AUTOMATED ANALYSIS OF LANGUAGE STYLE AND STRUCTURE

1969 - 1970

Report on research for the period
March 1, 1969 - August 31, 1970

The Research reported herein was conducted under Contract N00014-67-A-0321-001, Office of Naval Research, U. S. Navy, Task No. NR348-005

at the
University of North Carolina
at Chapel Hill

The research reported herein was conducted under Contract N00014-67-A-0321-001, Office of Naval Research, U. S. Navy, Task No. NR348-005

at the

University of North Carolina
at Chapel Hill

AUTOMATED ANALYSIS OF LANGUAGE STYLE AND STRUCTURE

1969-1970

Report on research for the period
March 1, 1969 - August 31, 1970

Sally Y. Sedelow, Principal Investigator
Martin Dillon, Consultant
Gerald Fisher, Consultant
Walter Sedelow, Consultant
Walter Smith, Consultant
H. William Buttelmann
John B. Smith
David Wagner

AUTOMATED ANALYSIS OF LANGUAGE

STYLE AND STRUCTURE

1969 - 1970

Report on Research for the Period
March 1, 1969 - August 31, 1970

Sally Yeates Sedelow, Principal Investigator

ABSTRACT

This report summarizes work directed toward comprehensive computer-based characterization of natural language text. A general-purpose thesaurus has been keypunched and proofread, and corrections are now being made; this work is preliminary to an effort to determine the structure of the thesaurus and, in general, to construct parametric models of thesauri. Preliminary approaches to the design of a statistical package for natural language analysis are also described; these approaches include a program to provide data formats for any type of language analysis. A User's Manual for the PL/I, S/360 list-structure and ring-structure VIA packages is also included in this report.

PREFACE

I would like to thank the staff of the Information Systems Branch, Office of Naval Research, for their unfailing helpfulness on matters connected with this contract. Thanks are also due to the Departments of English and Computer and Information Science for a reduced teaching load, permitting work on this research, and to the Office of Research Administration for assistance with administrative detail. I am also grateful to the secretaries in the Department of Computer and Information Science for getting the report into reproducible form and overseeing its circulation.

TABLE OF CONTENTS

I. Survey of the Automated Analysis of Language Project, March 1, 1969 - August 31, 1970

A. Introduction

The Automated Analysis Language project is directed toward developing a system of programming packages for the analysis of natural language. The broad goal of the project is to use the computer to characterize comprehensively a non pre-edited natural language text. Comprehensive characterization would imply that the style of any text so analyzed could be generated by the computer in accordance with the parameters for that text. Such complete characterization would permit identification of individual authors or speeches as well as permit the description of the content of their remarks. In addition, characterization of written or spoken texts from a specific culture, or subculture, would permit generalizations about language usage of a given group of people. Such generalizations should have real impact upon, for example, the verbal structuring of diplomatic documents or upon the interpretation for intelligence purposes of verbal materials. Although linguists have explored a few aspects (e.g., word lengths, sounds utilized) of similarities and differences among languages, there has been no major effort to look comprehensively at such patterns and then explore their implications for verbal interaction among nations and groups of nations. The importance of sophistication about language in a world where much action either consists of

language or is precipitated by language cannot be overemphasized. With the advent of the computer--especially interactive computing--the possibility of both extended study of language and of fast analysis of given sections of written or spoken text is more immediate.

It is the long-term aim of this project to develop analytical procedures which can themselves interact to perceive both the definitive and idiosyncratic elements in a given text. That is, given the results of one analytical procedure, automated decision-making algorithms would indicate which procedure should be next executed. To effect any such interdependence upon a large scale, much research is needed upon the properties and behavior of language, as well as upon the statistics needed to describe those properties and behavioral aspects. This project has been attempting to develop computer-based procedures which will facilitate such research; the procedures already developed are also being applied to current information analysis needs.

B. Summary of Past Year's Work

For clarity, the research of the past year should be set in the context of the earlier work connected with this project. The initial emphasis was upon developing a package of programs which would perform what a political scientist would call content

analysis,[1] what a literary scholar might call thematic analysis, or what an intelligence officer might term information analysis. This program package, called VIA, was first programmed in FORTRAN and TAC, the assembly language, for the Philco 2000, and has subsequently been reprogrammed in PL/I for IBM's System 360. A major addition to the PL/I VIA is a version that uses rings rather than the lists used in the FORTRAN version and an alternate PL/I version. A ring-structure version seemed desirable for several reasons: 1. Linked lists may give a sense of hierarchy when none is intended; 2. Because the ring does not imply a hierarchy, it would seem sensible to key a search on all elements in a ring; thus, a concept which might be lost in the list-structure version because the word heading the list does not occur with frequency sufficient to exceed some threshold might well be retained in the ring-structure version because the frequency of all the words in the ring does exceed the threshold.[2]

In order to link together the semantically-related words revelatory of significant content or themes, the VIA programs require inputs of semantically-related words. Until the past year, these inputs had to be compiled by the person using the programs;

---

[1] For a discussion of the programs from this point of view, see the chapter by S. Y. Sedelow and W. A. Sedelow in The Analysis of Communication Content, ed. George Gerbner, et al, John Wiley and Sons, Inc., 1969.

[2] The design of the ring-structure VIA is described in detail by William Buttelmann in the reports of 1967-68 and 1968-69 for this project. The PL/I list-structure version is outlined in the report for 1968-69.

that is, given computer output showing all words grouped together by root and the frequency of occurrence of all words in that group, the human being chose certain of those groups as worthy of further investigation and hand-compiled the lists which keyed further searches and linkages. Although such hand-compilation is not onerous for the scholar primarily interested in a few texts, it is quite impractical for the information analyst who must constantly deal with new texts. The desire to automate the compilation of semantically-related words led to research on several general-purpose thesauri.[3] The result of that research was a decision to keypunch Roget's International Thesaurus so that it might be studied further with a view toward modifying it to serve as the dictionary for the development of lists of semantically-related words. This past year, a research assistant proof read the keypunched version for errors and at the present the list of ordered pairs (category - word) generated from the thesaurus is being corrected.

Because it is difficult to test extensively the ring-structure VIA without an available thesaurus in computer-accessible form, work on it has been in abeyance while the Thesaurus has been proofread. A recent test run using a portion of the Thesaurus indicates that some minor modifications to the program have somewhat reduced the enormous quantity of output produced by the first runs of the ring-structure VIA; at present,

---

[3] For a description of this research see Gerbner, op. cit., and this project's research reports for 1966-67 and 1967-68.

it is impossible to provide a direct comparison because the first runs did not use the computer-accessible version of Roget's International Thesaurus and we anticipate that the use of the Thesaurus will increase the size of the output relative to that produced with other input. Nonetheless, it seems clear that further pruning algorithms must be devised, possibly in connection with an interactive version of the program. Preliminary tests of the list-structure VIA using Roget's International Thesaurus would suggest that the output will be sufficiently great to be informative yet not so great as to be difficult to comprehend. Although these statistics are subject to revision, it is at least suggestive to note that an earlier run using the word ACID to key the search produced an attached list with two words and the test run using Roget's produced an attached list of sixteen words. Because none of the words on the attached list also served as a search key no additional linked lists were produced in either case.

The effort to describe the structure of Roget's International Thesaurus, itself, has also awaited the correction of the key-punched version. We have, however, discussed approaches to the task which are described by David Wagner later in this report. Briefly, we are concerned with the degree of connectivity, with "growth rate" (e.g., what is the rate of expansion in terms of tokens, as well as types, from any given word), with bias as implicit in the words in the Thesaurus (as compared with other word lists) as well as implicit in the words functioning as nodes in the thesaurus. Optimum methods of providing an index for

such a thesaurus are also of interest.

Much of our energy this past year has been devoted to considering the design of a statistical package for natural language analysis; this consideration, in turn, prompted us to explore the design of data formats which would be suitable for any form of language analysis. This effort was collaborative, involving personnel associated with a NASA project for which W. A. Sedelow was Principal Investigator, as well as involving colleagues and graduate students in the Departments of English and Computer and Information Science. Some of the questions raised and conclusions reached are described in a sequence of papers later in this report by Martin Dillon, Gerald Fisher, and John Smith. Professor Dillon discusses the desirability of a taxonomy which would provide a way of classifying the diverse data bases and quantitative approaches which have been used in the analysis of textual material. This discussion is based upon a sizable inventory of verbal measures compiled as part of the NASA project. Professor Fisher describes the data formats which were devised to serve as the basis for any natural language analysis. A first implementation of these formats in PL/I was undertaken as an MA Thesis in the Department of Computer and Information Science by Joe Ragland. A brief description and a listing of his program are included as part of this report. A proposed design of a statistical analyzer is described by John Smith. As envisioned, the analyzer would assume data to be in the formats described in Professor Fisher's article and run its programs off that data.

To the best of our knowledge, there has been no effort comparable to our attempt to attack comprehensively the problems associated with the development of a statistical package for natural language analysis. It is clear that we have made only a beginning, yet the manhour input entailed by that beginning has been impressive. It is our hope that other researchers in this field will read the papers by Dillon, Fisher, and Smith, and provide feedback which will enable us to improve the design and eventual implementation of these programs.

Because the Principal Investigator for this project is moving to the University of Kansas, a final effort of the past year was to leave a PL/I S/360 version of VIA resident at the University of North Carolina so as to be accessible to other researchers. Arrangements have been made with the Institute for Research in Social Science at the University of North Carolina to run the 1969-70 version of VIA for those who request that service. Although it partially overlaps sections of earlier reports, it seemed desirable to include in this report a complete User's Manual for the 1969-70 version of VIA. The User's Manual appears on pp. 85.

C. Plans for Future

The move to the University of Kansas will provide the opportunity to design appropriate sections of the language analysis programs for a time-sharing environment. Within the

VIA package, the ring-structure VIA is a prime prospect for interactive computing; because William Buttelmann, the designer and implementer of the ring-structure VIA, will also be on the Computer Science faculty at the University of Kansas, the prospects for pursuing this particular project are very good. In addition to the ring-structure VIA, other programs would seem especially suited to an interactive mode. For example, MAPTEXT, a graphic display program which will in time be related to the statistical analysis package as well as to VIA, would lend itself to interactive computing. Uninteresting displays could be aborted and those that looked promising expanded.

VIA support programs will be implemented and further work will be devoted to the design and implementation of data formats for a statistical analysis package. The package, itself, will then be the next order of business.

David Wagner will continue work on models of thesauri as part of a doctoral dissertation. In addition to Roget's International Thesaurus, other computer-accessible reference works will be assembled with a view toward comparative studies and, ultimately, mutually advantageous modifications.

The association of the Principal Investigator with a department of Linguistics, as well as with Computer Science, will facilitate further research on problems associated with rigidities of usage, both syntactic and semantic, which will have implications for much language analysis. For example, Gerald Salton has discovered that within certain subclasses of documents, syntactic

parsing did not significantly improve relevant document retrieval--presumably because of rigid, patterned usage of potentially ambiguous terms. Our own analysis of translation of Soviet Military Strategy has produced similar results.

1. Thesaurus Research

by

David Wagner

We have barely begun the long range research thrust to develop an adequate model of our machine-accessible thesaurus and to explore possible methodologies which would yield parametric models of thesauri and measures of similarity and dissimilarity between these models. In this section, I will indicate what has been done to date. To aid in the clarity of this exposition I will shift my metaphor to that of graph theory and use the words nodes, labels and arcs. A node will capture the essence of a thesaural category, a label associated with a node will represent the list of all phrases which are members of that category, and an undirected arc connecting two nodes, say n1 and n2, will signify a non-empty intersection between categories whose nodes are n1 and n2. With this paradigm certain questions have rather immediate answers, at least conceptually. One will certainly be led to ask if the thesaurus is totally connected or, if not, what the structure is of the several subclusters or subthesauri out of which the thesaurus is constructed. From the graph theory one would

conceptually construct a logical transition matrix, T, with n rows and n columns where n is the number of thesaural categories, and where a 1 at the intersection of row i and column j signifies that there exists an arc connecting nodes i and j, or, translating back into the thesaural metaphor, that at least one word belongs to both category i and category j. Now since there is a sequence of arcs of length 2 leading from node n1 to n3 just in case there exists some node n2 with arcs connecting n1 and n2 as well as n2 and n3, one can easily conclude that T^2 (where T*T is the matrix logical product) contains a 1 in exactly those entries for which a path of length 2 connects the corresponding row and column nodes. Therefore T^m indicates all pairs of nodes connected by a sequence of arcs of length m. Finally, the graph is totally connected if T^{n-1} contains only 1's ("1" matrix) and since the main diagonal of T contains only 1's, the graph is totally connected only if there is some integer $m<n$ with T^m a "1" matrix. Therefore by computing T^1, $T^2 \cdots T^{n-1}$ one can decide whether the thesaurus is totally connected and if not, by locating certain 1 submatrices, find all totally connected subthesauri.

