SPEAKEASY
AN EVOLUTIONARY SYSTEM

by

S. Cohen

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.

Presented at:
Symposium on Very High Level Languages,
Santa Monica, California, March 28-30, 1974

U of C-AUA-USAEC

MASTER

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

See

To be published in Proceedings of the Symposium on Very High Level Languages, March 28—30, 1974, Santa Monica, Calif.

Speakeasy^{*}
An Evolutionary System

S. Cohen
Argonne National Laboratory

ABSTRACT

Speakeasy is a very high level user-oriented language imbedded in a modular computer system. This system, designed for use with attached libraries for its information base, is an evolutionary development constantly being augmented in power by its user community. This paper describes the basic structure of the system and demonstrates some of its capabilities.

1. INTRODUCTION

A modern computer complex is an extremely fast information processor with a vast amount of stored information available to it. Machines and their associated software base are becoming ever faster, ever larger and ever more complex. This growth in capability brings with it increasing responsibility to provide the normal user with adequate access to available information. Unfortunately a sense of isolation from advances in computer science is common in many disciplines and results in the people most in need of the capabilities provided by a computer being forced to rely on secondary sources for solutions to their problems or to ignore advances entirely and to program using first generation computer techniques (e.g., Fortran II programming techniques are still common in "standard" programs in the physical sciences).

One answer to this problem of information exchange can be found within the structure of a modern computer system. Development of a generalized modular system for computational processing designed to access and operate with libraries of stored information can make advanced algorithms and general computational information available to users in a sensible way. Properly structured, such a system can become the basic mode of operation for many users and can provide the tools necessary for effective utilization of a computer. Inadequacies in such a system are answered by adding new capabilities to satisfy

existing needs. The overall system evolves in a manner that is sensible viewed both from the applications and the development as a whole. A system designed for growth provides the environment for a long-term project to answer the needs of users; it can at the same time provide researchers in computational science with an audience for their work.

In order for such a system to be viable it is necessary that several conditions be satisfied.

- 1) The system must contain basic sets of validated features so that a new user can formulate his problem within the system with ease and with assurance that the results are correct.
- 2) The system must be capable of growth so that inadequacies can be overcome and so that the system can be expanded beyond its original design structure.
- 3) The system must be sufficiently complex and its facilities diverse enough to satisfy the needs of a large community so that its growth and the dissemination of information within the system can be assured.
- 4) The system must be adequately documented so that users can be made aware of the system and features available within it.
- 5) A community of active users must exist.

When these conditions are satisfied then the system becomes self-perpetuating. Users of the system feel that it is valuable enough to warrant the inconvenience caused by reorienting their normal mode of operation. The growth in size of the user community in turn attracts the addition of new facilities in the literature associated with the system which in turn attracts new users. A repository for ideas contributed by many and usable by more then becomes a realistic goal.

This paper describes a system of this type that has been in use and under development for

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

several years. Originally designed as a tool for research scientists, the generalized and easily used structure of this processor has now led to its acceptance by a large and varied user community. Although Speakeasy¹⁻⁴ has been in existence for many years, it is the widespread availability of time-sharing features such as TSO that accounts for the recent rapid growth in acceptance.

The user communities already include physical scientists, engineers, econometricians, highway departments, hospitals, government agencies and universities. The growth in the capabilities of Speakeasy in recent months is partially in response to the diverse nature of this user group and partially due to contributions from it. Acceleration in this growth rate can be expected as more communities are added.

The Speakeasy processor is currently available for use on IBM 360 or 370 computers operating under VS, OS, TSO, NUROS or MTS. It has been translated and is also usable on FACOM-60 and NEAC-2000 series computers. Translation to several other computer systems are now being carried out. There are currently over 70 installations of Speakeasy and that number is growing rapidly.

Operating under a time-sharing system, Speakeasy plays the role of an extremely powerful desk calculator. It is therefore easy to display its capabilities. Because of the richness of the available resources, each user finds within the system many features specifically tailored to his special discipline. He tends to view Speakeasy as a tool designed for his personal use. The real power of this processor though is its built-in capability for growth by accumulation and adaption. New users to this system, originally attracted by its ease of use are soon adding to its capabilities and are thus helping to provide an ever more powerful system for all users.

