms to the HIGH-WIDE convention for arithmetic operations. The HIGH-WIDE convention is that 2-dimensional arrays are compatible for arithmetic operations if the height (number of rows) of each array is either 1 or some constant value H and the width (number of columns) of each array is either 1 or some constant value W. The resulting array defined by the operation has H rows and W columns.

For example, an Nx1 array times an NxM array yields an NxM array; an NxM array times a 1xM array yields an NxM array; an Nx1 array times a 1xM array yields an NxM array.

An alternate call to turn on the HIGH-WIDE convention is HIWIDE(ON).

HTWIDE(OFF) turns off the HIGH-WIDE convention for array operations.

HLABEL HLABEL='ANY MESSAGE' labels the hôrizontal scale of a graph. The desired label may be anything. Its beginning and end are marked by apostrophes in the specifying statement.

HLABEL is available only in version GRAPHEZ.

HSCALE HSCALE=(LEFT,RIGHT) specifies the horizontal limits of a graph. LEFT is the left limit and RIGHT is the right limit of the scale. Values are labeled at inch marks. The default limits for the horizontal scale are 0 for the left and 10 for the right.

HSCALE is used for graphical output.

HSIZE HSIZE=X specifies the horizontal size of the graph. X is in inches. The default is 10 inches.

HSIZE is used for graphical output.

IF IF (X) Y is a conditional statement for scalar operations. X is a scalar expression. If the numerical value of the expression X is nonzero, then the associated statement Y is carried out; otherwise the statement Y is ignored.

IF is a restricted word.

IMAG IMAG(X) returns the imaginary part of X.
IMAG(X) defines an object with the same structure as X but with elements that are the imaginary part of the corresponding elements of X.

A synonym is IMAGPART.

IMAGPART IMAGPART(X) returns the imaginary part of X.
IMAGPART(X) defines an object with the same structure as X but with elements that are the imaginary part of the corresponding elements of X.

A synonym is IMAG.

INPUT INPUT A,B,...,C puts the system in a holding mode.
The word INPUT may be followed by any statement.

Statements entered in the holding mode are not restricted to the implied requests of the INPUT statement. The only restriction is that the EXECUTION mode may not be used for if it is used, the ability to resume from this point at a later time is lost.

Execution of the program is resumed by entering either RESUME, CONTINUE, GO, or a null line.

Synonyms for INPUT are PAUSE and STOP.

INPUT is only available in the PROGRAM mode.

INSERT INSERT N(I) inserts the statements that follow at N,N+I,N+2I,....
INSERT N(I) inserts the statements that follow the words INSERT N(I) into the positions N,N+I,N+2I,...

INSERT N inserts the statements that follow at N, N+1,N+2,...
INSERT N inserts the statements that follow the insert command into locations labeled by N,N+1,N+2,...

A null line stops the insert.

INSERT may be abbreviated by I.

INSERT is one of the commands available in the EDIT mode.

INTEG INTEG is a synonym for INTEGRAL.

INTEG(F:X) defines the integral of F with a variable upper limit. X is a 1-dimesional array (X1,X2,X3,...,XN). F is a 1-dimensional array which is a function of the array X. The result is a 1-dimensional array the same size as X. Each element of the result is the integral from X1 to Xi where Xi is the corresponding element of X. The integration is done by the trapezoidal rule.

INTEGERS INTEGERS(I,J,K) defines an array with the integers I,I+K,I+2K,...,J. The array ends when the value J is reached or passed.

INTEGERS(I,K) defines an array with the integers I,I+1,I+2,...,K. If K is less than I, INTEGERS(I,K) defines the integer array I,I-1,I-2,...,K.

A shortened form is INTS.

INTEGRAL INTEGRAL(F:X) defines the integral of F with a variable upper limit. X is a 1-dimensional array (X1,X2,X3,...,XN). F is a 1-dimensional array which is a function of the array X. The result is a 1-dimensional array the same size as X. Each element of the result is the integral from X1 to Xi where Xi is the corresponding element of X. The integration is done by the trapezoidal rule.

A shortened form is INTEG.

INTERP INTERP is a synonym for INTERPOL.

INTERP(Y,F,X) defines the function F at the points Y. INTERP(Y,F,X) is an interpolation method which is cubic except in the first and last intervals where it is quadratic. The result is an array with the values of the function F evaluated at the points Y. X and Y are arrays. If points in Y are outside the range, the interpolation is based on the last 3 points in the range. F(X) must be given.

INTERPOL INTERPOL(Y,F,X) defines the function F at the points Y. INTERPOL(Y,F,X) is an interpolation method which is cubic except in the first and last intervals where it is quadratic. The result is an array with the values of the function F evaluated at the points Y. X and Y are arrays. If points in Y are outside the range, the interpolation is based on the last 3 points in the range. F(X) must be given.

A synonym is INTERP.

INTPART INTPART(X) returns the integer part of X. INTPART(X) defines an object with the same structure as X but the elements of the result are the integer part of the corresponding elements of X. The integer part of a negative numbers is the negative of the integers to the left of the decimal point; ie INTPART(-1.2) = -1

A synonym is WHOLE.

INTS INTS is a synonym for INTEGERS.

INTS(I,J,K) defines an array with the integers I,I+K,I+2K,...,J. The array ends when the value J is reached or passed.

INTS(I,K) defines an array with the integers I,I+1,I+2,...,K. If K is less than I, INTS(I,K) defines the integer array I,I-1,I-2,...,K.

INT2 INT2(X) is a function that returns an integer2 object. X may be real8, real4, or integer4. The result is INTEGER2. This operation is used to store large arrays in packed form - it is not needed in most applications.

INT4 INT4(X) is a function that returns an integer4 object. X may be real4, real8, or integer2. The result is integer4. This operation is used to store large arrays in packed form - it is not needed in most applications.

INVERSE INVERSE(X) defines the inverse of the matrix X. The result is a matrix.

An alternative statement to obtain the inverse of the matrix X is 1/X.

KEEP KEEP XX saves the SPEAKEASY object XX. The object XX (either a variable, program or data file) is saved in the private library defined by the MYKEEP file. KEPT may be used to retrieve the object at a later time. The TSO command ISPEAKEP may be use to create a partioned data set for KEEP and the TSO command SPEAKEEP may be used to start a SPEAKEASY session that may use KEEP or KEPT (i.e. SPEAKEEP is typed instead of SPEAKEZ).

The most general form of KEEP is KEEP XX AS YY ON ZZ which causes XX to be saved in the file ZZ with the name YY. KEEP XX AS YY is an intermediate form.