The system of programs necessary to implement the above conceptual description is operational and has been applied to several test thesauri. As additional information, these programs generate a plot which depicts the rate of expansion for the input thesaurus; or what is the same information, the plot displays the relative number of disconnected node pairs. We expect this plot to resemble a negative exponential with a small time constant ($e^{-t/\tau}$, where τ is small and t is in units of depth or

length of connecting path).

We expect to derive such measures as the expected number of arcs emanating from a node, expected arc length between word pairs, maximum depth, etc. All of these measures can be seen as being incorporated into a parametric model of thesauri.

Although still very much in the realm of exploratory research, an approach to clustering using nonmetric multidimensional scaling may prove to be quite interesting. Essentially this methodology provides a spatial distribution of a set of given words with the spatial distance related to the categorical distance between the given words. This categorical distance, d, may be defined by:

d(p1,p2)=0 if p1=p2

d(p1,p2)=1 if p1, p2 co-occur in a thesaural category

d(p1,p2)=n if there is a sequence of categories

$\bar{p1},\bar{p2},\cdots\bar{pn}$, with p1 a member of $\bar{p1}$

p2 a member of $\bar{pn}$ and

a set of arcs such that $\bar{p1}$ is connected to $\bar{pn}$

and n is the smallest such integer.

Using this pseudo distance measure (ordinal distance or relative distance) defined upon pairs of words, the non metric multi-dimensional scaling methodology locates the set of words in a high order space. From such a scattering of words, one can redefine thesaural categories by a clustering technique which would collect together those words located in a hypersphere of a given radius centered at a given point. By a suitable definition of the origin of the spheres and their radii, one could

effect widely ranging transductions of the original thesaurus, varying from more tightly packed categories to more loosely packed categories. At least for computational purposes, one would like to be able to characterize a thesaurus by a small number of categories; this methodology could effect that transduction.

It is important to recognize that the ordinal distance given above is only one of many conceivable choices. Other choices might involve a normalized cardinality for intersecting expansions (that is, the number of words in common to the set of categories reachable by an arc sequence of a certain length from given word pairs). It is hoped that this thrust will also illuminate the disambiguation problem.

2. Preliminary Design Considerations of a Language Analysis Support Package

by

Martin Dillon

The purpose of these remarks is to provide an overview of a programming system for obtaining the widest possible range of basic measures from natural language text, with capability for assembling the measures into a convenient format for analysis. General design criteria include: the input data, natural language text in machine readable form, minimally encoded, of potentially millions of words; transformation of this material into files suitable for efficient, selective processing; a facility for

maintaining the text files in a form suitable for more advanced analysis; a facility for deriving all required measures from them and maintaining both in a form suitable for more advanced processing.

Background

A list of outstanding problems and questions posed by the analysis of natural language a decade ago would, but for numerous additions, remain unchanged today. Despite a growing number of investigators, more sophisticated use of technology, and an ability to reduce vast amounts of data to manageable proportions, the fundamental principles of natural language, its development and use, have resisted precise formulation. Few would deny that an adequate "explanatory model" has yet to be developed and that progress has been primarily in the discovery of how broadly based are the difficulties, how deeply they run, and how strongly they resist resolution. Perhaps the most outstanding failures fall in the areas of mechanical translation and information retrieval: each has had the encouragement of utility and substantial financial backing; each has drawn on the talents of skilled interdisciplinary teams; and each has failed to achieve a satisfactory solution while substantially contributing to our understanding of the problems posed.

Such failures, and the failure of linguists generally to produce a satisfactory model of language behavior, have led us to reexamine the bases of language analysis and to produce a tool capable of supporting fundamental language research. It should

be noted that the systemization to be discussed lacks, in Thomas Kuhn's sense, the influence of a "paradigm" (The Structure of Scientific Revolutions) drawn from linguistic theory. Of the two primary paradigms in current linguistics, the structural as typified by Zellig Harris in Structural Linguistics, usually referred to as descriptive or taxonomic linguistics, and the algebraic model developed primarily by the transformationalists following Chomsky (initially in Syntactic Structures), the work described here would seem to be more akin in spirit to the former. Despite polemicists on both sides, however, the approaches are complementary but not exhaustive, and it is likely that an acceptable paradigm must be comprised of their synthesis. It is toward this goal that we aim.

Prior to designing this language analysis support package (LASP), a literature search was undertaken to identify as many variables of interest as possible. A basic resource was Language and Language Behavior Abstracts which lists language research by discipline (24, ranging from anthropology to speech pathology) amounting to some 8,000 titles a year. Underrepresented in this journal, and particularly in its early years, are the fields of literary analysis, information storage and retrieval (not listed at all), readability research, content analysis. These were supplemented through other sources. Measures were sought with these properties: of interest to language characterization construed as broadly as possible; basic (as opposed to derived or determined from statistical theory, like factor analysis or

correlations); capable of automatic derivation from machine readable text. An attempt was made to group and order these measures to disclose underlying unities and patterns, both for clarification and to aid in the design effort.

Figure 1 represents a preliminary taxonomy of substantive language areas which was constructed to aid in this effort. While far from satisfactory, this taxonomy was highly useful, indeed essential, in relating diverse research efforts under one frame. (Lack of such global organizational tools is one of the more serious afflictions of language research, and seriously hampers interdisciplinary communication.)

Among the uses to which this frame is being put are: identifying the concerns of different fields engaged in language research; introducing some consistency and coherence into the massive amount of research already performed; and directing attention to areas which have been ignored for whatever reason.

The purpose of presenting Figure 1 in this context is to clarify the design features of the system under development, their origins and the extensions planned. (Complete interpretation of the table requires a consideration of Figure 2 and the approaches listed in Figure 3. The combination of these three tables defines for our purposes what we mean by LASP.) The column headings indicate the goal of the research, interpreted more by the size of the unit under investigation than definitions related to linguistic theory (though used here with approximately their normal meaning). That is, SEMANTICS refers to units of word size or smaller; SYNTAX to units of sentence size or smaller;

DISCOURSE to units beyond the sentence. Contrary to what one may suppose, even these fundamental distinctions must be accepted heuristically, and cannot be maintained strictly. Consider, for example, an investigation dealing with a word-sized unit 'a'. If a text is examined to determine its semantic characteristics (whatever is meant), it would be listed under column I. If, however, the purpose of investigating the behavior of 'a' is to characterize the text, the study would be listed under column three.

Distinctions defined across rows are also broadly heuristic. Their intent, between (1) and (2), is to distinguish studies by the amount of a priori knowledge brought to bear in the analysis. In the rows under (1) would appear studies depending on minimal knowledge, deriving categories or structure from basic statistics within the text itself; under (2) would be listed those studies which assume defined categories or structure as a basis of the analysis. Typical examples of a priori categories are: SEMANTICS, synonymous classes of content words in retrieval studies; SYNTAX, words classed by parts of speech; and DISCOURSE, the content categories of general inquirer studies. Structural assumptions come into play in the use of thesauri in SEMANTICS and parsers in SYNTAX.

Eventually, an annotated bibliography will be produced with a similar organization. For the present, typical entries have been prepared to clarify the logic of the table as it is being used to classify works from the fields under consideration. In the Figure itself, the numbers in the entries imply substantial contributions from the fields listed in the legend; verbal descriptors

give typical measures. The bibliography is referenced by fields, with the code at the left indicating where each work would fall in the table.

In order to understand the verbal tags in Figure 1, the implications of Figures 2 and 3 must be examined. Figure 2 represents in tabular form the units of analysis. Row descriptors indicate the units themselves, moving from the smallest to the largest; column headings indicate what the count would be related to. Column-row intersections with an X imply potentially meaningful measures. The measures themselves would depend on which of the "models" listed in Figure 3 was being used.

The easiest to consider is the first, Descriptive measures by hierarchy. This entry can be interpreted as referring to frequency distributions based on suitably defined row-column intersections. Under Semantic measures in Figure 2, for example, typical distributions would be word counts based on number of characters, phonemes, syllables, and morphemes. Under Syntax, typical distributions would be phrase or sentence counts based on the number of characters, phonemes, syllables, etc. All of these measures would fall in row 1.i of Figure 1 as basic counts. It should be noted that the basic counts of Semantics are included in those of Syntax, and those of Syntax included under Discourse (Descriptive measures can apply, of course, to units defined a priori, or derived from the text by appropriate measures.)

Approaches 2 and 3 in Figure 3, Sequential dependency and Structural constraints, have two interpretations here. They can

be used to derive classes of units, which would be classified under row 1.ii and iii of Figure 1; and they can be used with previously defined classes, in which case the measures would fall under 2.ii. In either case, there seems to be no reason why they could not be applied to all units presented in Figure 2. Approaches 4 and 5 are somewhat special. Linear Measures have been used in conjunction with stylistic investigations and are treated in earlier reports. The entry labelled Classifications can best be interpreted as the defined approach of structural linguistics.

Finally, considering the three tables together, all combinations of Figures 2 and 3 imply measures which can be used to clarify the substantive areas of Figure 1, though with two general difficulties. First it is not clear how measures in larger units can apply to smaller ones; how do sentences determine semantics, discourse measures affect sentences? It is clear they are dependent, but difficult at present to infer clear relationships, or interpret any which could be inferred. Second, some of the statistics which can be obtained have marginal or problematic value (not as many as one might at first suppose, though many may be redundant, i.e., derivable from some one or more others, either directly or empirically).

Before detailing the program design which will provide some of the measures defined by these three tables, some general remarks are in order to relate this large body of implied measures to current concerns and current research directions. First, the emphasis here has been solely on intratext measures (as is apparent in Figure 3). No attempt either here or in the program design has been made to incorporate techniques for testing hypotheses, drawing

inferences, obtaining comparative statistics, etc. Packages which perform such analyses are commonly available: the task of the system will be to produce the data required as input to such packages.

No estimate has been calculated for the total number of reasonable language measures which are implied by the three tables. If we consider, as an example, the substantive area comprised in II.1.i,ii,iii, using techniques derived from model 5, operating with units 1 through 10, we have, essentially, descriptive linguistics and an unmanageably large number of items to work with. Or, consider using Markov properties in conjunction with unit 8 (words) to construct an approximate grammar. Two-word dependencies lead to an n x n matrix (largely empty) where n is the number of words in the language. And two word dependencies are demonstrably inadequate. Using model 3, Structural Constraints, what one would ideally like to have is some estimate of the entropy of a language, based on its sentences, not, as has been worked out, the estimated entropy of various of the units 1-4. But this would require enough text to obtain accurate estimates of the likelihoods of occurrence of individual sentences, surely out of the question.

By any criteria, the number of measures which must be considered significant is enormous. It is easy, but probably false, to say that so few measures can be rejected as having no interest because of our yet primitive grasp of the nature of language. While one must acknowledge that the state of the art in language analysis is comparable to that of chemistry prior to Medeleyev's construction of the periodic table -- there gathering

data without reference to a general scheme or theory was the only available strategy -- it seems equally clear that such a generally acceptable scheme would not immediately reduce useful measures to some small number. Language incorporates chemistry, many chemistries, and much more besides. Some sense of scale may be obtained if one were to consider the task of reducing the techniques and methods of all science to a schema which displayed their individual logical natures, their range of applications, their levels of confidence, and their interrelationships. The task of reducing language to a representative schema is no less formidable.