Speakeasy is easily used by the novice, easily understood by the initiated and easily expanded by the expert. It provides a means for sensible dissemination of advanced computational techniques and makes their use direct both for the untrained user and for the expert. Growth by accumulation of the ideas of many users has already made Speakeasy one of the most powerful computer languages available, it has at the same time remained one of the most easily learned and easily used methods of communicating with a computer.

This paper is intended to explain, in a general way, the structure of a Speakeasy system. The real power of Speakeasy rests in this basic structure. The currently available system, in spite of its widespread use, is to be viewed only as a demonstration of the capabilities provided by any library-oriented system. Properly viewed,

the present Speakeasy processors are merely operational prototypes of much more powerful general-purpose processors. Discussions about Speakeasy as it exists today may crystallize the development of this type of system to the benefit of all users.

The paper is divided into sections describing the basic components of this system. It is important to emphasize that Speakeasy is an existing and operational system. The demonstratable realization of the concepts being discussed is of particular importance in this development because it is the feedback from users that has led both to the simplicity of notation found in Speakeasy and to the variety of facilities available. A section of this paper demonstrates some of the capabilities available in the currently available TSO version of the system. It shows that a user-oriented system can at the same time be a powerful one.

II. THE OVERALL DESCRIPTION

The structure of the Speakeasy system is shown diagrammatically in Figure 1. The Speakeasy system consists of a language processor designed to interpret the requests of the user and to break them down into basic syntactical components. A scratch-pad storage facility available to this processor enables it to maintain transient information for the duration of the particular application. The language processor and its scratch pad provide the mechanism for a user to define and address structured objects that represent intermediate information in the tasks being assigned to the system. The syntax of the communication language accepted by the processor is a straightforward one modeled on the conventional form of expression in the discipline of interest. The vocabulary of keywords directly associated with the processor is minimal, representing only those basic features of universal applicability.

Libraries of information are attached to this processor to provide the system with its working vocabulary. When a specific operator is requested of the system, it searches the libraries attached to the system at that time. If the requested operator is found, then it is dynamically attached to the system so that it may perform its function. In most applications such operations involve operands previously defined and saved in the transient scratch-pad area. Results are saved in the scratch-pad area and are therefore available for later use.

This operator-operand formulation with operators on structured objects is the heart of the Speakeasy system. The ability to define structured objects such as arrays, vectors and matrices makes it meaningful to develop a large set of operators tailored to specific data structures. The use of attached libraries make it possible to provide an ever increasing set of such operators without

creating a system of extreme size and complexity. At the same time the use of libraries provides the mechanism for tailoring systems to specific applications without restricting the overall development.

The extreme modularity of this system, the extensive use of libraries and the simplicity of its control language combine to provide a system with extreme power, ease of adaptability and the capability for sustained growth.

III. DESIGN OF THE LANGUAGE

The Speakeasy system is a repository for ideas. The communication language for the system (also called Speakeasy) is intended to make this repository usable. In designing the language and its syntactical rules the foremost considerations were naturalness, ease of use and tolerance for trivial mistakes. The basic specification for the language is a simple one. If a request by a user is unambiguous and looks correct to him then it should be accepted by the system. (For the most part this specification has been realized in the current processors.)

This is somewhat different in philosophy from normal computer languages that are designed for programming a calculation for computer processing. Such languages are, to a large extent, tailored to the existing hardware facilities. The specifications of many conventional languages are designed so that a large number of decisions that relate only indirectly to the calculation must be made by the programmer. To give just a few examples, fixed point versus floating point numbers, dimension statements for quantities that occur only as intermediate results, and input and output formats—none of these are directly related to the statement of the problem itself. Often such relatively trivial specifications constitute the bulk of a program—the parts of the program related to the calculation are but a small part of the material written by the user.