KEPT KEPT(XX) retrieves the SPEAKEASY object XX. The object XX (either a variable, program or data file) is retrieved from the private library defined by the MYKEPT file or if there is no MYKEPT file from the MYKEEP file. XX must previously have been put in the library by means of the KEEP statement. If XX is a variable, KEPT(XX) is a variable of the same type and may be used in expressions such as

```
  XX = KEPT(XX)     or      Z = KEPT(XX)/5 + KEPT(Y).
```

If XX is a program or data file the statement KEPT (XX) restores XX to core and makes it available for subsequent use (EDIT, EXECUTE, etc.)

The most general form of KEPT is KEPT(XX ON ZZ) which uses the file ZZ for the library.

See KEEP for the TSO commands for using KEEP and KEPT.

LABEL LABEL refers to a statement label. SPEAKEASY statements are labeled by a name of 8 symbols or less, beginning with an alphabetic character. This label is separated from the statement by a colon. The label may be in any column but preceeds the statement.

LE .LE. is the relational operator "less than or equal to". The periods must appear on both sides of the operator. The result of A .LE. B is the logical value true or false. In SPEAKEASY, these are defined as real numbers with the values of 1 or 0 respectively.

LENGTH LENGTH(X) defines a scalar that is the number of elements in X. If X is undefined, the result is zero.

A synonym is NOELS.

LIBINDEX LIBINDEX(Y) defines the names of the members of Y. LIBINDEX(Y) defines a literal 1-dimensional array with the names of the members of the library Y as components of the array.

LIBNAMES LIBNAMES returns the names of the SPEAKEASY libraries. LIBNAMES defines an array whose elements are the names of the libraries attached to the SPEAKEASY processor.

LINKLIB LINKLIB='XX' adds the LINKULE library XX to those available. XX is an additional library of linkules that is searched if a designated member is not found in the other libraries.

LINKULES LINKULES is a library of operations that are FORTRAN subroutines. LINKULES are usually efficient routines for carrying out specific mathematical operations. However, any FORTRAN subroutine can be joined to SPEAKEASY by an appropriate interface and become a LINKULE.

To obtain a listing of the present LINKULES, use the statements

```
T=LIBINDEX(KEPT)
TABULATE T
```

LIST LIST MEMBER X OF LIBRARY Y FROM I TO J lists lines I through J OF X. X is the name of a member of library Y to be listed. The words MEMBER, LIBRARY, FROM, and TO are keywords. If TO is omitted, the listing begins at line I and goes to the end of the member. If FROM is also omitted, the entire member is listed. LIBRARY Y specifies the library to be searched for the member. The default library is PROCLIB. If the member is not specified, the library index is listed.

LIST in the EDIT mode lists the edit file. The entire file is listed unless statement numbers follow the word LIST. LIST may be abbreviated by L in the EDIT mode.

LIST N in the EDIT mode lists line N of the edit file.

LIST N,M in the EDIT mode lists lines N through M.

LISTHEAD LISTHEAD(N,XX,YY) lists N lines of member XX of library YY.
N specifies the maximum number of lines of XX to be listed.
XX is the name of the member to be listed.
YY is the name of the library to be searched for member XX.
If YY is not specified, PROCLIB is default.

LISTMEMB LISTMEMBER(XX,YY) lists member XX of library YY.
XX is the name of the member to be listed.
YY is the name of the library.
If YY is omitted, the default is PROCLIB.
The entire member is listed.
An alternative form is LIST MEMBER XX OF LIBRARY YY.

LISTPROG LISTPROG specifies that the program is to be listed.
If not specified, the default for batch jobs is usually LISTPROG.
In the interactive operation, DONTLIST is default.

LOADDATA LOADDATA(A,NAME) loads A with 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.
LOADDATA is a restricted word.

LOC LOC is a synonym for LOCS.
LOC(X) gives the indices of the nonzero (true) elements of X.
X is a 1-dimenional object. LOC is a logical function and the argument X is often an array with a relational operator. The result is usually used as a structured index to produce a new array.

LOCMAX LOCMAX(X) specifies the location of the maximum element of X.
X must be a 1-dimensional array.

LOCMIN LOCMIN(X) specifies the location of the minimum element of X.
X must be a 1-dimensional array.

LOCS LOCS(X) gives the indices of the nonzero (true) elements of X.
X is a 1-dimensional object. LOCS is a logical function and the argument is often an array with a relational operator. The result is usually used as a structured index to produce a new array.
LOC is a synonym for LOCS.

LOG LOG(X) returns the natural logarithm of X.
LOG(X) defines an object with the same structure as X but with elements equal to the natural logarithm of the corresponding elements of X.
LOG(X) is not defined if any element in X is 0.

LOGGAMMA LOGGAMMA(X) defines the natural logarithm of the gamma function of X. LOGGAMMA(X) defines an object with the same structure as X but with elements that are the natural logarithm of the gamma function of the corresponding elements of X.

The elements Xi of X must be real and satisfy the condition 0<Xi<4E60.

LOWERTRI LOWERTRI(X) returns the lower triangular part of X. The result is a 1-dimensional object of the same family as X with elements corresponding to the lower triangular elements and the diagonal elements of X.

X must be a 2-dimensional square object.

LT .LT. is the relational operator "less than". The periods must be on both sides of the operator. The result of A .LT. B is the logical value true or false. In SPEAKEASY, these are defined as real numbers with the values of 1 or 0 respectively.

MARGINS MARGINS enables the user to contol the margins of the output.

MARGINS(N,M) specifies the left and right limits of output. MARGINS(N,M) means that the output is restricted to columns starting at N and ending with M.

MARGINS(M) is equivalent to MARGINS(1,M).

MARGINS(0,M) is equal to MARGINS(1,M) without control functions. MARGINS(0,M) eliminates all carriage control characters that are used to control the vertical spacing and to position output on the top of a new page.

MAT MAT is a synonym for MATRIX.

MAT(N,M:) defines an N-by-M matrix. If no additional arguments are given, the matrix has all elements set to zero.

MAT(N,M:I,J,...,K) defines an N-by-M matrix and presets the elements. The elements are set row by row by use of the values I,J,...,K or the elements of I,J,...,K if they are structured objects. If a complex element is encountered, then a complex matrix is defined. If all the elements are not preset by the element list, the unspecified elements are set to zero.

MATRIX MATRIX(N,M:) defines an N-by-M matrix. If no additional arguments are present, the matrix has all elements set to zero.

A shortened form is MAT.