Thus there is little likelihood of drastic simplification in the number and relationships of quantifiable language variables in the near or distant future.

Programming considerations

There are two operational assumptions which underlie the design of the system: maximal incorporation of available measures defined by meaningful combinations of I, II, and III; and the potential of applying them to large amounts of text (millions of words), retaining in files all information necessary to add, delete, or combine selectively any and all measures, eventually in an experimental mode interactively.

Such a program is a large order, only a small part of which will be available in the foreseeable future, though it is being designed with the long range goal in mind. There are five essential parts to such a system, each critical in the success of achieving the long range goal:

1.) a general text processor, which accepts as input minimally coded natural language text and a description of its coding conventions, and produces as output an appropriately transformed and formatted file for use in subsequent processing.

2.) a well-conceived set of files for both textual data and statistical information, capable of being updated in all parts economically, and operated on by a variety of analysis subprograms, each of which adds to or modifies some portion of the files in well-understood ways.

3.) an integrated statistical package with facilities for performing any combination of the counts implied by 3 on any combination of the units listed in 2, using the facilities and goals listed in 1, on any portion of the files produced by the text processor or on outputs of the statistical package itself.

4.) a second statistical facility, employing comparative techniques to operate on the output of (3).

5.) a facility for user interaction through which he describes in a natural and efficient way what parts of the available facilities he wishes to employ on what portions of the text and what he wants to do with the output. (This includes, interactively or in batch mode, producing histograms or graphs of measures on hard copy or on CRTs.)

A programming system incorporating the above facilities is a major effort both in the design phase and in the actual programming. Much of the design work has been done in the past year, and every effort has gone into assuring that whatever

preliminary programming is done will be compatible with the long range plans. Such a system, when completed, it is fair to say, will offer an invaluable tool to all investigators involved in natural language analysis. Indeed, one could probably make a good case for asserting that such a facility is essential for continued and orderly growth in knowledge of language processes by offering a context for communicating results, sharing data, and providing a means to involve subject knowledgeable people without programming capability to perform analyses.

Future Directions

Design specifications for existing program parts are presented elsewhere in this report. What remains is a word about what the next phase of development should bring. Given an adequate data file definition, which we believe we have, and an organization of processing steps capable of realizing design goals, there are three areas which require further consideration and development, and such improved knowledge can only come about through experimental use of a prototype of the full system. These areas are the user language for communication with the system, the required statistical routines beyond those outlined above, and the most useful form for the analyst output (to be contrasted here with the output stored in the permanent data files): summary statistics, thresholding capabilities, graphic output, and the like. Each of these is a major effort in itself and each, in various ways, requires more experience with the kinds of research the package is being developed to

support. The first, for example, requires a fine trade-off between the precision required of such languages in their operation and the convenience of users unsophisticated in the formalisms of programming languages: like all such special purpose languages, it must have strong subject orientation, but retain precision, logical clarity, and consistency. The other two depend on what proves useful.

Beyond programming considerations, it is clear that the efforts sketched here to unify the gathering and interpretation of language statistics into some general frame must be pursued. In addition to adding detail to the substantive taxonomy (Figure 1) through an extensive annotated bibliography drawing from those fields engaged in such research (currently being prepared), and perhaps modifying or extending it at the same time, some effort should be directed toward drawing out the theoretical implications of the network of concerns manifested in the total field. Where has the emphasis been placed and why? What are the points of stress, where are the lacunae? Is it possible to relate the variety of statistical models being pursued into some theoretical frame? Is there some coherent way to integrate such models into one comprehensive model of human communication? All of these questions, some premature perhaps, are worth pursuing and can only be pursued from a perspective as broadly defined as possible. One major goal of the work being described here is to encourage such thinking and provide a point of departure for it.

Figure 1. SUBSTANTIVE TAXONOMY

	I. Semantics	II. Syntax	III. Discourse
1. String analytic			
i. basic	1,4,5,6,8 basic count by root,word.	1,2,4,5,6,8 basic count by phrase, clause, sentence	1,2,5,6,8 basic count by text.
ii. categorical	1,2,3,4,5,6,8 word classes by cooccurrence.	1,2,4,5,6,8 phrases, sentences.	1,2,5,6,8 word clauses by cooccurrence.
iii. structural	1,2,4,5,6,8 clusters, clumps	1,2,4,5,8 word, structure dependencies.	1,2,5,6,8 class dependencies
2. A Priori			
i. categorical	1,2,3,4,5,6,7 morphemes, roots, component sets.	1,2,4,5,6,8 parts of speech, structural classes.	1,2,3,5,6,8 content categories - connectors.
ii. structural	1,2,3,4,6,7 thesauri, componential set dependencies.	1,2,4,5,6,7,8 parts of speech and structural dependencies.	2,3,5,8 cooccurrence of content categories.

Legend (field or subfield)

1. Literary analysis
2. Information storage and retrieval
3. Content analysis
4. Language acquisition
5. Linguistics
6. Psycholinguistics
7. Psychology
8. Communication sciences

FIGURE 2

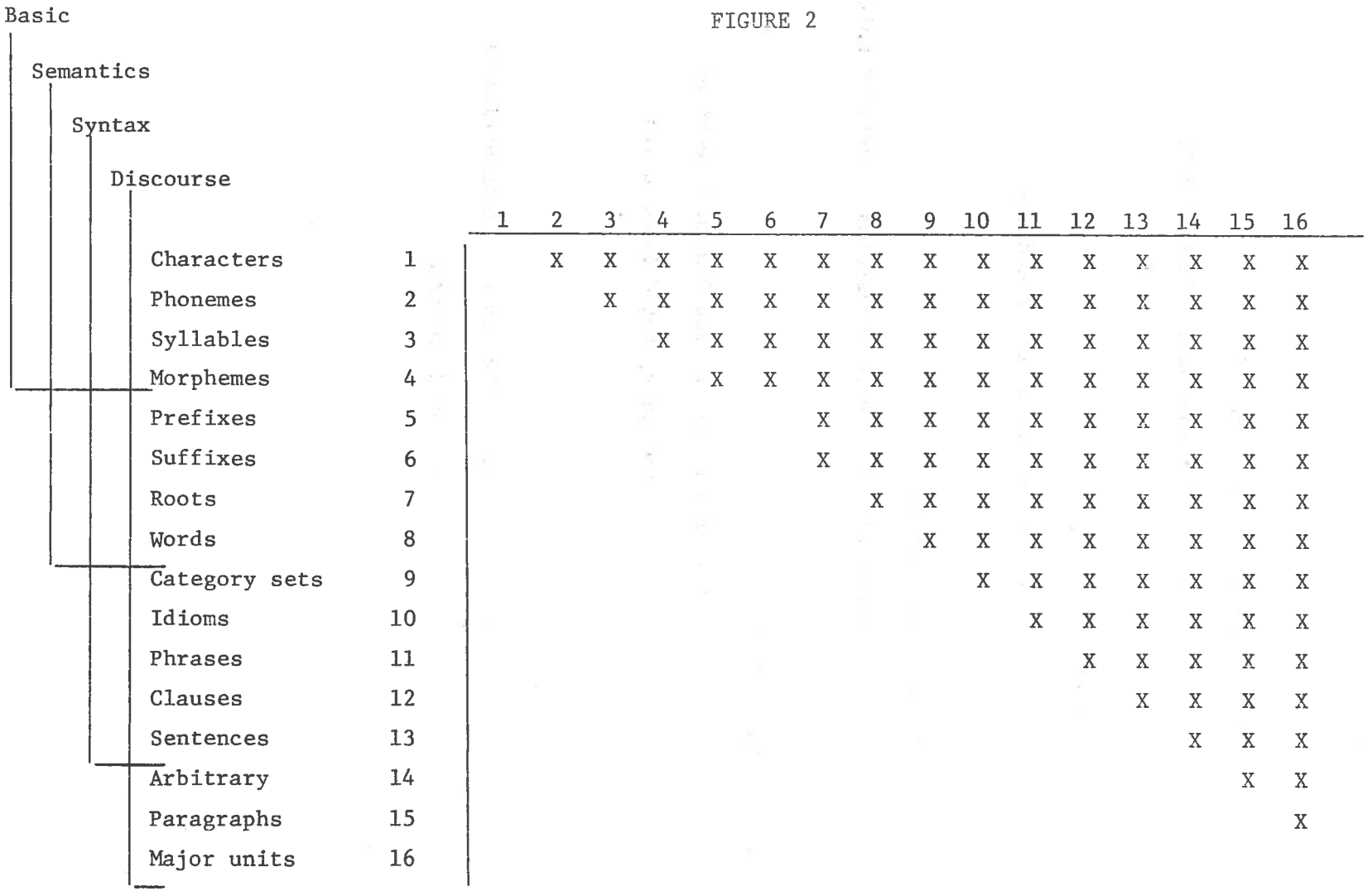

Basic
Semantics
Syntax
Discourse
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Characters 1
Phonemes 2
Syllables 3
Morphemes 4
Prefixes 5
Suffixes 6
Roots 7
Words 8
Category sets 9
Idioms 10
Phrases 11
Clauses 12
Sentences 13
Arbitrary 14
Paragraphs 15
Major units 16

FIGURE 3 - ANALYTICAL MODELS

1. Descriptive measures by hierarchy

   frequency distribution of defined units

2. Sequential dependency

   Markov properties of defined units

3. Structural constraints

   information theoretic properties of defined units

4. Linear measures

   serial properties of text (instantaneous or cumulative) for defined units

5. Classifications

   grouping of units through the distribution of their environments

BIBLIOGRAPHY

General works are those which contain methodological, global or other noncategorical support. Major works are those which could be inserted in many places since they use a number of measures or techniques; they are listed without a link to Figure 1. Other entries use primarily the method implied by their point of entry, appearing to the left in the margin, of the taxonomy related to Figure 1.

General

Bailey, R.W. and D.M. Burton, S.N.D. English Stylistics: A Bibliography. Cambridge, Mass.: The M.I.T. Press, 1968.

Bendix, E.H. "Componential Analysis of General Vocabulary: a Semantic Structure of a set of Verbs in English, Hindi, and Japanese." International Journal of American Linguistics, Vol. 32, no. 2, Part II (April 1966). Also, The Hague: Mouton, 1966.

Botha, Rudolf P. The Function of the Lexicon in Transformational Grammar. The Hague: Mouton, 1968.

Herdan, G. The Calculus of Linguistic Observations. The Hague: Mouton, 1962.

_____. Quantitative Linguistics. Washington: Butterworths, 1964.

_____. Type-Token Mathematics. The Hague: Mouton, 1960.

Lenneberg, E. Biological Foundations of Language - section on color. New York: John Wiley and Sons, 1967. Pp. 337-370.

Meetham, A.R. Encyclopedia of Linguists -- Information and Control. Oxford: Pergamon Press, 1969.

Marcus, Solomon. Algebraic Linguistics; Analytical Models. New York: Academic Press, 1967.

Milic, L.T. Style and Stylistics -- an Analytical Bibliography. New York: The Free Press, 1967.

Sedelow, S.Y. and W. A. Sedelow, Jr. "A Preface to Computational Stylistics." Reprinted from The Computer and Literary Style, Kent Studies in English, No. 2. This article originally appeared as SDC document SP-1534, February 1964.

Sokol, R.R. and P.H.A. Sneath. Principles of Numerical Taxonomy. San Francisco and London: W. H. Freeman, 1963.

1. Literary Analysis

Burwick, F. L. "Stylistic Continuity and Change in the Prose of Thomas Carlyle." In Statistics and Style. L. Dolezel and R.W. Bailey (Eds.) New York: American Elsevier, 1969. Pp. 178-196.

Carroll, John B. "Vectors of Prose Style." In Statistics and Style. L. Dolezel and R.W. Bailey (eds.) New York: American Elsevier, 1969. Pp. 147-155.