In Speakeasy most of such decisions are considered to be part of the internal functions of the system. The user is supposed to formulate his problem in as brief and as natural a manner as possible, the system is supposed to translate this formulation into an executable form, relieving the user of as many decisions as possible. Actually the Speakeasy language specifications bear little relationship to the structure of a computer and only very indirectly reflect the form of calculation as executed in the computer.

The Speakeasy language is designed to operate with structured objects. Arrays of numbers, mathematical vectors and true matrices are among the many classes of structured objects that a user may define and use in his calculations. Implicit algebraic rules that are class dependent provide a

variety of tools for formulating calculations while operating on defined objects as single entities.

The availability of structured objects means that an operator-operand form of language with great richness can be developed (one of the limitations of a normal language is that the scalar structure of the language itself limits the availability of built-in functions to those that operate on scalars). Each of the operators in Speakeasy is designed as a self-contained module whose operations are dependent upon the structure of its operands. In this way all of the decisions normally necessary in invoking routines in conventional programming techniques are internally contained in the operators of the system. Each operator is clearly definable in terms of what it does and contains many checks to see that it is being properly applied. By placing decision processes such as these within the system the user is relieved of most of the mundane parts of programming associated with languages such as Fortran or PL/I. The user can concentrate his effort on formulating the overall logic of his computation and be assured that a vast number of decisions are being properly made.

Tolerance is somewhat foreign to computer languages. A misplaced comma in a format statement in most programming languages results in a complete abort of the calculation even though the intent of the user was obvious. In Speakeasy the primary intent is to provide the user with the answers to his problem, not to force him to specify his problem within rigid language rules. The structure of the language itself makes errors unlikely and the tolerant analysis aids even more by ignoring unimportant syntactical errors.

As compensation for the tolerant language a large amount of internal consistency checking is carried out for each operation. The presence of structured objects in the language means that a large amount of information is available to the processor and as a result a large number of tests can be carried out automatically. Checks are thorough and prevent erroneously formulated problems from being carried out. Such internal checking combined with the tolerant language syntax provide the means for quickly and easily formulating a problem for processing with the assurance that the overall logic is correct. Actually the checking process is so detailed that completion of a problem without a detected error is real assurance of correct logic specification, a feature missing in most other languages. The overall language designed around an operator-operand concept combined with conventional algebraic tools is powerful, easily learned and easily used. It is logically complete and is extensible in both the types of structure supported, the algebraic rules for new structures, and the operators available for operating on objects.

IV. THE TRANSIENT SCRATCH-PAD— NAMED STORAGE

A special dynamic storage scheme is an important component of the Speakeasy system. This storage facility provides the mechanism for defining and retrieving the structure objects discussed previously. In this scheme each defined object has associated with it a complete description of its structure along with particular values for its elements. Most importantly, the object and its descriptive information can be referenced by name. The name is all that is needed to locate any defined object, to determine everything about its structure and to use it in a calculation.

Named Storage⁵ was developed in 1964 for use with calculations in nuclear physics.^{1, 7} It is this storage technique that led to the development of Speakeasy. Individual modules of the Speakeasy system (the operators available to the language processor) are designed to operate on objects defined in Named Storage and to produce new objects there. Such modules (called Named Storage manipulators) are logically self-contained and complete. Each manipulator has available to it all structural information of the operands and is designed to perform a specific operation on structured objects. All decision processes, all tests on capability, etc. are made within the manipulator and the resultants are defined in Named Storage. The extreme modularity of the Speakeasy system is a consequence of this rather straightforward storage scheme.

Each defined object in Named Storage has descriptive information appended to it which designates its class, the type of data in its elements and dimensionality. The allowed ranges for these designators has been made large so that new data types, new classes, etc. can be supported in later developments. Since no computational capabilities are contained in the storage package it is possible to extend the capabilities of Speakeasy entirely through additions to the manipulator attached to the system.

V. LIBRARIES

In most computer systems each new addition brings with it increases in complexity, more overhead and a larger processor. Even with a very modular system this remains the case. The benefit of each addition must be weighed against the consequence that it will have on the overall system performance. It is therefore unlikely that a feature of great benefit to a few users will ever be adopted if it degrades the system's use for others.