MATRIX(N,M:I,J,...,K) defines an N-by-M matrix with preset elements. The elements are set row by row by use of the values I,J,...,K or the elements of I,J,...,K if they are structured objects. If a complex element is encountered, then a complex matrix is defined. If all the elements are not specified by the element list, the unspecified elements are set to zero.

MAX MAX(X) specifies the value of the maximum element in X.

MAXOFCOL MAXOFCOL(X) returns the largest element in each column of X. X is a 2-dimensional array or a matrix. The result is a 1-dimensional object of the same family as X with elements that are the largest element of each column of X.

MAXOFROW MAXOFROW(X) returns the largest element in each row of X. X is a 2-dimensional array or a matrix. The result is a 1-dimensional object of the same family as X with elements that are the largest element of each row of X.

MELD MELD(I,J,...,K) gives an odometer ordering of elements of I,J,...,K. The arguments I,J,...,K must be 1-dimensional objects. MELD alters the structure of I,J,...,K so that every element in each object of the argument list is associated with every element of every other object in the list. The last argument varies most rapidly.

After the statement MELD(I,J,...,K), any function of I,J,...,K can be written in a straightforward manner.

For a further explanation and examples, refer to the SPEAKEASY-3 manual or the linkule document.

MFAM MFAM(X) defines a member of the matrix/vector family. The result has the same structure as X. If X is a 1-dimensional array or a vector, the result is a vector. If X is a 2-dimensional array or a matrix, the result is a matrix.

MFAM and VFAM are synonyms.

MIN MIN(X) specifies the value of the minimum element in X.

MINOFCOL MINOFCOL(X) returns the smallest element in each column of X. X is a 2-dimensional array or a matrix. The result is a 1-dimensional object of the same family as X with elements that are the smallest element of each column of X.

MINOFROW MINOFROW(X) returns the smallest element in each row of X. X is a 2-dimensional array or a matrix. The result is a 1-dimensional object of the same family as X with elements that are the smallest element of each row of X.

MOVE MOVE N,M,K(I) moves statements N through M to K,K+I,K+2I,...
MOVE N,M,K moves statements N through M to K,K+1,K+2,...
MOVE N moves statement N to the last position.
MOVE N,M moves statements N through M to the last positions.
MOVE is a command available in the EDIT mode.

MYDOCS MYDOCS is the library name for private documents. MYDOCS is the library name for private documents other than the private HELP documents that are attached to the SPEAKEASY processor.

MYHELP MYHELP is the library name for private HELP documents. MYHELP is the library name for private HELP documents that are attached to the SPEAKEASY processor.

MYKEEP MYKEEP is a library for private information. MYKEEP is a library used to save information. Information is stored in this library by the use of the command KEEP. It can be retrieved by the use of the word KEPT. The contents of this library can be listed by the use of the command LIBINDEX.

MYKEPT MYKEPT is a library for previously kept private information. Information is retrieved from this library by the use of the command KEPT. The names of information stored in this library are obtained by use of the LIBINDEX command.

MYLINKS MYLINKS is the library name for private linkules. MYLINKS is the library name for private linkules to be attached to the SPEAKEASY processor. It is of the same form as the system library but contains personal routines. If MYLINKS is an attached library, it will be searched for a given member prior to the search of the LINKULE library.

MYPROCS MYPROCS is a private library. If no library is specified when CREATE or CREATEMEMBER is used, the member created is put in the library MYPROCS. Information in the library MYPROCS may be listed by use of the commands LIST or LISTMEMBER. A member of MYPROCS may be used by use of the commands USE or USEMEMBER.

NAMES NAMES prints the names of all currently defined SPEAKEASY objects. The command NAMES is particularly useful in the interactive mode since it provides the user with a means of obtaining a listing of the names of variables that he has previously defined.

NE .NE. is the relational operator "not equal to". The periods must appear on both sides of the operator. The result of A .NE. B is the logical value true or false. In SPEAKEASY, these are defined as real numbers with the values of 1 or 0 respectively.

NEUMANN NEUMANN(NU,X) returns cylindrical Bessel fns. of the second kind. NU and X are real and nonnegative. There is a SPEAKEASY restiction that (X+NU)<400.

For further details of accuracy and the method used for the calculation refer to the linkule document.

NEWGRAPH NEWGRAPH causes the next graph to be drawn on a new area of paper. NEWGRAPH may be used in conjunction with ADDGRAPH to completely avoid the use of the statement GRAPH.

NEWGRAPH is available only in version GRAPHEZ.

NEWPAGE NEWPAGE causes the printer to start a new page. The output following the control statement NEWPAGE is printed on the top of the next page.

NEWPAGE is a restricted word.

NEWS NEWS is information of current interest to users. This document will list current news about the SPEAKEASY system.

NOCOLS NOCOLS(X) defines a scalar which is the number of columns in X.

NOECHO NOECHO suppresses the echoing of input data. This command causes just the results to be printed out. This statement may occur at any place in the program.

NOECHO is default for the interactive operation; ECHO is default for batch jobs.

NOELS NOELS(X) defines a scalar equal to the number of elements in X.
If X is undefined, the result is zero.
A synonym is LENGTH.

NORATION NORATIONALIZE stops rationalization of subsequent output.
There are no arguments for NORATIONALIZE.
RATIONALIZE (see HELP document) is used to initiate the rationalization of output.

NOROOTS NOROOTS(F) gives the number of roots of the function F.
The number of times the function F changes sign (goes through zero) is found.
A synonym is NOZEROS.

NOROWS NOROWS(X) defines a scalar equal to the number of rows in X.

NOT .NOT. is the logical operator "not".
The periods must be on both sides of the operator.
.NOT. A = 1 if A is zero; otherwise .NOT. A = 0.
A may be any real number.

NOZEROS NOZEROS(F) gives the number of zeros of the function F.
The number of times the function F changes sign (goes through zero) is found.
A synonym is NOROOTS.

NUMBERS NUMBERS refers to numerical data.
All numerical data are specified in decimal form. The lack of a decimal point implies that it belongs just to the right of the last digit.
NUMBERS can be real, imaginary, or complex.
A terminal I implies that the previous number was imaginary.
Examples of imaginary and complex numbers are: 21.6I; 6.1+5.2I
Very large or very small numbers may be expressed by ending the number in E followed by the power of 10 associated with that number. For example, 1.052E+2 means 105.2. If a number is imaginary, the I comes before the E, eg., 6.2IE-3.