Dolezel, L. and R. W. Bailey. Statistics and Style. New York: American Elsevier, 1969.

Miles, Josephine. Eras and Modes in English Poetry. Berkeley and Los Angeles: University of California Press, 1964.

Milic, Louis T. A Quantitative Approach to the Style of Jonathan Swift. The Hague: Mouton, 1967.

III.2.i Mosteller, F. and D. L. Wallace, "Inference in an Authorship Problem; A Comparative Study of Discrimination Methods Applied to the Authorship of The Federalist Papers." Presented at a session of Special Papers invited by the Presidents of the American Statistical Association, The Biometric Society, and the Institute of Mathematical Statistics at the statistical meetings in Minneapolis, Minn., September 9, 1962.

Sebeok, Thomas A. Style in Language. Cambridge, Mass., and New York: The M.I.T. Press, 1960.

I.1.ii Yule, George U. The Statistical Study of Literary Vocabulary. Cambridge, Mass.: The M.I.T. Press, 1964.

2. Information Storage and Retrieval

I.1.iii and III.1.iii Bonner, R.E. "On Some Clustering Techniques." IBM Journal of Research and Development, Vol. 8, no. 1 (January 1964).

Doyle, L.B. "The Microstatistics of Language." Information Storage and Retrieval, Vol. 1, no. 4 (November 1963).

5. Linguistics

Akhamanova, O.S., et. al. Exact Methods in Linguistic Research. Berkeley and Los Angeles: University of California Press, 1963.

Allerton, D. J. "The Sentence as a Linguistic Unit." Lingua, Amsterdam, Vol. 22, no. 1 (1969), pp. 27-46.

I.1.ii and II.2.i Card, William and Virginia McDavid. "English Words at Very High Frequency." College English, Vol. 27 (1966), pp. 596-604.

III.1.i and ii Dewey, Godfrey. Relative Frequency of English Speech Sounds. Cambridge, Mass.: Harvard University Press, 1923.

II.2.i Earl, L.L., B.V. Bhimani and R.P. Mitchell. "Statistics of Operationally Defined Homonyms of Elementary Words." Mechanical Translation, Vol. 10:1, no. 2 (1967), pp. 18-25.

II.1.ii Harris, Zellig S. String Analysis of Sentence Structure. The Hague: Mouton, 1962.

_____. Methods in Structural Linguistics. Chicago: The University of Chicago Press, 1951. (Reprinted as Structural Linguistics, 1961.)

I.2.i Harwood, F.W. and A.M. Wright. "Statistical Study of English Word Formation." Language, Vol. 32 (1956), pp. 260-273.

III.1.i Hultzen, L.S. et. al. Tables of Transitional Frequencies of English Phonemes. Urbana: University of Illinois, 1964.

Kucera, H. and W.N. Francis. Computational Analysis of Present Day American English. Providence, R.I.: Brown University Press.

II.2.ii and II.2.i Lakoff, George. "Instrumental Adverbs and the Concept of Deep Structure." Foundations of Language, Vol. 4, no. 1 (1968), pp. 4-29.

I.1.ii Resnikoff, H.L. and J.L. Dolby. "The Nature of Affixing in Written English." Mechanical Translation, Vol. 9, no. 2 (June 1966).

III.1.ii Sedelow, Sally Yeates. Stylistic Analysis: Report on the Third Year of Research. Santa Monica: Systems Development Corporation, Report TM-1908/300/00, 1967.

I.2.ii Simmons, R.F. and J.F. Burger. A Semantic Analyzer for English Sentences. SDC (SP-2987), January 1968.

II.1.ii and III.1.i Spang-Hanssen, Henning. "Sentence Length and Statistical Linguistics." Structures and Quanta: Three Essays on Linguistic Description. Copenhagen, 1963.

I.2.ii White, J. H. The Methodology of Semantic Analysis with Special Application to the English Preposition. SP series (1339). Santa Monica: Systems Development Corporation, 1963.

I.2.ii Zimmer, Karl E. "Affixal Negation in English and Other Languages; an Investigation of Restricted Productivity." Supplement to Word, Vol. 20.

6. Psycholinguistics

II.2.i Brown, Roger. "Linguistic Determinism and the Parts of Speech." Journal of Abnormal and Social Psychology, Vol. 55 (1957), pp. 1-5.

III.2.i Eiferman, R. "Negation: A Linguistic Variable." Acta Psychologica, Vol. 17 (1960), pp. 258-273.

Gardner, J.W. "Psycholinguistics: A Survey of Theory and Research Problems." JASP, Vol. 49 (1954), supp.

Schlesinger, I.M. "Sentence Structure and the Reading Process." (Jan. Linguarum, Series Minor 69) The Hague: Mouton, 1968.

7. Psychology

II.2.i Boder, David P. "The Adjective-Word Quotient: A Contribution to the Psychology of Language." Psychological Record, Vol. III (1940), pp. 309-343.

Deese, James. The Structure of Associations In Language and Thought. Baltimore: Johns Hopkins, 1966.

Hunt, E.B. Concept Learning: An Information Processing Problem. New York: Wiley, 1962.

II.2.i Miller, G. A. "Some Psychological Studies of Grammar." American Psychologist, Vol. 17, no. 11 (November 1962), pp. 748-762.

8. Communication Science

III.2.i Barber, Charles L. "Some Measurable Characteristics of Modern Scientific Prose." Contributions to English Syntax and Philology, ed. by Frank Behre. Gothenberg, 1962, pp. 21-43.

II.2.i Earl, L. L. (Lockheed Palo Alto Research Lab., Calif.) "Automatic Determination of Parts of Speech of English Words." Mechanical Translation, Vol. 10, no. 3-4 (1967), p. 53-67.

Gerbner, George, et. al. The Analysis of Communcation Content. New York: Wiley, 1969.

III.1.i, ii, and iii Shannon, C. E. "Prediction and Entropy of Printed English." Bell System Technical Journal, Vol. 30, p. 50.

III.1.i, ii, and iii Shannon, C. E. and W. Weaver. The Mathematical Theory of Communcation. Urbana: University of Illinois Press, 1949.

II.2.i Stolz, W. S. Syntactic Constraint in Spoken and Written English. 1964 Doctoral Dissertation (Philosophy-Mass Communications), University of Wisconsin.

3. Design Requirements for a Generalized Data Formatting Program

by

Gerald Fisher

Our plan for a statistical package led us to investigate the basic data structure formats such a package would require. After considerable discussion we decided that a generalized program was needed. This program would provide not only the units for statistical analysis but also the basic computer representation of text for the VIA system. Speed, modularity, and adaptability to diverse research goals were the guiding design criteria. The concepts of the design, if not the program itself, should be valuable to other researchers in the program itself, should be valuable to other researchers in the text processing area. Although the program design is oriented toward an OS/360 implementation, the specifications could easily be changed to suit the operating environment of any medium or large scale computer. It is now generally recognized that operating systems are integral to the specification of high level programming languages and applications programs. Although

there is considerable diversity in the operating systems presently in use, it should nevertheless be possible to adapt the concepts described herein to systems other than OS/360.

The work reported here represents the joint efforts of S. Sedelow, G. Fisher, M. Dillon, J. Smith and D. Wagner. A preliminary version of the program has been written by Joe Ragland, an M.S. student in Computer and Information Science, under the direction of G. Fisher as part of his Master's Thesis. A listing of his program appears in an appendix to this report. The design of the control card language reported herein (but not implemented in the preliminary version) is due in large measure to William Blair, a student of the University of North Carolina; Mr. Blair performed this work as a course project for a Computer Science course in text processing taught by Gerald Fisher.

This report is in two sections. The first delineates the specifications of the generalized program. The second describes the preliminary implementation achieved by Joe Ragland.

The purpose of the program is to produce a machine readable 'index' of an input text. The term general is used because the program is really only a utility routine. Hopefully, it will be used by many people, each specifying a different combination of options to the program, to produce the exact type of index desired. The actual processing of the text will be accomplished in programs which have as input the output of the general index program.

Conceptually the input text is viewed as a stream of characters in which meaningful units are separated by user defined delimiters. Each unit is called a category. Categories may be arranged into one or more independent groupings called hierarchies. These may be used to signify logical groupings, such as sentence, paragraph, and chapter, or physical groupings, such as line number and page number. In the text a category is marked by the presence of delimiters appropriate to that category. Units at the lowest or base level are called "words" or "tokens". It should be noted, however, that it is possible to consider individual characters or analogous units the "base" level by specifying the null string (") as the delimiter for the base category (see example on page 59). The index program divides the input stream into tokens and in sequence associates with each one its relation to the user selected hierarchies, an indication of the word type, and a pointer to the list of occurrences for that word type.

There are two types of input to the program: program control and input text specifications, and the input text(s). The output of the program consists (logically) of four files. These are:

(1) the Token file, which contains the text tokens in their linear order.

(2) the Type Index file, which contains in alphabetical order each type together with a list of pointers to each occurrence (token) of that type.

(3) the Type Glossary file, which is essentially identical to the type index file except that the list of pointers is not present.

(4) the Group Index file, which contains pointers to the rightmost token in each group of every category except the base category.

For each execution of the program, certain parameters are specified by the users: these may be constant throughout the entire run or may vary for each text processed. Utilizing the output of the program it is possible, for example, to determine very quickly and efficiently whether or not the text contains a certain type and to enumerate all tokens representing that type.

Example

To make these ideas concrete we consider a simple and artificial example. We assume the text to be presented to the machine as a stream of characters. The text to be indexed is as follows:

AA B C B C AA . D B A .. AA A

B $$$ A B . AA D .. D $$$ ¢¢¢

In this example there are two independent hierarchies. The first represents the logical organization of the text while the second represents its physical arrangement. The base category uses a blank to delimit tokens; this category is the only one assumed to belong to each hierarchy. It is not numbered or named. The category and hierarchy specifications would be as follows:

Hierarchy	Category	Delimiter Set
	BASE	BLANK
1	1 (SENTENCE)	.
1	2 (PARAGRAPH)	..
1	3 (CHAPTER)	$$$
2	1 (VOLUME)	¢¢¢

The manner in which this control information is given to the program will be delineated below. The output produced would be as follows:

TOKEN FILE

length	token	type pointer	CC
2	AA	6	0000
1	B	7	0000
1	C	8	0000
1	B	7	0000
1	C	8	0000
2	AA	6	0000
1	.	2	0000
1	D	9	1000
1	B	7	1000
1	A	5	1000
2	..	3	1000
2	AA	6	0100
1	A	5	0100
1	B	7	0100

TOKEN FILE (continued)

length	token	type pointer	CC
3	$$$	4	0100
1	A	5	1010
1	B	7	1010
1	.	2	1010
2	AA	6	0010
1	D	9	0010
2	..	3	0010
1	D	9	1110
3	$$$	4	1110
3	¢¢¢	1	0000

TYPE INDEX FILE

length	type	freq.	occur.
3	¢¢¢	1	24
1	.	2	7,18
2	..	2	11,21
3	$$$	1	23
1	A	3	10,13,16
2	AA	4	1,6,12,19
1	B	5	2,4,9,14,17
1	C	2	3,5
1	D	3	8,20,22

TYPE GLOSSARY

length	type	freq.
3	¢¢¢	1
1	.	2
2	..	2
3	$$$	1
1	A	3
2	AA	4
1	B	5
1	C	2
1	D	3

GROUP INDEX FILE

hier.	cat.	token numbers
1	1	7,11,15,18,21,23,24
1	2	11,15,21,23,24
1	3	15,23,24
2	1	24

The field designated "length"contains the token or type length. The field "type pointer" gives the rank of the token in the Type file. Thus, the token "AA" occurs sixth in the Type list.

The field labeled "CC" serves to indicate category changes. This field is a bit string of length equal to the number of categories (excluding the base category). In this example there are four categories besides the base category. Each position in this string corresponds to one of these categories. Thus, the first

position is used to indicate changes in the SENTENCE category, the second changes in the PARAGRAPH category, etc.