The library orientation of the Speakeasy system solves this problem in a particularly interesting way. The use of attached libraries

containing operators such as those described earlier provides a system with easily expandable capability in a way that neither constrains the growth nor increases the basic structure of the system. Operators that are dynamically attached to the processor, called linkules,⁶ provide the means for adding any desired capability for those who need it, without others even being aware of its existence. Private libraries of linkules can be used to give each user community a processor tailored to its own needs, while at the same time providing them all with a common base of important features.

The operator-operand formalism is a general one; there are no constraints therefore on the number or type of operators that can be in the libraries of the system. Operators are added to provide new capabilities, to provide alternatives to existing facilities or to combine together the functions of several existing operators. The growth in the power of this system is then almost automatic since new capabilities are continuously being provided by those users that find particular features desirable but not yet in the system.

Each new operator added to the system brings with it capabilities that are enhanced by the presence of other operators in the system. A system such as Speakeasy thus reaches a critical stage where the interrelationship between the operators begins to make itself felt. After this threshold is reached the growth of the capabilities of the system are often based on finding new ways to interconnect existing facilities. Speakeasy has passed that threshold and it is no longer possible to clearly define the limits of the capabilities of the existing system. It should be stressed that this added capability could occur in any very large, modular system but that without the library capabilities being discussed, it is unlikely that such a system could be created.

To illustrate this feature more concretely might be of value. Several of the linkules of the current TSO system are designed to provide user interfaces to the existing data management facilities provided by the operating system. These interfaces, originally designed to provide the means of saving information between Speakeasy runs, were found to provide extra capabilities when combined with a linkule designed as the HELP processor for Speakeasy. Rather than being forced to construct new capabilities for indexing these documents, it was possible to make use of existing facilities in ways that were not originally planned. A simple HELP processor embedded in the existing system became one of great power in a very unexpected way.

Similarly, it was discovered during the design of a tutorial processor that all of the capabilities already existed in this system and that by merely properly combining these facilities, a very

powerful teaching machine could be produced.

In addition to the linkule libraries attached to Speakeasy provisions are made to directly attach document libraries and libraries of procedures written in Speakeasy. These libraries provide the user with the information necessary to use the system and its associated operators.

Access methods to other libraries can, of course, be provided in the attached linkule library. It is therefore in principle possible to address any information stored in the computer complex. From the users' viewpoint this is the ultimate aim of the Speakeasy development.

The freedom provided by a library oriented system is obvious. Growth by the accumulation of new information is automatic. Maintenance is particularly easy since replacement, modification and additions are made on members of libraries and not on the processor itself.

In the Speakeasy system there have been two distinct evolutionary developments. The basic processor has been evolving slowly in the direction of greater simplicity and lessened capability. The libraries have on the other hand grown rapidly and are providing ever increasing capabilities.

In recent weeks a statistical package has been interfaced to Speakeasy, the entire EISPAK⁹ system is now provided as part of the method for dealing with eigenvalues and eigenvectors of matrices and new graphics capabilities have been provided to TSO users with appropriate terminals.

The primary problem of growth in this system is now one of validation and acceptance procedures. For a large system with many users this is a satisfying difficulty to face.

VI. THE USER INTERFACE

The user's first introduction to a system such as Speakeasy can be through a variety of devices. The most natural and most satisfying is in an interactive system with a time-sharing terminal. The variety of terminals that are available and the features available to particular devices makes it important that a system such as Speakeasy functions equally well for each type of device and that it be able to exploit the peculiar capabilities of each. In this system this is accomplished by isolating all input and output to specific components of the system, designing interfaces to classes of devices, and providing other general facilities that can be selectively made available to users of specific devices on demand.

This ability to interface correctly to every type of terminal is of great psychological importance since the intent is to make this system

appear natural to the casual user. If the introductory session with the processor is spent describing how a particular device is used or why a multitude of peculiar keys are used then it is unlikely that you will ever convince the user of the naturalness of the language.

The currently operational versions of Speakeasy can be used with card readers, printers, ASCII terminals, IBM 2741's, Tektronix 4000 series graphics terminals, IBM 2250's or any combinations of these devices. In each case the adaption to the device is one in which the device is natural to use and one where all of its hardware capabilities can be exploited.