OBJECT OBJECT(I,J,...,K) generates an inplace name.
Each of the arguments I,J,...,K can be either a literal quantity or a nonnegative number. For literals, the expression itself is used. For numbers, the integer part of the number is used as a literal. The various arguments are then joined together to compose the name.
The expression OBJECT(I,J,...,K) may occur anywhere in a SPEAKEASY statement. It is particularly usefull combined with the word HENCEFORTH.

OMITCLAS OMITCLASS suppresses the printing of the class.
If OMITCLASS is not specified, the class of the resulting structured objects is printed out when they are printed. The statement OMITCLASS suppresses the printing of the class.

ONERROR ONERROR(X,Y) specifies the action to be taken after an error.
X may be DUMP or NODUMP.
The options DUMP or NODUMP specify whether all defined data is to be printed or not.
Y may be MANUAL or CONTINUE.
MANUAL means that computation is continued in the manual mode.
CONTINUE means that the error does not affect the path of execution.
The default options are DUMP,MANUAL.

OR .OR. is the logical operator "or".
The periods must be on both sides of the operator.
A .OR. B = 1 if either A or B is nonzero; otherwise A .OR. B = 0. A and B may be any real numbers.

ORDERED ORDERED(X) gives the elements of X in increasing order.
ORDERED(X) defines an object with the same structure as X but whose elements are the elements of X arranged in order of increasing magnitude.
A synonym is RANKED.

ORDERER ORDERER(X) gives the indices of the ordered elements of X.
ORDERER(X) gives the indices of the elements of X arranged in order of increasing magnitude. X must be a 1-dimensional object.
A synonym is RANKER.

PAUSE PAUSE puts the system in a holding mode.
The word PAUSE may be followed by any statement such as
PAUSE FOR INPUT
Operations in the holding mode are completely general except that the EXECUTION mode should not be used for if it is used the ability to resume from this point at a later time is lost.

Execution of the program is resumed by entering either RESUME, CONTINUE, GO, or a null line.

Synonyms for PAUSE are STOP and INPUT.

PAUSE is only available in the program mode.

PLOTSYMB PLOTSYMB(N,M) specifies the symbols used and frequency of points.

N is an integer which determines the frequency with which each symbol is plotted. (1 means every point, 2 means every other point, 3 means every third, etc.) A negative value for N indicates that only the symbol should appear. A positive value means that a line should join successive points.

M is an integer 0 through 12 that designates one of 13 different symbols to be used in plotting data. The integers and their corresponding symbols are:
0 is a square.
1 is a circle.
2 is a triangle.
3 is a plus (+).
4 is an X.
5 is a diamond.
6 is an arrow pointing upward.
7 is an X with a bar on top.
8 is a Z.
9 is a Y.
10 is an X with a circle.
11 is an asterisk (*).
12 is an X with a bar on top and bottom.

PLOTSYMB is used for graphical output.

PLOTTITL PLOTTITLE='ANY MESSAGE' titles a graph.
The title may be anything. Its beginning and end are designated by apostrophes in the specifying statement. The title is printed at the top of a graph.

PLOTTITLE is only available in version GRAPHEZ.

PRINT PRINT(A,B,C,...,Z) specifies that A,B,C,...,Z are to be printed. The user may control the form of output by using the control words MARGINS, SIGNIFICANCE, AUTOTAB, TABULATE, and COLWIDTH.

PRINT is a restricted word.

PRINTCLA PRINTCLASS prints the class of structured objects. PRINTCLASS causes the class of structured objects to be printed when the structured objects are printed. PRINTCLASS is default. To turn it off, use OMITCLASS.

PROCLIB PROCLIB is a library of stored SPEAKEASY statements. The proceedures in this library are available as input and may be used by a statement of the form
USE MEMBER NAME
where NAME is a proceedure in PROCLIB.

PROD PROD(X) defines the product of the elements of X. The result is a scalar.

PRODCOLS PRODCOLS(X) multiplies the elements in each column of X. PRODCOLS(X) defines a 1-dimensional object that is a member of the family of X. Each element of the result is the product of the elements in a column of X.

PRODROWS PRODROWS(X) multiplies the elements in each row of X. PRODROWS(X) defines a 1-dimensional object that is a member of the family of X. Each element of the result is the product of the elements in a row of X.

PROGRAM PROGRAM NAME gives a name to a program. A program is a SPEAKEASY procedure. Naming a program enables the user to refer to that program by name to store it, to use it within another program, or to execute it.

A program is a defined object. If its name is identical to a previously defined object, its definition will replace that object.

PROGRAM is a restricted word.

PUNCH PUNCH(F:X) punches the object X on cards in the format F. The PUNCH statement gives the user a means of transmitting information to other non-SPEAKEASY programs.

The format F is specified by a statement of the form F='(FORMAT)' where FORMAT is the standard 360-FORTRAN IV format excluding fixed-point form (integers). The F, D, and E formats are equally acceptable.

X is the array to be punched.

QUIT QUIT terminates execution of a SPEAKEASY program. When all input is processed in normal batch operation, execution automatically terminates. This word is therefore not normally used for such cases.

RANDOM RANDOM(X) generates random numbers. The result has the same class and structure as X but its elements are random numbers between 0.0 and 1.0.

RANDOM will produce the same numbers on successive calls unless the first use of it is preceded by the statement LOAD(RANDOM).

RANKED RANKED(X) gives the elements of X in increasing order. RANKED(X) defines an object with the same structure as X but whose elements are the elements of X in increasing order.

A synonym is ORDERED.

RANKER RANKER(X) gives the indices of the ordered elements of X. RANKER(X) gives the indices of the elements of X arranged in order of increasing magnitude. X must be a 1-dimensional object.

A synonym is ORDERER.

RATIONAL RATIONALIZE (EPS, NDIGITS) causes rationalization of output. Subsequent speakeasy output will be printed as rational fractions if the difference of the rational form and the actual number is less than EPS in magnitude and if the total number of digits required to express the numerator and denominator of the fractional part is approximately less than NDIGITS.

EPS may be entered in one of two ways:
1) as a number of the form 10**(-N) i.e. .00 ... 01
2) as a positive integer N in which case EPS will be 10**(-N)

Note that in the first form, EPS must be a negative power of 10; .025 is, for example, invalid.
In both cases 1 <= N <= 20 must be satisfied.
The default value of N is 14 (EPS = 10**(-14) ).

NDIGITS must be an integer in the range 2 <= NDIGITS <= 18. The default value of NDIGITS is 6.

If either or both of EPS or NDIGITS is zero, the rationalization process is suppressed on future output. (The command NORATIONAL (see HELP document) also stops rationalization.)