The field labeled "occur" in the Type Index file is a vector of extent equal to the frequency. This vector contains in order the ranks of the type in the Token file. These ranks are called the linear numbers or linear occurrences of the type in the text. Thus, the word type "AA" occurs as the first, sixth, twelfth and nineteenth text tokens.

Note that the use of linear numbers instead of the complete indexing information (i.e., sentence number, paragraph number, chapter number, and volume number) results in greater storage efficiency. The cost, however, is that the Group Index file must be created. This file indicates for each category the linear numbers of the rightmost token for each instance of that category. For example, the first sentence ends at linear position 7, the second at position 11, etc. When used in conjunction with the Type Index file, the Group file facilitates the construction of a complete index.

In this example all delimiters are treated as tokens. In general this is not desirable since a delimiter is frequently extraneous to the text and introduced only for the purpose of classification. Thus, we distinguish between natural and unnatural delimiters. A natural delimiter is one such as a period, that is to be treated as a token and assigned a linear number. An unnatural delimiter such as $$, is used to force a category change but is not retained in the Token file. The DELIMITER control card, to be described below, contains a parameter for

distinguishing between natural and unnatural delimiters.

Overall Program Structure

The general data formatting programs consists of four basic routines. The first of these is the Input Specification Processor. By means of control cards the user specifies the text, hierarchies, categories, delimiter sets, and other control information to the program. The input specification processor interprets this control information and modifies the program accordingly. The Text Scanner is the second routine. This part of the program separates the text tokens and delimiters. Each token (and natural delimiter) is assigned a linear number and a bit string, called the category change (or CC) indicator which signals by alternation changes in category grouping. The token, together with its linear number and CC indicator, is fed to a general sorting program. The linear number of the rightmost token of each category is also sent to the sort. The third step is the First Token Sort. The tokens are sorted into alphabetical order and the sorted file is used to construct the Type Index, Type Glossary, and Group Index files. The Token file is rewritten but appended to each token record is its type pointer. The fourth step is a Second Sort of the Token file. This sort puts the file back into linear order. In the final phase of the sort, the linear number is deleted from the token record. The entire process is repeated for each input text to be processed. A detailed description of these steps and the job and program control statements which might be used for their execution in an OS/360 environment follows.

Input Specification Processor

The general index program is controlled by job control statements and program statements. Program control statements are read from a data set defined by the SYSIN DD statement. Usually this data set will be located in the input stream, and the DD statement defining it would appear as follows:

```
//SYSIN DD  {*
            {DUMMY
.
.
.
Program control statements
.
.
.
/*
```

It is possible to have no program control statements. In that case, the SYSIN DD statements may be omitted entirely, or DUMMY coded, resulting in utilization of default options as indicated in the discussion of the individual program control statements.

Program Control Statements

Program control statements are coded in a manner similar to that for the OS/360 system utilities. The reader may wish to refer to the IBM Systems Reference Library publication Utilities, form GC28-6586, Appendix C.

Program control statements have the following standard format:

name operation operand (comments)

The name symbolically identifies the control statement and can be omitted at the discretion of the user. When included, the name must begin in column one and can consist of one to eight alphameric characters, the first of which must be alphabetic, and must be followed by one or more blanks.

The operation specifies the type of control statement. It must be preceded and followed by one or more blanks.

The operand consists of one or more positional or keyword parameters separated by commas. The operand field must be preceded and followed by one or more blanks.

If desired, comments may be written in a control statement, but they must be separated from the last parameter of the operand field by at least one blank.

Program control statements are coded on cards or card images. Columns 73 through 80 are ignored. The control statement must be contained in columns 1 through 71. A control statement that is longer than 71 characters may be continued onto as many additional cards as necessary. Continuation may be indicated in several ways. The operand may be interrupted in column 71 and a nonblank character placed in column 72. The operand may be interrupted after a comma and a nonblank character in column 72, or column 72 may be left blank. Note that if the last character of the operand is not a comma (whether or not it is in column 71) the continuation must be explicitly indicated by a nonblank character in column 72, else the control statement will be interpreted as one which is exactly 71 characters long. The continued portion of the control statement must begin in column 16 of the following card; columns 1

to 15 must be blank. Comments may be placed on any card containing a complete or partial control statement.

```
FIRSTONE    INDEX     MAXDELIM=30,LENGTH=20,STORAGE=SIZEC
            CATEGORY  TYPE=BASE   DEFINITION OF BASE CATEGORY
DELSI       DELIMITERS SINGLE='"#$%"()*+@¢?/.,:;',          *
            BLANK=YES
PARAGPH     CATEGORY   NAME=PARAGRAPH
DELS2       DELIMITERS BLANK=NO,'###','$$$'='NEWPGH'
```

Notation for Defining Control Statements

The hyphen (-), underscore (_), braces ({}), brackets ([]), or symbol (1), and ellipsis(...) are used to define control statements but are never written in an actual statement. Upper case letters and words, numbers, and the apostrophe ('), asterisk (*), comma (,), equal sign (=), parentheses (()), and period (.) are written in an actual control statement exactly as shown in the statement definition. Lower case letters and words appearing in a statement definition represent variables for which specific information is substituted in an actual statement. Stacked items represent alternatives; only one should be selected. Hyphens join lower case letters and words to form a single variable, e.g., text-ddname. An underscore indicates the default option. Braces and brackets group related items, such as alternatives. However, brackets indicate optional items; everything within brackets may be omitted. An ellipsis indicates that the preceding item or group of items may be repeated more than once in succession.

Program Control Statements

The following are the types of control statements accepted by the Input Specification Processor:

The INDEX Statement

The INDEX statement is used to request that a text indexing operation be performed and to specify the maximum length and maximum number of delimiters, and the amount of storage that the index program may use. The statement is coded as follows:

```
[name]   INDEX MAXDELIM=n,LENGTH=m,[,STORAGE=(to be defined later)]
             [,LISTDD=listddname]
```

If the INDEX statement is not present, then no CATEGORY or DELIMITERS control statements can appear.

MAXDELIM=n

Specifies that a maximum of n delimiters will be specified in subsequent DELIMITERS control statements. If this limit is exceeded the program will not continue execution.

LENGTH=m

Specifies that the maximum length of any literal that will be specified in subsequent DELIMITERS control statements is m. If a delimiter is specified that is longer than this length, the remaining characters will be ignored, and a warning message issued. The maximum length that is permitted is implementation defined but normally would not be more than 53 (see description of DELIMITERS control statement).

STORAGE=(to be defined later)

Specifies that the index program is to use no more than the indicated amount of storage. (This will have to be fudged since PL/I does not allow conditional storage allocation requests. When implemented, the correspondence between core used and, say, the dimension of some dynamically allocated variable will have to be empirically determined.) If this parameter is absent, the program will attempt to use all available storage.

LISTDD=listddname

Specifies that the DDNAME to be used for the message dataset is listddname. If omitted, SYSPRINT is assumed. If nullified, no listing is produced.

A possible embellishment is to permit any of the options of the INDEX statement to be specified in the PARM field of the EXEC statement. This would serve two purposes: (1) if no INDEX statement were provided (e.g., //SYSIN DD DUMMY or no //SYSIN DD statement at all), these options could be specified, and (2) the options specified in the PARM field could override the normal defaults (if any), setting new ones for a whole batch of INDEX's (unless explicitly specified on the INDEX statement).

The CATEGORY Statement

The CATEGORY statement is used to specify up to sixteen additional categories for text processing. If used, all CATEGORY statements must follow the INDEX statement, and come before all TEXT statements. There are two versions of the CATEGORY statement which are coded as follows:

(1) [name] CATEGORY TYPE=BASE

Specifies the base category. It is not necessary to code a CATEGORY statement for the base category unless one wishes to provide a DELIMITERS statement to override the default values assumed when no DELIMITERS statement is coded. This version of the CATEGORY statement indicates that the specifications on the following DELIMITERS statement (if any) are to apply to the base category.

(2) [name] CATEGORY NAME=category-name[,HIERARCHY=n]

Specifies an additional category other than the base category. Any subsequent DELIMITERS control statement applies to this category.

NAME=category-name

Specifies the category name. For example, NAME=PARAGRAPH. This operand must be present when the second version of the category statement is used, and must consist of one to twenty alphameric characters.

HIERARCHY=n

Specifies that the designated category is a member of hierarchy n. If this keyword is omitted, then 1 is assumed. The hierarchy numbers specified on successive CATEGORY control statements must be not more than one greater than the previous one or an error will be signaled and the program will not continue execution. (Note that the hierarchy number of the base category is effectively zero.) The maximum hierarchy number is 16.

The DELIMITERS Statement

The DELIMITERS statement specifies the category delimiter set. If this statement is omitted, then defaults may be provided depending upon the hierarchy number and the implementation conventions. This statement, if used, must immediately follow the CATEGORY statement to which it refers. The statement is coded as follows:

```
[name]     DELIMITERS SINGLE='...string of characters...'
           BLANK={YES|NO}
           {'xxxxxx'['='yyyy']...}
           [PICTURE='zzzzzz']
```

SINGLE='...string of characters...'

Specifies that each character enclosed within the apostrophes is to be considered individually as a natural delimiter. The maximum length of this string is implementation defined, but should not necessarily be dependent upon the LENGTH parameter of the INDEX statement. If it is desired to include an apostrophe, it must be represented as two consecutive apostrophes.

For the base category only, it is possible to specify SINGLE='', that is, nullify the natural delimiter set. This specification causes the program to treat each letter (character) as a token. Note that BLANK=YES may still be specified. Indeed, any of the delimiter specification options described below are compatible with SINGLE=''. Whenever more than one delimiter is specified, the program will always scan for delimiters in order of decreasing length. Treating the blank as a token may be useful in certain instances where the format of the text has been altered, or it is simply desired to consider blanks to be tokens.

BLANK={YES|NO}

specifies whether or not the blank character is to be considered a category delimiter. If this operand is omitted then NO is assumed. YES would normally be specified only for the TYPE=BASE category. (This operand is necessary because the text might have been altered from the conventional format of texts for various valid reasons, and blanks do not delimit tokens.) If BLANK=YES, the blank character will be considered an unnatural delimiter. However, there is an exception to this described below.

'XXXXXX'

specifies that the string 'XXXXXX' is to be considered a category delimiter. The string 'XXXXXX' will be treated as a natural delimiter, and can be of any length up to the implementation defined maximum (but not exceeding that specified on the INDEX control statement). In particular, strings of length one and the null string may be specified. Specification of the null string is equivalent to SINGLE=''. If the string is the blank character, then the blank is viewed as a natural delimiter.

[='YYYYYY']

specifies that the string 'YYYYYY' is to replace the string 'XXXXXX' in the TOKEN file. If 'YYYYYY' is the null string, then the delimiter 'XXXXXX' is not to appear as a token - i.e., it is an unnatural delimiter. The replacement string is optional. The specification ' ' ='' is equivalent to BLANK=YES. The specification '$$$'='' implies that $$$ is an unnatural delimiter.

PICTURE='ZZZZZZ'

Specifies that the delimiter contains variable alphanumeric information. A variable digit is coded by a "9", a letter by a "V". Constants are enclosed in parentheses. Thus, PICTURE='(MILT)9999' signifies that any text string composed of the string "MILT" followed by four digits is a delimiter. This feature can be used for numbering lines of poetry.

Default options will be implementation defined for all categories (including the base category); consequently, the use of DELIMITERS statements is optional.

The TEXT Statement

The TEXT statement is used to identify the texts that are to be processed and the data sets onto which the index program output is to be written. The statement may also be used to specify margins for text retrieval. The statement is coded as follows:

```
[name]    TEXT   [FROMDD=text-ddname,TODD=(token-file-ddname,
                  type-index-file-ddname,text-glossary-ddname,
                  group-index-ddname),]
                 [NAME=textname,]
                 [LENGTH=n,]
                 [FIELD=(length,left-margin,right-margin)]
```

FROMDD=text-ddname

Specifies the DDNAME of the data set from which the program is to read the input text. If this operand is specified then the TODD operands must be also. If omitted, the default name TEXT is supplied.