VII. DOCUMENTATION

Any computer system must be properly documented in order that it be usable. A large amount of user oriented documentation is already available for this system.³ Documentation for a major system is not an easy task. If approached solely by conventional means it would be even more difficult in a system like Speakeasy that is designed for growth. Any printed documentation would be outdated before it was published. Fortunately, the system itself is capable of providing facilities that not only supply the documentation but do so in a way that is guaranteed to be self-sustained.

A library of documents referred to as the Speakeasy HELP documents⁸ is attached to the system. This library is addressed by a linkule in the normal library. Each word, concept or facility available in the system is described in a member of the HELP document library. The library is designed as a tree structure so that a user probing the library in an interactive session is led quickly to the specific concept of interest.

A new linkule is accepted into the system only if a corresponding HELP document is provided. Since this is the only mechanism provided to advertise the presence of the new operation it is almost automatic that new facilities are documented even before they become operational within the Speakeasy system. To gain some insight into the size of this library the currently distributed version of Speakeasy has about 350 documents. Facilities are provided to attach private HELP libraries, and to obtain printed copies of all documents in publishable format for general distribution.

Additional documentation, more detailed than those described above, are available in other libraries that can be attached to the system. In addition a large amount of conventional printed documents are available.

VIII. THE AVAILABLE VERSION

Although the primary intent of this paper is to discuss the structure of an idealized Speakeasy

system, the existence of an operational version of a prototype system is not to be ignored. Many theoretical systems, though advanced in conceptual structure, represent systems that cannot be realistically evaluated since implementation is not necessarily an end product of the study. By contrast Speakeasy has been an operational system for years and has already evolved under pressure from users and from the impact of new concepts. This evolutionary growth has led to a processor that is greatly simplified as compared to the original one and at the same time to one with much more power. In part, this has been possible because the original implementation was carried out with the knowledge that it would be the vehicle for the testing of new concepts and that it should therefore be capable of being greatly modified and augmented. Its structure was therefore designed specifically with this in mind and a very modular, expandable programming technique was used. This evolved into a completely dynamic structure such as the one described earlier that represents the form of all of the additions that have been made in recent years.

The current implementation though partially inhibited by its historical structure still contains all of the capabilities of the idealized system. Growth in new directions has been dramatic as new types of user communities have discovered the processor and explored its built-in growth capabilities.

IX. EXAMPLES OF SPEAKEASY

This section is designed to show some of the capabilities of Speakeasy as currently available under IBM-TSO. It should be explained once again that Speakeasy is a generalized interface to libraries of information. The features demonstrated here are therefore only intended to show some of the variety currently available. It is likely that by the time of publication of this article there will be entirely new facilities added to satisfy new needs (e.g., a basic statistical package has just been installed in our library and access methods for distributed computing in a network are being designed). The examples chosen mostly illustrate capabilities that are unique to Speakeasy or ones that are particularly interesting because of the simplicity of

notation. Readers interested in more detailed descriptions are referred to other publications.⁴

Figures 2 through 9 are an unedited (except for pagination and comments) session at a TSO terminal. It was run in a 140K region under release 21.6 in an unmodified TSO environment. The characters `_` are the normal manual mode prompt characters. Other prompts are supplied for different modes of operation.