Alternate forms of the RATIONALIZE statement are
RATIONALIZE (EPS)
RATIONALIZE
in which the default values are used for the omitted arguments.

READ READ(F:X) reads data from cards punched in format F and puts them into X. The READ statement enables the user to read information that has been punched in some specialized format.

The format F is specified by a statement of the form F='(FORMAT)' where FORMAT is the standard 360-FORTRAN IV format excluding fixed-point form (integers).

X is the array (structured object) into which the data are put. It must exist and its size must be specified. The number of cards read is the number needed to fill the array.

READ is not available in all versions of SPEAKEASY.

REAL REAL is a synonym for REALPART.

REAL(X) returns the real part of X. REAL(X) defines an object with the same structure as X but the elements of the result are the real part of the corresponding elements of X.

REALPART REALPART(X) returns the real part of X. REALPART(X) defines an object with the same structure as X but the elements of the result are the real part of the corresponding elements of X.

A synonym is REAL.

REAL4 REAL4(X) is a function that returns a real4 number. X may be real8, integer4, or integer2. The result is real4.

REAL8 REAL8(X) is a function that returns a real8 number. X may be real4, integer4, or integer2. The result is real8.

RECLASS RECLASS(A:B,C,...,Z) alters the structure of B,C,...,Z. RECLASS(A:B,C,...,Z) alters the structure of B,C,...,Z to agree with A. B,C,...,Z are previously defined objects with the same number of elements as A.

RESUME RESUME causes execution of a program to resume from the holding mode. RESUME is used in the holding mode after statements are enterred to cause execution of the program to continue from the point where it was interrupted by PAUSE, INPUT, or STOP.

Synonyms for RESUME are CONTINUE, GO, and a null line.

RESUME may only be used in the holding mode.

RETURN

RETURN returns execution to the program calling the stored program. The statement executed after the statement RETURN is the one following the EXECUTE statement that invoked the stored program. A RETURN statement is always implied before the END card of any program.

RETURN is restricted to the program mode. It is ignored if it is used in the manual mode.

RETURN is a restricted word.

ROOTS

ROOTS(F:X) finds the roots of the function F. The result is a 1-dimensional array. X is a 1-dimensional array of the same length as F. Roots defines a new object whose elements correspond to the zeros of the function F. The method of trapezoidal interpolation is used.

A synonym is ZEROS.

ROWARRAY

ROWARRAY(N:) defines a 2-dim. N-component array that is a row. If no further arguments are given, all N-components are set equal to zero.

ROWARRAY(:I,J,...,K) defines a 2-dim. row array with preset els. The components are preset by the element list I,J,...,K. If any argument of the element list is structured, then the elements of that structured object are used. If a complex element is encountered, then a complex 2-dimensional row array is defined.

ROWARRAY(N:I,J,...,K) defines a 2-dim. N-component row array. The components are preset by the element list I,J,...,K. If any argument of the element list is structured, then the elements of that structured object are used. If a complex element is encountered, then a complex 2-dimensional row array is defined. If all N-components are not preset by the element list, the unspecified components are set equal to zero.

ROWMAT

ROWMAT(N:) defines a 1-by-N matrix which is a row. If no further arguments are given, all N elements are set equal to zero.

ROWMAT(:I,J,...,K) defines a row matrix and specifies the els. The element list I,J,...,K specifies the elements of the matrix. If an argument of the list is structured, the elements of that structured object are used. If one of the elements is complex, the matrix is complex.

ROWMAT(N:I,J,...,K) defines a 1-by-N row matrix and specifies the els. The elements are preset by the element list I,J,...,K. If an argument of the element list is structured, the elements of that structured object are used. If an element is complex, the matrix is complex. If all N elements are not specified by the element list, the unspecified elements are set equal to zero.

ROWMAX ROWMAX(X) specifies the row containing the maximum element of X.

ROWMIN ROWMIN(X) specifies the row containing the minimum element of X.

RUN RUN is equivalent to COMPILE followed by the command EXECUTE. RUN used in the EDIT mode returns the processor from the EDIT mode to the MANUAL mode, compiles the edited program, and then executes it.

SELECT SELECT(A,B,...,C:I) truncates or expands A,B,...,C using the index I. A,B,...,C are 1-dimensional objects. I is a structured index that acts as a control array. A,B,...,C are truncated or expanded so their final components are those indexed by the elements of the control array I.

For an example, refer to the SPEAKEASY-3 manual.

SETGAUSS SETGAUSS(N,X,W,XLO,XHI) returns Gauss-Legendre coords. and weights. SETGAUSS(N,X,W,XLO,XHI) defines the N-point Gauss-Legendre quadrature coordinates and weights with the coordinates in the interval XLO to XHI. The arguments XLO and XHI may be omitted. If XLO and XHI are omitted, the interval is -1 to +1.

N is the number of points required. N must satisfy the condition 0<N<51.

X is returned as a 1-dimensional array whose elements are the coordinates.

W is returned as a 1-dimensional array whose elements are the weights.

The points and weights are used as
Integral(XLO to XHI)F(x) dx = SUM(W*F(X)).

For further information about the method of calculation and the accuracy, refer to the LINKULE document.

SETINFIN SETINFINITY(VAL) specifies an upper limit to the numbers printed. Any number whose absolute value is greater than VAL is printed as INF for infinity. The default value is 1.E+30.

SETJACOB SETJACOBI(N,X,W,A,B,TX,TW) defines Gauss-Jacobi coords. and weights. SETJACOBI(N,X,W,A,B,TX,TW) defines the Gauss-Jacobi quadrature coordinates and weights for the interval -1 to +1 and gives the user an idea of the precision with which the coordinates and weights are calculated. The coordinates are the roots of the Jacobi polynomial of degree N and order (A,B). A, B, TX, and TW are optional arguments and may be omitted. If A and B are omitted, the order of the polynomial is 0.

N is the number of points required. N must be an integer satisfying the condition 0<N<51.

X is returned as a 1-dimensional array whose elements are the roots of the Jacobi polynomial of degree N and order (A,B).

W is returned as a 1-dimensional array whose elements are the weights of the integration proceedure.

A and B are real numbers specifying the order of the polynomial. A must satisfy the condition A>-1. B must satisfy the condition B>1. The default value for A and B is 0.

TX is the theoretical sum of the coordinates.

TW is the theoretical sum of the weights.

The coordinates and weights are used as

Integral(-1,1) (1-x)**A (1+x)**B F(x) dx = SUM(W*F(X))

For further information about the method of calculation and the accuracy, refer to the LINKULE document.