TODD=(token-file-ddname,type-index-file-ddname,text-glossary-ddname,group-index-ddname)

Specifies the DDNAMES of the data sets that the program is to use for the four output index data sets. If this operand is specified then the FROMDD operand must also be given. If omitted, defaults of (TOKEN,TYPE,GLOSSARY,GROUP) are used.

NAME=textname

Specifies that the text name is to be textname. If there are blanks in the textname then it must be enclosed in apostrophes. If this operand is omitted the input DDNAME is used.

NAME=*

Specifies that the first logical record of the text dataset is to be interpreted as the text name.

LENGTH=n

Specifies the maximum token length to be admitted is n. Tokens in the input text longer than n characters will be split into two or more "tokens" of the maximum length. If this operand is not specified then a default will be provided.

FIELD=(length,left-margin,right-margin)

Specifies the length of a logical record in the input text dataset and the limits of scan for text. Since the input text file is processed by STREAM I/O this parameter need not be specified unless the left or right margins are not 1 and length, respectively. If not specified, defaults of (80,1,80) are

supplied. The program will read length bytes and extract text from positions left-margin through right-margin for processing. For example, a specification of FIELD=(400,1,320) and an input text dataset actual LRECL of 80 would result in every fifth logical record (e.g., card) being skipped.

The END Statement

The END statement is used to terminate all processing. Its use is optional. If coded, any control cards after it will not be processed. The statement is coded as follows:

```
[name]     END     optional information/comments
```

Examples

The following example illustrates the simplest procedure that can be executed by the general index program.

```
//INDEXIT JOB (acct.info),'pgmr.name',MSGLEVEL=(1,1),REGION=44K
//STEP1   EXEC  PGM=GENINDX
//STEPLIB DD    DISP=SHR,DSNAME=your.program.library
//SYSPRINT DD   SYSOUT=A,SPACE=(space information),
//   DCB=(RECFM=VBA,LRECL=137,BLKSIZE=1648)
//TEXT     DD   DISP=SHR,DSNAME=input.text.dsname
//TOKEN    DD   DISP=(,PASS),DSN=&TKNFIL,UNIT=SYSDA,
//   SPACE=(space information),DCB=(DSORG=DA,OPTCD=R,BLKSIZE=nn)
//TYPE     DD   DISP=(,PASS),DSN=&TYPNDX,UNIT=SYSDA,
//   SPACE=(space information),DCB=(DSORG=DA,BLKSIZE=nn,
//   KEYLEN=max-token-length+1)
//GLOSSARY DD   DISP=(PASS),DSN=&GLOSRY,UNIT=SYSSQ,
//   SPACE=(space information),LABEL=(,SL)
```

```
//GROUP    DD   DISP=(,PASS),DSN=&GROUP,UNIT=SYSDA,
//   SPACE=(space information),DCB=(DSORG=DA,BLKSIZE=??)
//SYSIN  DD  DUMMY
//
```

Input Specification Example

The following example illustrates the processing of more than one text with varying specifications.

```
//EX2 JOB (acct.info),'pgmr.name',MSGLEVEL=(1,1),REGION=44K
//STEPONE  EXEC PGM=GENINDX,PARM ='LISTDD=OUTPUT'
//STEPLIB DD DISP=SHR,DSNAME=your.program.library
//OUTPUT  DD SYSOUT=A,SPACE=(space information),
//   DCB=(RECFM=VBA,LRECL=137,BLKSIZE=1648)
//TEXT1   DD *
this is the text name for the first text to be processed!
//         DD DISP=SHR,DSNAME=STORED.TEXT1
//TOKEN1  DD DISP=OLD,DSNAME=TEXT1.NEW.TOKENFIL
//TYPE1   DD DISP=OLD,DSNAME=TEXT1.NEW.TYPEFIL
//GLOSS1  DD DISP=OLD,DSNAME=TEXT1.NEW.GLOSSARY
//GROUP1  DD DISP=OLD,DSNAME=TEXT1.NEW.GROUPFIL
//TEXT2   DD DISP=OLD,UNIT=(2400,2,DEFER),LABEL=(,NL),
//   VOL=(,RETAIN,2,,SER=(TAPE01,TAPE02)),
//   DCB=(RECFM=FB,LRECL=130,BLKSIZE=2600,EROPT=ACC)
//TOKEN2  DD DISP=(,PASS),DSN=&TOKEN,UNIT=2314,SPACE=(CYL,(5)),
//   DCB=(DSORG=DA,OPTCD=R,BLKSIZE=16)
//TYPE2   DD DISP=(,PASS),DSN=&TYP,UNIT=2314,SPACE=(CYL,(5)),
//   DCB=(DSORG=DA,BLKSIZE=nn,KEYLEN=16)
```

```
//GLOSS2  DD DUMMY,DCB=BLKSIZE=69 (to prevent undefined file on OPEN)
//GROUP2  DD DISP=(,PASS),DSN=&GRP2,UNIT=2314,SPACE=(TRK,(10,2)),
//   DCB=(DSORG=DA,BLKSIZE=??,KEYLEN=20)
//SYSIN DD *
RUNTWO   INDEX MAXDELIM=20,LENGTH=13,STORAGE=SIZEA
BASECAT  CATEGORY TYPE=BASE
BASDLIM  DELIMITERS BLANK=YES,SINGLE='"#$%&*-/,:;+=',"/*","*/",
              '. ',';  ','?',
              '¢'
CAT1     CATEGORY NAME=PARAGRAPH
PGHDLM   DELIMITERS '$$$'=''
CAT2     CATEGORY NAME=XXXXXXXX,HIARCHY=2
XXDLM    DELIMITERS '$$$$$$'=''
TEXT1    TEXT  NAME=*,LENGTH=12,FROMDD=TEXT1,TODD=(TOKEN1,TYPE1,
              GLOSS1,GROUP1),FIELD=(80,2,72)
TEXT2    TEXT  NAME='THIS IS TEXT # 2',LENGTH=15,FROMDD=TEXT2,
              TODD=(TOKEN2,TYPE2,GLOSS2,GROUP2),
              FIELD=(130,22,117)
         END           THIS WAS RUN # 2
//STP2    EXEC PGM=...  to process the output files
...
//
```

Text Scan Processor and Sorts

The second basic routine is the text scan processor. This routine is the most critical factor in the overall performance of the program. It is probable that it will run CPU-limited on a Model 75. The purpose of the text scan processor is to locate

tokens in the input text, assign them a linear number and a category change indicator, and pass them to the first token sort. If enough core is available, processing can be expedited by passing the tokens directly to the OS/360 sort program via the PL/I facilities. This will increase the core requirements by approximately 30K bytes, and the sort program DD statements will have to be added to the job step. The advantage of doing it this way is that a sort routine will not have to be written to handle the expected large volume of text. The OS sort program will sort all of the data in a small region.

As a token is located, its token record is immediately sent to the sort program. When a non-base category change occurs, this is noted and sent to the sort in appropriately coded form for the eventual placement of this data into the Group Index file. This file will be written when the sort is finished. The text scan processor signals the sort routine that all records have been passed. The Token and Group Index file records are then sorted. As they return from the sort program, four processes occur: First, the Group Index file is written; Next, the occurrences of types are noted and the Type Index file is written; Third, the Type Glossary file is written; Finally, each sorted token record with its newly inserted type pointer is written into the Token file.

Thus, after the first token sort is complete, the Type ndex file, the Type Glossary file, and the Group Index file are completed. The Token file has been written, but it is not in linear order. It must be re-sorted in order to get it back into

original text order. This roundabout process is necessary because at the time that the Token file could have been written in linear order, the Type Index file pointer was not available. Re-sorting is the simplest way to get it back into order.

The next process is to send the token records back to the sort routine. This process is termed the second token sort. The OS sort program is again invoked, except that this time it is directed to read the input records from an existing disk data set, namely the Token file, by passing the DDNAME of the Token file to the sort program instead of obtaining the input records via a PL/I procedure as in the first token sort. Having the sort program obtain the input records directly is desirable because the records are already on disk and having the PL/I program read them and pass them to the sort program in a procedure would simply be wasted effort. As the resorted token records return, the Token file is rewritten, this time dropping the linear numbers since it is now in correct order.

The program is now finished. All files are closed and control is returned to the input specification processor. If there is another TEXT control card, the process begins again.

File Processing

As noted previously, there are four output files. One, the Type Index file, is obviously a candidate for REGIONAL(3) organization. The Type Glossary file may be a simple sequential file. It can be located on any "SYSSQ" device, tape being a good candidate if disk space is at a premium because of the Token

and Type Index files. Alternatively, REGIONAL(1) organization can be used. The Group Index file is a special case. It could just as well be sequential, but our preference is to make it REGIONAL(3) for the same reasons that the Type Index file is REGIONAL(3), and access the category records using the category name specified on the CATEGORY control card as the key. It is desirable to place the (logical) Group Index file physically with the Type Index file, that is, as a prefix to the Type Index file. Under certain circumstances this may not be feasible. In particular, the Group Index file records and keys must fit onto the data set which is to contain the Type Index file. Since the maximum size of any block must be specified, placing the Group Index records onto the Type Index file could require a larger maximum blocksize (BLKSIZE) specification, and the length of the keys (KEYLEN) of the file would be the maximum of the length of category names (which is 20) and the maximum token length+1. If one wished to store only 12 byte tokens he would be wasting 37% of the available space on a 2314 track in order to store an extra 21 bytes per Type Index record (see discussion below). This is, however, the worst case. It is much simpler, we feel, to place housekeeping information on a single, separate file, e.g., put all "header" records and the Group Index file on a separate REGIONAL(3) dataset; although not efficient in terms of storage, it is preferable in terms of program complexity and CPU processing time. This data could be accessed once, and the file closed, then reopened every time another access was needed, and the program would run faster and use less core than otherwise. In order to

make a definite choice on this particular subject, it would be helpful to perform some empirical tests.

The Token file is designed to be a sequential file processed by STREAM I/O. REGIONAL(1) is an alternative choice for this file. Unfortunately, this mode of organization would require more storage (since fixed length records must be used).

The Type Index and Group Index files present an unusual application of REGIONAL(3) organization. Usually when one uses a REGIONAL(3) file, there is some obvious relationship between what is going into it and where it should go, i.e., the track or region number. But in this case REGIONAL(3) is being used only because a DIRECT file with variable length records is needed. There is no real need to control the physical placement of any particular record. So what really needs to be done here is to write (create) the Type Index data set "consecutively," packing the records onto the disk just as they come. But a REGIONAL(3) data set cannot be created with a CONSECUTIVE file as can a REGIONAL(1) data set. This inability to use a consecutive file appears to be a problem because the program is forced to specify a region number - which is something it does not really know how to do, since it cannot know how many records will fit onto one track - they are, after all, variable in length.

The solution to this problem takes advantage of the fact that region numbers are automatically incremented -- once. When an attempt is made to write a record to a region (track) that has become filled, the operating system will increment the region number automatically, that is, write it onto the next track

(region). Subsequently, when the same WRITE statement is executed again -- specifying the same (old) region number -- the PL/I KEY condition (sequence error; ONCODE = 53) is raised. When the KEY condition occurs, the program must do three things. First, the region number pointer(s) in the Token file for the previous Type Index file record (which had its region number incremented automatically) must be updated. This updating simply involves delaying the writing of the Token file records until the next Type Index file record has been successfully written. Second, the program variable for the region number must be updated. Third, the interrupted WRITE must be reissued, with the now updated region number. ON UNIT processing consists of up-dating the region number, up-dating the region number pointers in all the token records for the last type, and branching to the offending WRITE statement.

To search the Type Index file for a particular type it is not necessary to know the region number. Simply specifying a region number of 0 (1 if region number 0 contains header data) will cause the entire file to be searched. For a large file, this could be quite slow (approximately 1.6 minutes for a whole 2314 disk pack). If REGIONAL(1) organization is used for the Type Glossary, significant search time reductions for the Type Index file will result if the region number of the type is stored with the Type Glossary record. Another alternative is to store in the Group Index file a table containing the last type written in each region (track). This table would be relatively small and could be rapidly searched.