REFERENCES

1. S. Cohen, "The DELPHI-SPEAKEASY System," *Computer Phys. Commun.* 2(1), 1-10 (January 1971).
2. S. Cohen, SPEAKEASY: "RAKUGO," First USA-Japan Computer Conference Proceedings, Tokyo, 1972.
3. S. Cohen, "The SPEAKEASY-3 Reference Manual," Argonne National Laboratory Report ANL-8000 (May 1973).
4. S. Cohen, "SPEAKEASY," *SIGPLAN Notices* 8, 7 (July 1973), 31-44.
5. S. Cohen, "Named Storage: A Dynamic Storage-Allocation Scheme with Manipulative Routines," Argonne National Laboratory Report ANL-7021 (April 1964); "Named Storage/360" (to be published).
6. S. Cohen, R. N. Kimmel, K. Rich and F. J. D. Serduke, "SPEAKEASY-3: Linkules and Interfaces," Argonne Physics Division Information Report PIHY-1972C.
7. S. Cohen, R. D. Lawson and M. H. Macfarlane, "Automated Nuclear Shell Model Calculations," *Methods in Computational Physics*, Vol. 6, Academic Press, New York (1966).
8. J. K. Fink, "The SPEAKEASY HELP Documents," Argonne Physics Division Informal Report PIHY-1972G (July 1972).
9. B. T. Smith et al., "Matrix Eigensystem Routines—EISPACK Guide," *Lecture Notes in Computer Science*, Vol. 6, Springer-Verlag, Berlin (1974).

Fig. 1

The structure of the SPEAKEASY system. Three basic blocks represent its major components: 1) the user interface; 2) the processor; and 3) the attached libraries.

```

READY
  @unpack
  TSO SPYFAEASY 5F IOTA 11:25 A: 12/13/75
  _a4=17/8-6
  _a4=17/8-6 = 87.5
  _acosf-13
  ACOSf-13 = 3.1416
  _significance 30
  _answer
  ANSWER = 3.1415926535898
  _significance 5
  _signanswer
  _sign=3+7
  _a6
  _a6 = 19
  _answer=5
  ANSWER=5 = 95
  _answer/5
  ANSWER/5 = 19
  _51
  51 = 120
  _SQRT(3)
  SQRT(3) = 1.7321
  _SQRT(-3)
  SQRT(-3) = SQRT(-3) = ##### COMPLEX NUMBER
  _domain complex
  _sqrt(-3)
  SQRT(-3) = 3.7521i
  _{(3-3i)/(6-3i)}
  {(3-3i)/(6-3i)} = 3-30i
  _sqrt(answer)
  SQRT(ANSWER) = 6.5808-6.7046i
  _sin(30)
  SIN(30) = -.49805
  _answer in degrees
  _sin(30)
  SIN(10) = .15
  _rationalize
  _sin(30)
  SIN(10) = 1/2
  1/3+5/2-3/5
  1/3+5/2-3/5 = 47/105
  _rational
  _answer
  ANSWER = .44762
  _name
  CURRENTLY DEFINED NAMES
  PC, ANSWER, I

```

```

i:=0;
Z (A 3 BY 3 MATRIX)
1 2 3
5 2 7
3 5 8

i:=1;
Z (A 3 BY 3 MATRIX)
20 9 17
36 21 19
0 0 20

i:=2;
1/A (A 3 BY 3 MATRIX)
-.32079 .00975 .25
-.65625 -.20125 .25
-.03225 -.15625 -.25

i:=rationalize;
i:=simplify;
ANSWER (A 3 BY 3 MATRIX)
-7/32 3/32 1/6
21/32 -9/32 1/6
-1/32 5/32 -1/6

i:=rationalize; simplify;
Z=ANSWER (A 3 BY 3 MATRIX)
1 -5.5511E-17 1.9078E-17
0 0 0
2.7758E-17 -.16175E-17 1

i:=simplify; (10=10)
i:=simplify;
ANSWER (A 3 BY 3 MATRIX)
1 0 0
0 1 0
0 0 1

i:=evaluates(x)
Z=evaluates(x) (A VECTOR WITH 3 COMPONENTS)
1.0000 -2.0204E-.66667 -2.0204E-.66667

i:=evaluates(x)
Z=evaluates(x) (A 3 BY 5 MATRIX)
-.00294 -.32665-.71360 -.32665-.71360
-.87710 2.0065-.65666 2.0065-.65666
-.32225 -.40636-1.0061 -.40636-1.0061

CET(x) = 32

i:=transpose(x)
Z=TRANSPOSE(x) (A 3 BY 3 MATRIX)
2 7 6
7 6 8
5 8 0