SETLAGUE SETLAGUERRE(N,X,W,A) returns Gauss-Laguerre coords. and weights. SETLAGUERRE(N,X,W,A) defines the Gauss-Laguerre quadrature integration points X and weights W appropriate for evaluating an integral from 0 to infinity with weight points of X**A exp(-X). The integration points are the roots of the Laguerre polynomial of degree N and order A. A may be omitted. If A is omitted, the polynomial calculated is order 0.

N is the number of points required. N must be an integer and satisfy the condition 1<N<33.

X is returned as a 1-dimensional array whose elements are the roots of the Laguerre polynomial of order A and degree N.

W is returned as a 1-dimensional array with the weights of the integration proceedure. The factor X**A exp(-X) is included in W.

A is the order of the polynomial. The default is 0. A must be a real number greater than -1. A value of A of the order 20 could cause overflows in the determination of the weights for large values of N.

The points and weights are used as

Integral(0 to inf) F(x) x**A exp(-x) dx = SUM(W * F(X)).

SETLEGEN SETLEGENDRE(N,X,W,XLO,XHI) returns Gauss-Legendre coord. and weights. SETLEGENDRE(N,X,W,XLO,XHI) returns the coordinates X and weights W for the Nth order Gauss-Legendre quadrature formula in the interval XLO to XHI. The arguments XLO and XHI may be omitted. If XLO and XHI are omitted, the interval is -1 to +1.

N is the number of points required. N must satisfy one of the conditions 0<N<51, N=64, or N=96.

X is returned as a 1-dimensional array whose elements are the coordinates.

W is returned as a 1-dimensional array whose elements are the weights.

The points and weights are used as Integral(XLO to XHI)F(x) dx = SUM(W*F(X)).

SETLEGENDRE is equivalent to SETGAUSS for 0<N<51. The only difference is that SETLEGENDRE includes N=64 and N=96 while SETGAUSS does not.

SETLIB SETLIB(XX,YY) changes the name of library XX to YY.

SETNULL SETNULL(VAL) specifies a lower limit to numbers printed. Any number whose absolute value is less than VAL is to be printed as 0. The default value is 1.E-30.

SETPLOT SETPLOT(X,Y,Z) specifies BOX, NOBOX; SCALES, NOSCALES; LINES, POINTS.

X is either BOX or NOBOX. BOX means to draw a frame around the graph. NOBOX specifies no frame.

Y is either SCALES or NOSCALES. SCALES specifies that values at inch intervals are to be indicated. NOSCALES causes the indication of scales to be omitted.

Z is either LINES or POINTS. LINES specifies that points are to be joined by lines leaving the points unmarked. POINTS specifies that the points are to be marked with crosses and not joined by lines.

The default options are BOX, SCALES, and LINES.

SETPLOT is available only in version GRAPHEZ.

SIGN SIGN(X) specifies whether X is positive or negative. SIGN(X) defines an object with the same structure as X but with elements of the result equal to +1 for elements of X >0, equal to -1 for elements of X <0, and equal to 0 for elements of X equal to 0.

All the elements of X must be real.

SIGNIFIC SIGNIFICANCE(N) gives the number of significant figures to be printed. N is the number of significant figures desired. The default is 5 significant figures.

SIMEQ

SIMEQ(A,B) solves a set of simultaneous linear equations. A is an N-by-N matrix. B is an N-by-M matrix of constants. The result, X has the same structure as B and is the matrix that satisfies the simultaneous linear equation A*X=B.

SIN

SIN(X) returns the sine of X. SIN(X) defines an object with the same structure as X but the elements of the result are the sine of the corresponding elements of X.

Each element Xi of X must satisfy the condition |Xi|<1.E15.

SINH

SINH(X) defines the hyperbolic sine of X. SINH(X) defines an object with the same structure as X but with elements equal to the hyperbolic sine of the corresponding elements of X.

The elements Xi of X must be real and are restricted by the inequality |Xi|<170.

SIZE

SIZE=N,X specifies space for data and must be the first card. SIZE=N,X specifies the amount of data space needed and its location.

N is the number of kilobytes of storage to be set aside for SPEAKEASY data. One kilobyte is approximately 120 user-defined numbers.

X may be MAIN or LCS. LCS means the data storage area may be placed in LCS. MAIN is default.

The card specifying the size is the first one in the SPEAKEASY deck.

SMAT

SMAT is a synonym for SYMMAT.

SMAT(N:I,J,...,K) defines a symmetric N-by-N matrix. The element list is used to fill the lower triangular part (including the elements along the diagonal) by rows. The portion above the diagonal is then filled by making the matrix symmetric. If any argument in the list defining the elements is structured, the elements of that structured object are used.

SPACE

SPACE(N) skips N lines. SPACE is a restricted word.

SPHBES SPHBES(L,X) returns the spherical Bessel fn. of the first kind. The result is j sub L of X (little j). L must be a nonnegative integer. Underflow will occur for excessively large L. X must be real and nonnegative. There is a SPEAKEASY restriction that (X+L)<400.

For information on the method of calculation, refer to the LINKULE document.

SPHBESN SPHBESN(L,X) returns the spherical Bessel fn. of the second kind. The result is n sub L of X (little n). L must be a nonnegative integer. Overflow will occur for excessively large L. X must be real and nonnegative. There is a SPEAKEASY restriction that (L+X)<400.

For information on the method of calculation, refer to the LINKULE document.

SQRT SQRT(X) defines the square root of X.
SQRT(X) defines an object with the same structure as X but with elements equal to the square root of the corresponding elements of X.

If the domain of the calculation is complex, complex roots can be obtained. Otherwise, the square root of a negative number leads to an error message.

STOP STOP puts the system in the holding mode.
The word STOP may be followed by any statement such as
STOP FOR INPUT PLEASE
Any statements may be entered in the holding mode except that the EXECUTION mode should not be used since if it is used, the ability to resume from this point at a later time is lost.

Execution of the program is resumed by entering either RESUME, CONTINUE, GO, or a null line.

Synonyms for STOP are INPUT and PAUSE.

STOP is only available in the program mode.

SUM SUM(X) sums the elements of X.
SUM(X) defines a scalar object equal to the sum of the elements of X.

SUMCOLS SUMCOLS(X) sums the elements in each column of X.
SUMCOLS(X) defines a 1-dimensional object that is a member of the family of X. Each element of the result is the sum of the elements in a column of X.