It should be noted that the use of REGIONAL(3) for the Type Index file has one disadvantage. The word itself is recorded three times: once in the data itself, again for the key area of the record, and once more for the key area of the block (which contains just one record). The last two cannot be disposed of. The type should, therefore, be eliminated from the Type Index record, since to read it, the word itself is required as the key unless the file is being read sequentially; in the latter case the KEYTO option will retrieve the key of the type index record being read.

Output Record Formats

(1) The Text Header Record

This record contains the control information given in the input specification cards and various counts useful for subsequent processing. Precise formating and location are implementation defined. It seems best to locate this header information in the Group Index file. However, there may be advantages to prefixing each file with some portion or all of the header record. The record should contain the following:

(1) Text Name

(2) # of Tokens

(3) # of Types

(4) Max. Token Length

(5) # of Categories (other than base-category)

(6) Vector of Category Names

(7) # of Hierarchies

(8) A hierarchy vector of extent equal to the number of categories -- each element to contain the hierarchy number of the corresponding category.

(9) A vector with frequencies for each category.

(10) A ddname table - a table containing the ddnames of all output files for this text.

(11) Type Index region table (optional) - a vector whose i^{th} entry is the key of the last type recorded on the i^{th} region.

(2) The Token Record

The Token file should contain the following:

(1) Type pointer

(2) Region # (optional)

(3) CC indicator

(4) Token length

(5) Token

The Region # field contains the region number of the word type. Thus fields (1) and (2) enable one to address directly the type index record corresponding to a given token.

Although the Token file is to be processed as a stream file, it is possible to avoid time and space-consuming data conversions. All fields can be written in A-format and processed with the UNSPEC built-in function. The Token record could be processed in PL/I as follows:

```
dcl  1  token_record,
       2  convert
          3 type_ptr fixed(15,0) bin,
```

```
            3 region_#   fixed(15,0) bin,
            3 cc_ind     bit (16),
            3 token_len  fixed(15,0) bin,
          2 token   char(maxtoklen) var,
str  char__(8)__defined convert;
/*  maxtoklen is the maximum token length */
/*  str overlays the convert structure and
    eliminates data conversion            */
get file (tok) edit (str,token) (a(8),a(token_len));

/*  This reads the token record into the structure token-record.
    No data conversions are performed and no extra processing
    steps are required - furthermore when processed in this way,
    a token record requires token len+8 bytes instead of token
    len+28 bytes - a substantial saving.   */
```

(3) The Type Index Record

Type file records should contain the following:

(1) Type length

(2) Type

(3) Type frequency

(4) Vector of indices (linear occurrences) of this type.

Fields (1)-(3) require no explanation. Field (4), however, will necessarily present implementation difficulties. If the Type Index file is sequential (not the preferred alternative), then an extra temporary file will be needed to build the vector of indices (at least for words with large frequency). The reason for this is simply that it is most desirable to have the type

frequency precede the vector of indices. If the file is DIRECT and REGIONAL(3) organization is used, this difficulty does not arise. However, it will be necessary at times for a single type index record to span several logical (and perhaps even physical) data set records. This must happen since it is necessary to bound in advance the logical record length of the Type Index data set. Consequently, there will be a bound on the dimension of the vector of indices, a bound which surely will be exceeded by some word types. When this happens, several logical data set records must be used to represent a single type index record. Thus, the implementation will require an additional character position in the record key to indicate the sequence of the logical data set record within the Type Index record.

(4) The Type Glossary Record

This record will contain fields (1)-(3) of the Type Index record.

(5) The Group Index Record

This record will contain the following:

(1) The Hierarchy number

(2) The Category number

(3) The number of Category Changes

(4) The Vector of rightmost indices for the Category.

All of the considerations given above for the Type Index record apply to field (4) of this record.

A Preliminary Version of the General Data Formating Program

In this section we describe the preliminary version of the program written by Joe Ragland. The ideal version discussed above was not fully designed when the programming effort began and the parallel theoretical and practical work complemented each other nicely. We quickly realized that the first routine, the input specification processor, would entail considerable programming time and yet would not significantly affect expected program execution time. For this reason it was decided to employ, for the most part, the input specifications used by the already existing VIA INDEX program.

Category Restrictions

Emphasis was placed on the efficient handling of large prose texts. Thus the set of delimiters for the base category is restricted to the blank character. All punctuation marks, such as the comma, the quotation mark, etc., are assumed to be separated by one or more blanks. The blank itself is considered an unnatural delimiter. Eight non-base categories and two hierarchy levels are allowed. The first four categories are used for logical text groupings, while the last four are used for physical groupings. Explicit category names are not present. Implicitly, the first two denote sentence and paragraph respectively. The categories and their delimiter sets are as follows:

Hierarchy	Category	Delimiters
1	1 (Sentence)	{.,?,!}
1	2 (Paragraph)	{..}
1	3 (Chapter)	{$$$}
1	4 (Volume)	{$$$$}
2	5 (Line)	{¢}
2	6 (Page)	{¢¢}
2	7	{¢¢¢}
2	8	{¢¢¢}

Each delimiter is assumed to be surrounded by one or more blanks. No interpretation is provided for categories seven and eight. There appeared to be no need for more than eight categories and two hierarchies. Moreover, the restriction to eight categories allowed the use of one byte (or character position) to represent the CC indicator bit string. All delimiters except the sentence delimiters are considered unnatural and are not retained in the token file. The paragraph delimiter, however, is replaced with the sentence delimiter '.' whenever a paragraph change is not immediately preceded by a sentence delimiter.

Text Parameters

For each text to be processed, a text header card must be present on SYSIN. This entire card is retained as part of the header record that prefixes each output file. The text itself need not be on the SYSIN data set. One of the parameters that may be specified on the EXEC card is the input text file ddname. If no ddname is present, SYSIN is assumed. The other parameters

passed in this way are TLEN, CDMGIN, and SIZE. The parameter TLEN gives the maximum token length; the default is TLEN=18. CDMGIN=(XX,YY) specifies that the input text logical records are to be scanned from XX to YY. The default is CDMGIN=(1,72). The SIZE parameter gives the amount of storage available for the OS/360 SORT program. The default is SIZE=44000. Because of the OS/360 sort requirements, TLEN must exceed 10 and SIZE must be at least 30000.

Program Structure

The program is composed of three external procedures - viz., DETEXT, SORT1, and SORT2. At present the program also contains a listing routine for testing purposes.

The DETEXT Procedure

This main procedure scans the PARM string from the EXEC card and determines program options. It then calls SORT1 and SORT2.

The SORT1 Procedure

This procedure performs the following functions: text scanning, first token sorting, and type index, group index, and type glossary building. The OS/360 sort is dynamically invoked within this procedure. As the text is scanned, token records and group index records are sent to the sort. The sorted output is used to write out all files except the token file. The procedure is essentially as described above. There is one important difference, however. It was deemed

more expedient to use only sequential files in this preliminary implementation. This choice necessitated the use of a temporary file to hold the vector of indices whenever the extent of this vector exceeded 100. The need for doing this for the Group Index file was obviated by a decision to change the group index record format to one in which the category number precedes each of its rightmost linear numbers.

The SORT2 Procedure

This procedure performs the second token sort and writes out the token file in final form. An attempt was made to pass the DDNAME of the temporary token file when invoking the sort in order to avoid passing the records themselves to the sort. As indicated above this appears to be the most efficient method. However, technical difficulties were encountered and it was decided to pass the records to the sort.

Record Formats

The record formats employed differ somewhat from those described above. They are as follows:

(1) Header Record

This record is 134 bytes long and precedes each file produced.

Field	Format
file name (e.g., 'TOKEN')	A(5)
text id	A(80)
maximum Token Length	F(3)
total number of tokens	F(6)

Field	Format
number of occurrences:	
category 1	F(5)
category 2	F(5)
.	.
.	.
.	.
category 8	F(5)

(2) Type Index Record

Field	Format
Token Length	A(2)
Token	A(token length)
Frequency	A(2)
Vector of indices	A(4)(Frequency)

Note that internal representation of numeric information is preserved and conversion avoided through the use of A-format and overlay defining. Thus, internally token length is fixed (15,0) and each index is fixed (31,0). These records can be read using the technique illustrated above in the discussion of the Type Index record.

(3) Type Glossary Record

Field	Format
Token Length	A(2)
Token	A(Token Length)
Frequency	A(2)

(4) Group Index Record

Field	Format
Category number	A(1)
Token number	A(4)

(5) Token Record

Field	Format
Type pointer	A(2)
CC indicator	A(1)(internal representation of B(8))
Token length	A(2)
Token	A(token length)

Note again that in all of these records the internal representation of the fields is preserved in the external representation of the record on a data set. Each of the four files, the Token file, the Type Index file, the Group Index file and the Type Glossary file, is sequential and designed for stream processing. The Token file is in linear text order. The Type Index and Glossary files are in alphabetical order. The Group Index file is in category number order. Each file is prefixed with the header record. Particular DDNAMES have been assigned. These are TOKEN, TYPE, GLOSS, and INDTAB. The name INDTAB is for the Group Index file. The other name associations are obvious.

Storage and Time Estimates

As of this writing the program is in the final testing phase and consequently no exact data are available. However, some remarks can be made. First, the storage requirements were larger then expected. A considerable amount of space is

required for buffers and the sort program if efficient running times are to be achieved. We do not yet know the optimum trade-off point, but it does seem that 44K is appropriate for the sort and that a 7200 byte blocksize is best for fast input/output. This entails the use of approximately 100K bytes.

Preliminary timing estimates show that the program is indeed fast. A text of 175,000 words can be completely processed in less than 25 minutes on a Model 75. A text of 20,000 words was processed in 4 minutes.

4. A Design for a General Statistical Analyzer for Natural Language Texts

by

John B. Smith

If we assume the desirability of making a number of statistical measures over a text of natural language, we are faced with the question of what such an analyzer should look like. In this discussion I shall examine some of the general problems inherent in a procedure of this kind and then suggest what some of the characteristics of a computer program to make these counts should be.

First, we must infer what sorts of counts or measures are likely to be wanted. Many of these will fall into several fundamental categories. These include determination of whether or not a unique string occurs in a text: a count of the number

of times such a string occurs in the text; and perhaps a linear distribution of its occurrences over the text, or certain sub-sections of the text. Next we may want to group several such strings into categories and determine the presence of the category in the text, its frequency, and/or its linear distribution. Also, we may desire individual counts and distributions of the elements of the category, but these statistics represent only repetitions of the first class of measures mentioned. Thirdly, we may wish to determine the existence of particular configurations of categories, the frequency of occurrence of these configurations, and, again, their distributions. This third class of measures may be "simple," applying only to sentences - for example, syntactic analysis; or it may be extensive and attempt to define the entire semantic structure of a text in terms of associations or linkings among a set of categories. No assumptions are made concerning the population over which these measures are taken; therefore, inherent is the possibility of comparing various texts, canons, even languages when such amalgamations are viewed as independent categories. Thus, it should be possible to place most, if not all, statistical measures into this taxonomy. The practical result is that a program that can make the kinds of measurements we have described should have enormous flexibility. The program described below represents one attempt at defining such generality into a workable system.

There are five basic sections of the Analyzer. A driver program allocates resources, determines analytic methods, and

"oversees" the operation of the entire complex. Input and output functions are handled by independent modules. Analysis requests are scanned by a module that establishes logical control vectors to be used by the driver program. Finally, there are a number of analytic modules that perform a variety of functions including making various counts, searching for associative tendencies among elements and categories, and mapping of elements onto thesaural-like semantic structures. Not all of these analytic methods will be available from the start, but because of their modularity they can be added as they are written and as research needs expand and change. For each analysis module, there will be a separate request analyzer subprogram to examine the parameters necessary to define a particular run request. Each of the five major divisions of the program will be described in detail below.