```



```

x_where(a[32] == y-x
y_where(a[10,12] == y
i_cubulate(x y z
  N V Z
  0 8 -8
  .2 0.04 -0.04
  .4 0.16 -0.16
  .6 0.36 -0.36
  .8 0.64 -0.64
  1 9 -9
  1.2 9.44 -9.44
  1.4 9.96 -9.96
  1.6 10.56 -10.56
  1.8 11.24 -11.24
  2 12 -12
  2.2 12.84 10.64
  2.4 13.76 11.36
1.sum(x)
SUM(n) = 65
1.overage(x)
AVPAG(EV) = 16.5
1.sum(y)/nops(y)
SUM(y)/NOELS(y) = 16.5

```

Vocabulary

CLEAR	ELEMENT	INTEG	MAXFPOCL	PI-INCLTA	SHIFTUP	VARIABLE
CLEARDAY	ELINCOLS	INTERCEPS	MINFPOCL	PLINCLTA	SIGN	VARIANCE
COPACON	ELINELS	INTERCAL	FEAR	PLINCLF	SIMIFIC	VECTOR
CORRELAY	ELINELT	INTERP	FEAR	PFDOCLUS	SIGN	VECTOR
CORMAT	ELINRWDS	INTERP	FEAR	PFDOORUS	SIN	VERSION
CORMAX	ELLIPSE	INTERFPL	MIN	PFDOUCTS	SINGLEVA	VECTORS
CORMIN	ELLIPK	JMPART	MINOFCL	PROGMA	SINX	VERSIONS
COSWIDTH	END	INTPPCUD	MINOFPM	PROGPMAS	SIZE	VFM
COMMANDS	ENPAUTOP	INTS	MINOFPM	PROGRAMS	SLAY	VLADEL
COMPILE	ENDLAMP	INTS	MISCELLA	PUNCH	SORT	VOCABULA
COMPLEX	EOHE	INTS	MISCELLA	QUICK	VECLA	VECLA
CON	EQ	INVERSE	MODULO	RANDUM	SPECIAL	VIS
CONJUGAT	ERF	KEEP	MOVET	RANKED	SPMBS	MIATIS
CONSTRAT	ERFC	KEEPTSU	MOVE	RANKED	SPMBWS	MIERE
CONTINUE	ERKORS	KEPT	MYCDS	RATIONAL	STOT	WHEREVER
CONVERT	EXECUTE	KEEPTSU	HYTHPL	READ	STANDREV	WHOLE
COPY	EXP	LABEL	ITKPEE	REAL	STATISTI	WHITE
CORRELAT	FIN	LEVEL	REALPART	REAL	ZEPOS	ZEPOS
CORR	FINRWMD	LENGTH	HYTHKLS	REAL	STRUCTUR	
COSH	FLIP	LINKINDEX	MYHPCOS	REALS	SUI	
COT	FOR	LIBRARYES	NAMEG	RECLASS	SUBCOLS	

Fig. 6. The available vocabulary is large and growing. The user may display the available vocabulary at this terminal.

```

_x=grid(0,2*pi,pi/20) ; y=sin(x) ; z=cos(x)
_vsize=50 ; vscale=-1,3 ; lineplot( y z )

```

Fig. 5. Plotting of results even at nongraphics terminals is also valuable for many applications.

```

> To obtain the words 'in a given subclass SC, enter
HELP SC
< help statistics
STATISTICS returns the words related to statistical analysis.
AVERAGE returns the average value of the elements of an object.
CORRELATION gives the correlation coefficient between 2 data sets.
MEAN returns the mean of the elements of an object.
RANDUM generates random numbers.
STDEV returns the standard deviation.
VARIANCE returns the variance.
<
> To obtain a description of a given word KKE, enter
HELP KKE
< help correlation
CORRELATION(X,Y) returns the correlation coefficient of X and Y.
The correlation coefficient is a measure of covariance or association
between two sets of data. X and Y must be arrays of equal length.
The correlation coefficient between X and Y if X and Y are both
arrays of length N, is given by the fraction NUM/DEN where
NUM= SUM( (X(I)-AVERAGE(X)) * (Y(I)-AVERAGE(Y))) and
DEN= SQRT( (SUM( (X(I)-AVERAGE(X))^2)) * (SUM( (Y(I)-AVERAGE(Y))^2)) )
and each SUM is taken from I=1 to I=N.
I:= 1 2 3 5 7 9 11
J:= 2 2 4 6 8 10
correlation(a,b)
CORRELATION(a,b) = .18063