SUMROWS SUMROWS(X) sums the elements in each row of X.
SUMROWS(X) defines a 1-dimensional object that is a member of the family of X. Each element of the result is the sum of all the elements in a row of X.

SUMSQ SUMSQ(X) sums the squares of the elements of X.
SUMSQ(X) defines a scalar that is the sum of the squares of all the elements of X.

SUMSQCOL SUMSQCOLS(X) sums the squares of the elements in each column.
SUMSQCOLS(X) defines a 1-dimensional object that is a member of the same family as X. Each element of the result is the sum of the squares of elements in a column of X.

SUMSQROW SUMSQROWS(X) sums the squares of the elements in each row.
SUMSQROWS(X) defines a 1-dimensional object that is a member of the family of X. Each element of the result is the sum of the squares of the elements in a row of X.

SYMBOLS SYMBOLS designate mathematical operations or special cards.
The symbols and a brief description of their meanings are:
+ is the addition operator.
- is the subtraction operator.
* is the multiplication operator.
For matrices, * implies matrix multiplication.
For vectors, * implies take the inner (dot) product.
For arrays, * is element-by-element multiplication.
If A and B are 1-dim. arrays, C=A*B is Ci=Ai*Bi, where i implies the ith element.
/ is the division operator.
For matrices, A/B implies matrix multiplication of the matrix A times the inverse of matrix B.
For vectors, / is not defined.
For arrays, division is defined element-by-element.
** raises to a power.
** is not defined for matrices.
For vectors, ** means to take the outer (cross) product.
For arrays, the operation is done element-by-element.
= means replace the object on the left by the expression on the right.
: separates the statement label from the statement or separates arguments in some words.
; separates successive statements on a single card.
& designates a continuation of the previous card.
$ sets off a comment. The comment is written between two $.
There is an implied $ at the end of a card that has a single $ on it.

SYMMAT SYMMAT(N:I,J,...,K) defines a symmetric N-by-N matrix. The element list is used to fill the lower triangular part (including the diagonal elements) by rows. The portion above the diagonal is then filled by making the matrix symmetric. If any argument in the element list is structured, the elements of that structured object are used. . A shortened form is SMAT.

TABULATE TABULATE(A,B,...,C) prints 1-dim. objects A,B,...,C in tabular form. Each column is labeled by the name of the corresponding object.

The user may control the tabulation by the specifications MARGINS, COLWIDTH, and SIGNIFICANCE. All tables may be made uniform by using AUTOTAB.

Only 1-dimensional objects may be tabulated.

TAN TAN(X) returns the tangent of X. TAN(X) defines an object with the same structure as X but the elements of the result are the tangents of the corresponding elements of X.

Each element Xi of X must satisfy the condition |Xi|<1.E15.

TIME TIME gives the time in seconds from which one starts. TIME gives the total "clock on the wall" time in seconds elaspsed since the start of the SPEAKEASY step.

TIME(0) gives the elapsed time since the start of a SPEAKEASY step. This form can be used in SPEAKEASY statements.

TOTALINT TOTALINT(F:X) defines the definite integral of F over the array X. X is a 1-dimensional array, (X1,X2,X3,...,XN). F is a 1-dimensional array which is a function of X. The result is a scalar equal to the definite integral from X1 to XN. The integration is done by the trapezoidal rule.

A shortened form is TOTINT.

TOTINT TOTINT is a synonym for TOTALINT.

TOTINT(F:X) defines the definite integral of F over the array X. X is a 1-dimensional array (X1,X2,X3,...,XN). F is a 1-dimensional array which is a function of X. The result is a scalar equal to the definite integral from X1 to XN. The integration is done by the trapezoidal rule.

TRACE TRACE(X) gives the trace of the matrix X. The trace is the sum of the diagonal elements.

TRANSP TRANSP is a synonym for TRANSPOSE.
TRANSP(X) defines the transpose of X.
TRANSP(X) defines an object that is of the same class as X and is the transpose of the object X.

TRANSPOS TRANSPOSE(X) defines the transpose of X.
TRANSPOSE(X) defines an object that is of the same class as X and is the transpose of the object X.
A shortened form is TRANSP.

TUTORIAL TUTORIAL teaches you about SPEAKEASY.
Type TUTORIAL to turn on the tutorial session.
Type MORE to get the first and subsequent pages of the tutorial.
Only a few pages are available at this time; more will follow.

UMAT UMAT is a synonym for UNITMAT.
UMAT(N) defines an N-by-N unit matrix.
All elements are zero except the elements along the diagonal, which have the value 1.

UNITMAT UNITMAT(N) defines an N-by-N unit matrix.
All elements are zero except the elements along the diagonal, which have the value 1.
A shortened form is UMAT.

UPPERTRI UPPERTRI(X) returns the upper triangular part of X.
X must be a square 2-dimensional object. The result is a 1-dimensional object that is of the same family as X and whose elements are the upper triangular elements including those along the diagonal of X.

USE USE MEMBER X OF LIBRARY Y causes the member X to be used as input.
If the library reference is omitted, PROCLIB is assumed. If the library MYPROCS is attached, it will be searched for the member X before the library PROCLIB is searched.
This provides the user with a simple means of supplying commonly used constants or SPEAKEASY programs to the processor.
USE is a restricted word.

USEMEMBE USEMEMBER(XX,YY) causes input of the member XX of library YY.
XX is the name of the member desired.
YY is the name of the library to be searched for member XX.

If the library reference is omitted, PROCLIB is assumed. If the library MYPROCS is attached, it will be searched for member XX before PROCLIB is searched.

This provides the user with a means of supplying commonly used constants or SPEAKEASY programs to the processor.

An alternative form is USE MEMBER XX OF LIBRARY YY.

VARIABLE VARIABLE(T,J) defines a 1-dim. array of 101 points from I to J. The increment is chosen by the computer so that the points are equally spaced. Both points I and J are included.

A synonym is GRID(I,J).

VARIABLE(I,J,K) defines the 1-dim. array I,I+K,I+2K,...,J. The last element in the array is J. If the increment K is not such that I+NK=J, then a new increment K' is chosen by the computer so that the last element is J and the elements are equally spaced.

VARIABLE(I,J,K) is equivalent to GRID(I,J,K) only when K is such that I+NK=J.

VEC VEC is a synonym for VECTOR.

VEC(N:) defines a vector with N components. If there are no additional arguments, the values of the components are set equal to zero.

VEC(:I,J,...,K) defines a vector and specifies the components. The components are preset by the argument list I, J,...,K. If any argument of the argument list is structured, then the elements of that structured object are used. The size of the vector is equal to the total number of elements specified. If a complex element is encountered, then a complex vector is defined.

VEC(N:I,J,...,K) defines an N-component vector with preset els. The components are specified by the element list I,J,...,K. If any argument of the element list is structured, then the elements of that structured object are used. If a complex element is encountered, then a complex vector is defined. If all N components are not specified, the unspecified components are set equal to zero.

**VECTOR** VECTOR(N:) defines a vector with N components. If there are no additional arguments, the values of the components are set equal to zero.

A vector is a 1-dimensional member of the matrix/vector family. A shortened form is VEC.

VECTOR(:I,J,...,K) defines a vector by presetting the components. The components are preset by the argument list I, J,...,K. If any argument of the argument list is structured, then the elements of that structured object are used. The size of the vector is equal to the total number of elements specified. If a complex element is encountered, then a complex vector is defined.

VECTOR(N:I,J,...,K) defines an N-component vector with preset els. The components are specified by the element list I,J,...,K. If any argument of the element list is structured, then the elements of that structured object are used. If a complex element is encountered, then a complex vector is defined. If all N components are not specified, the unspecified components are set equal to zero.

**VERSIONS** VERSIONS refers to the various versions of the SPEAKEASY processor. These versions differ in their space requirements and in their efficiency of operation. The present versions are:

STANDARD
BABY
GRAPHEZ
CONSOLE
TSO

For further detail about these versions, refer to the SPEAKEASY-3 manual.

**VFAM** VFAM(X) defines a member of the matrix/vector family. The result has the same structure as X. If X is a 1-dimensional array or a vector, the result is a vector. If X is a 2-dimensional array or a matrix, the result is a matrix.

**VLABEL** VLABEL='ANY MESSAGE' labels the vertical scale. The desired label is enclosed between apostrophes.

VLABEL is available only in version GRAPHEZ.

**VOCABULA** VOCABULARY generates a list of all currently defined words. The list is the actual list used by the processor in deciphering the user's SPEAKEASY program. The printout in response to the word VOCABULARY will change whenever a new word is added to the language.

VSCALE VSCALE=(BOTTOM,TOP) specifies the vertical limits of a graph. BOTTOM refers to the lower limit and TOP the upper limit of the vertical scale. Default values of the limits on the vertical scale are 0 and 8. The scale is labeled at inch marks.

VSCALE is used for versions with graphical output.

VSIZE VSIZE=Y specifies the vertical size of the graph. Y is in inches. The default is 8 inches. Y cannot exceed 10.

VSIZE is used for versions with graphical output.

WHERE WHERE(X) Y is a conditional statement for array operations. The statement X contains a structured object. The associated statement Y is an equation defining a structured object of the same size and class as the structured object in the statement X. Only the elements of the structured object in X for which the statement X is true are replaced by elements of the structured object defined by Y.

WHERE is a synonym for WHEREVER.

WHEREVER WHEREVER(X) Y is a conditional statement for array operations. The statement X contains a structured object. The associated statement Y is an equation defining a structured object of the same size and class as the structured object in the statement X. Only the elements of the structured object in X for which the statement X is true are replaced by elements of the structured object defined by Y.

A shortened form is WHERE.

WHOLE WHOLE(X) returns the integer part of X. WHOLE(X) defines an object with the same structure as X but the elements of the result are the integer part of the corresponding elements of X. The integer part of a negative number consists of the negative of the integers to the left of the decimal point; ie WHOLE(-1.2) = -1

A synonym is INTPART.

WRITE WRITE(F:X) prints the defined object X in the format F. The WRITE statement enables the user to print information in a specific format.

The format F is specified by a statement of the form F='(FORMAT)' where FORMAT is the standard 360-FORTRAN IV format excluding fixed-point form (integers).

X is the defined object to be printed in the specific format.

WRITE is not available in all versions of SPEAKEASY.

ZEROS ZEROS(F:X) finds the zeros of the function F. The result is a 1-dimensional array. X is a 1-dimensional array of the same length as F. ZEROS defines a new object whose elements correspond to the zeros of the function F. The method of trapezoidal interpolation is used. A synonym is ROOTS.

# APPENDIX

HELP documents are written as follows. Each word in the HELP documents is written in a specific format for retrieval of various levels of information. The first line of each document may be obtained by itself. Consequently, it is a one-line definition of the word. The information in each document is structured in levels so that each paragraph goes into greater detail. Eventually, the printing of a document will be halted after each paragraph so that the user will have the option of stopping or requesting further information. If further information is requested, the next paragraph will be printed.

To facilitate obtaining output on various devices, the length of each line is limited to 72 characters. For uniformity, a new paragraph is indented two spaces. Thus a line that begins a new paragraph is limited to 70 characters.

The first line of each document consists of (1) the word being defined (in capital letters), (2) the arguments of the word (in capital letters), and (3) a concise definition of the word. Since the first line is the beginning of a paragraph, it is indented two spaces so 70 characters are left for the word, argument list, and definition. Ideally this should be a complete sentence. Sometimes abbreviations must be used in order to convey the meaning of the word in so little space.

In addition to the restriction that the rest of the paragraph and the following paragraphs have lines of 72 characters or less, there is the convention that the word being defined and any of the arguments appear in capital letters throughout the document. If a word has more than one calling sequence, it is listed with each calling sequence as a new paragraph. The first line of this new paragraph begins with the word being defined and follows the same format as the first line of the document.

The last line of the document should have the initials of the contributor in columns 68—72. The contributor assumes responsibility for the validity of operation of the word described.

The documents created in this way are placed into a staging data set for validation before introducing them into the system library. However, the newly defined documents are available to SPEAKEASY users at Argonne.

HELP documents are written in upper and lower case. Consequently, words may be put in only on devices that have both upper and lower case, e.g., on the IBM-2741 terminal.

HELP documents are written in TSO EDIT using TEXT to provide upper and lower case letters. Note that the HELP documents have a blocksize of 1680 and a logical record length of 80 unlike datasets produced via the TSO EDIT TEXT NEW command. Consequently, a new HELP document is most easily created by editing an old HELP document using TEXT OLD NONUM and saving it under a new name.

# INDEX

Capitalized entries are words available in one form or other to users of the current SPEAKEASY processors. References in parentheses refer to examples of the use of the word. Underscored references refer to HELP documents. All of the HELP documents are separately indexed (see pp. 181—185). Extensive aids to locating specific words are provided there. For this reason not all HELP documents are referred to in this index.

INDEX

GPO 881-910