```

Fig. 7. Each word in the vocabulary refers to a short description document. These documents are arranged as a tree structure to enable the user to find a particular word quickly.

```

1_tutorial
INDEX PAGE = 0
CONTENTS
Index to the Speakeasy tutorial sessions 12/12/79
SESSION SUBJECT
Start An introduction to Speakeasy
Arrays Array definitions and operations
Matrix Matrix definitions and operations
Vector Vector definitions and operations including
vector-matrix operations
Logic Use of logical and relational operators
Edit Use of the editor
Kemp Saving information between runs
Tek Tektronix graphing
Misc Miscellaneous information
Type TUTORIAL XXX to begin the tutorial session called XXX.
Type TUTORIAL XXX N to display page N of the session, XXX.
(TUTORIAL XXX 0 will give you a table of contents for that session.)
Type MORE to continue a session.
To obtain an index of keywords used in all the tutorial sessions
type in the command MORE at this time.
1_tutorial start
START PAGE = 0
START
Table of Contents
1 How to use the Tutorials
2 Introduction
3 Arithmetic Operators
4 Answer
5 Representation of Real Numbers
6 Representation of Complex Numbers
7 Significance
8 Defining a Variable
9 built-in Functions
10 Conclusion
1_more
START PAGE = 1
HOW TO USE THE TUTORIALS
This tutorial is a series of short paragraphs written to help
you learn about Speakeasy. It is written for use with the
TSO-Speakeasy processor. To use this tutorial, read the information
printed, and try to carry out some examples of the topic being
discussed. When you feel that you understand the subject, enter
the command MORE. This will request the next part of the tutorial
to be displayed. If you wish to restart the tutorial at some later
time, at a particular place, type in the statement TUTORIAL XXX N
where XXX is the heading of the section, and N is the page number.
Remember that typing SPEAKEZ will allow you to use the
TSO-Speakeasy processor and typing QUI? will return you to TSO.
Now, enter the command MORE.

```

Fig. 8. A large built-in tutorial is also part of the system. This shows how the tutorial is structured.

```

1_more
START PAGE = 2
INTRODUCTION
Speakeasy is an interpretive processor designed to provide
access to information stored in a computer. The ability to do
numerical calculations is one of the more basic yet one of the
most powerful capabilities of Speakeasy. In this mode of operation
Speakeasy can be viewed as a super desk calculator. In the
sections immediately following, some of the capabilities will
be introduced to enable you to use Speakeasy as a desk calculator
effectively.
1_more
START PAGE = 3
ARITHMETIC OPERATORS
Arithmetic operators in Speakeasy resemble arithmetic
operators of standard computer languages. Operators and their
meanings are:
* raise to a power
*/ multiply, divide
+ add, subtract.
In evaluating an expression without parentheses, Speakeasy carries
out operations in the following order:
1) all powers
2) all multiplications and divisions
3) all additions and subtractions.
In each case, evaluation occurs from left to right. Thus 3/2*4
means (3/2)*4, and not 3/(2*4). Of course, you may freely use
parentheses to be certain of the order of evaluation. If there
is any ambiguity in your mind, parentheses should be used.
Try writing simple arithmetic expressions such as 3*3
and then proceed to expressions such as (3*4)*(7-5)/8.
1_more
START PAGE = 4
ANSWER
Each of the calculations so far has been independent. The
result of a calculation such as those you have performed
is called ANSWER. You may use ANSWER in the next statement to
stand for this result. Thus,
3*4 (your input)
3*4*7 (Speakeasy response)
1_ANSWER*3
ANSWER*5*10
1_ANSWER*ANSWER**3
ANSWER*ANSWER*5*10*10
Try a few of these until you can carry out evaluations of long
expressions without difficulty.

```

Fig. 9. This is a continuation of the tutorial session started in Fig. 8. It indicates the informal structure of the sessions.