The master control program or driver program manages the individual subroutines and the allocation of resources. Upon initiation of the program, its first task is to pass control to the run request analyzer which either reads a card from the card reader or receives a line of input from a remote terminal. Upon completion of the run request analysis, this program is returned a control vector indicating the various text analytic modules to be used, individual control vectors for them, control information for input data sets to be used, and the particular configuration of output display facilities requested. From a consideration of the number and kind of individual statistical measures to be made, the driver allocates various arrays, structures, banks of counters, etc. as permissible from resources

available. If analysis is to be performed on only a portion of a text or data set, this information is set up in logical form for the input program. Data sets are then opened and control passed to the input routine. After this step, control is passed back and forth between the input routine and the analysis modules until the analysis is complete or the ends of the data sets are reached. At this time the driver again assumes command and establishes any control necessary for post-analysis processing such as sorting, accumulating frequencies from scratch data sets, etc. The Output program then takes control, performs the "housekeeping" functions described above, and prepares the final printed or graphic display of the results of the analysis as well as machine-readable data sets of these values. If the program is being run in the conversational mode, output is directed to the terminal and the driver waits for additional instructions: if not, the driver terminates operation.

The second phase, the run request analyzer, consists of several modules that are brought into operation according to the specific analytic modules called for. In explaining this process, I shall use for an example the request analyzer associated with the COUNT analysis program that compiles a number of counts or distributions over a text. This particular analyzer is called in when the keyword COUNT appears on a run request card. What follows is a series of requests for specific distributions defined in terms of four major kinds of parameters. (In effect, the set of these parameters and the ordering procedure that determines how they can be arranged amounts to a fourth level "user language".)

The first two parameters specify a particular alphabet or set of symbols to be used and the name of a dictionary in which text words exist in terms of the symbols of this alphabet. For example, the alphabet could be the set of phonemes for English and the dictionary, "TEXT.PHONEME," could contain the text expressed in these phonemic symbols. Logically, the dictionary entry could refer to a subprocedure or algorithm that converts the character representation of a word into the appropriate alphabet-symbol representation. More will be said below in the discussion of the Input procedure concerning the assumed data format of the dictionary.

The third level of parameters specifies the linear dimensions, if any, to be used in the analysis. These values are used to develop distributions such as words per sentence, nouns per 500 words of text, etc. Within this level are five specific parameters that are arranged as follows: $n(x_1,y_1)(x_2,y_2)XN$. Starting from the inside and working out, the (x_1,y_1) refers to the upper limit of an index hierarchy for which counts are developed. Assume, for example, that the scheme of INDEX including counts for volume, chapter, paragraph, sentence, and word-in-sentence had been used for hierarchy one; then (1,3) would refer to the paragraph level. If this were the only parameter specified, the program would develop all distributions of the hierarchical scheme below this level: namely, character/word, words/sentence, character/sentence, words/paragraph, sentences/paragraph, etc. If the ordered pair (x_2,y_2) had been specified, then only those counts between the two levels would have been made. For example, (1,3)(1,4) would

develop the distribution for sentences/paragraph only. The symbol X may be used if only a particular count is desired. ((1,4)X would result in a count of words/sentence.) Without the second hierarchical level specified, the basic unit is assumed to be words; however, if the second level is indicated in conjunction with X, that unit is taken as the "atom" for the distribution. Thus (1,2)(1,4)X would result in the distribution of sentences/chapter only.

The last symbol in the sequence, N, is used if counts are to be normalized. If one were studying the distribution of function words over sentences, he would most likely wish to have this distribution normalized between 0 and 1 so that he could determine the relative, not absolute, placement of this category of words. Thus the position in the sentence when such a word appears is divided by the length of the sentence, thereby giving the relative proportion of the way through the sentence -- or normalized distribution -- at the occurrence of these function words. Finally, the first symbol, n, specifies that a group of elements is to be used as the basis of the distribution. If nothing else follows n, the unit is assumed to be words. Thus, "500" would imply that counts are to be made for each 500-word unit of text. 5(1,3) would indicate that hierarchical counts are to be accumulated for each five paragraph blocks of text.

The fourth level of parameters is similar to the third but allows the user to partition his distribution into several separate sets of counts. For example, the user could obtain separate distributions of function words per sentence for

sentences of length 1-10 words, 11-20, 21-30, etc. This facility may reveal important conditional characteristics of certain stylistic features that vary according to particular textual dimensions. The general format of this level of parameters is $m(x_3,y_3)(x_4,y_4)$. As before, the $(x_3,y_3)(x_4,y_4)$ terms specify the units and appropriate level at which these units apply. M specifies the multiple size. Thus 50(1,3) would indicate that paragraph counts are to be broken into distributions based upon 50 word units. Separate distribution would be maintained for paragraphs of fifty words or less, paragraphs of 51-100 words, etc. Similarly, 10(1,3)(1,2) would indicate that separate distributions are to be kept on the basis of the number of sentences per paragraph: i.e., paragraphs of 1-10 sentences, 11-20 sentences, etc.

Obviously, not all of these four levels of parameters will necessarily be used all of the time; and, indeed, not all combinations are meaningful to the analyzer. For example, there can be no fourth level partitioning of counts unless level three is defined for a particular analysis. A rough guide to admissible combinations and the kinds of counts they develop is indicated by the table on the following page in which a "1" indicates that a parameter of some kind is present for that level and a "0" implies the absence of that level parameter. Similar to this kind of organization, additional request analyzers will be developed for other processing modules.

The input phase of the program is kept logically independent

to allow the user to adapt his data sets to the format required for analysis. The program is set up to take as input combinations of the output data sets from the INDEX program described elsewhere in this report. From those data sets, the INPUT subprocedure prepares data lists in fixed formats required by the analysis modules. For example, a distribution of the frequency of occurrence of phonemes over a text would require that the INPUT program first pass to COUNT a type list of phonemic symbols. Then, as the data is read, it would construct lists of text words defined in those same symbols. COUNT would in turn analyze these lists. Similarly, if hierarchical counts are being made, the input program would pass a text-ordered list of logical vectors (00011, etc.) that indicate what positions in the hierarchy are changing with a particular text word or punctuation mark. If the user's data is in a different format from that provided by the INDEX package, he can substitute his own INPUT module to accept his data set and produce the necessary lists of data required by the analysis modules.

The analysis phase of the program is kept modular to allow for growth of the system. Most immediate plans call for the development of the COUNT procedure, discussed briefly above; however, it is likely that after the completion of that procedure a VIA analyzer for semantic associations and a principal component analysis program for developing tendencies among words or groups of words to appear in close proximity in a text or some section of a text will be adapted to this modular design. Since COUNT is the analysis procedure that is most likely to be developed first, more discussion of its operation is in order.

As mentioned in the description of INPUT, COUNT takes as its input an ordered list of data. On that list, according to the control parameters passed to it by the driver program, COUNT accumulates several kinds of statistical measures. First, if an alphabet is specified, it establishes a bank of counters corresponding to that list of symbol-types. Then as the text is passed to it in terms of those symbols, it increments the appropriate counter. For example, if the text word were "æ t", the counter for æ would be incremented by one as would the counter for t. If some type of hierarchical count is being made, COUNT receives a text-ordered list of logical bits that indicate what level of the hierarchy is changing with the current text word. For example, if the text word was a "." that happened to end a paragraph and if the indexing hierarchy established was for volume, chapter, paragraph, sentence, and word in sentence, then the logical vector would be (00111) indicating that word, sentence, and paragraph counts all change with this symbol. COUNT would, in turn, increment the appropriate counters for each of these levels.

The third major function of this procedure is to maintain separate distributions for various textual parameters. This function is handled by outputting the data accumulated into a temporary data set corresponding to the appropriate partition. If, for example, the data is to be partitioned according to paragraph length in multiples of ten sentences, there would be a temporary data set for counts for paragraphs of length 1-10, one for 11-20, etc. It would then be the responsibility of the OUTPUT procedure to accumulate these individual counts and print out or otherwise display the

results.

OUTPUT has four major functions. Its primary responsibility is to display in an efficient and effective format the statistical counts developed by the analysis procedures. This may be in list form or it may be the plotted output of line graphs, histograms, etc. Secondly, it should store these data in a machine-readable form so that they may be used in subsequent analyses. Formats should be written onto the data sets themselves in a form that the INPUT procedure may analyze and use and thus relieve the researcher of many of the headaches of file manipulation and compatibility. Thirdly, in some analyses, temporary data sets were created. In these cases the OUTPUT program will sort these records, if necessary, accumulate counts, and then display and store the results. Finally, in analyses that require additional utilities -- such as principal component analysis -- the OUTPUT procedure will prepare the counts accumulated in a format that can be used by the utility, write this data along with card images of required control statements onto a temporary data set, and then pass control to the system for execution of the utility program.

Because of the modularity of this package, it should provide enormous flexibility for the user. With it, he will be able to accumulate the large number of parametric values necessary to test actively complex models of style and content so that these areas may move from speculation into validated implementations.

VIA

(S/360 PL/I)

User's Manual

by

H. William Buttelmann

John Smith

David Wagner

CONTENTS

0. Introduction

VIA (Verbally Indexed Associations) is a computer programming system to aid the researcher in analyzing natural language text for stylistic patterns. The system is being developed as part of a research project in the techniques of language analysis.[1] The philosophy and development of the VIA system is discussed in detail in the annual reports of this project.

It is the purpose of this manual to provide a succinct collection of the information the user of VIA must have in order to execute and effectively use the system implemented in PL/I for the IBM System/360 family of computers. For more detailed information, the user should consult the project's annual reports. The procedures described in this manual have been used long enough to become fairly static, but the user should keep in mind that the entire system is still being developed. One may wish to contact personally project personnel for recent information.

Input to the system is natural language text coded with a minimum of keying conventions on punched cards or card-image records. The output differs considerably for various parts of the system, but in general it is intended to display information in such a way that stylistic patterning can be indicated.

---

[1]The development of this system is part of a research project under the direction of Professor Sally Yeates Sedelow, Departments of Computer Science and Linguistics, University of Kansas. The research is funded under a grant from the Office of Navel Research.

The system is separated into two phases, VIA1 and VIA2. VIA1 is a data preparation phase, which performs indexing, root-association, and counting functions, and optionally eliminates function words. VIA2 is a collection of analysis packages which use different techniques for analyzing the prepared data.

PART I

1. Introduction

The VIAl package takes as input natural language text with a minimum of conventional symbols, in the form of punched cards or card images on some other medium. Output takes different forms as each program in the package is executed: for example, the first program produces a token data set of logical records containing indexing information (where in the text "this" occurrence of a word is located), and a later program produces a token data set of logical records which optionally provides indexing information and obligatorily groups together by root content words in the text (e.g., mad, madly, and madness would be grouped together and identified by a unique number). These data sets can be used for various forms of statistical and content analyses.

As indicated, the package consists of several independent programs that can be run together as steps of a single job, or they may be run individually as separate jobs. For texts of more than ten thousand words the latter method is recommended to reduce the time required for a single job. This separation of jobs will reduce the possibility of lost effort due to system failure and will also result in faster turn-around. For medium size jobs - 10 to 30 thousand words - it is often possible to combine several steps into a single job and thus reduce the total number of jobs to be run.

The basic programs in the package are INDEX, SORT, PREFIX, and SUFFIX; following SUFFIX, the user may select either the package of programs comprising the ring-structure version of VIA or the list-structure version or the single program MAPTEXT. Part I of the User's Manual describes the basic programs and Part II the alternative analysis packages.

2. Programs

2.1. INDEX

INDEX is a multifaceted indexing program that takes as input 80 character punched cards or card images on some other medium such as magnetic tape. Provisions are made for input from a variety of natural language sources. General processing modes exist for prose, poetry, both verse and prose plays, and transcription of spoken text: there is a special processing mode for long poems, such as Milton's Paradise Lost, which may be divided into books, or cantos, or something analogous, as well as lines--because Paradise Lost is the one such case with which INDEX has dealt, this mode is called MILT. It is assumed that text will be found in columns 1-71 of each card image. A hyphen for the continuation of a word should be placed in column 72. Page numbers may be placed in columns 73-75, and if the user wishes to put sequence numbers on his data cards - in case the deck is dropped - he should do so in columns 76-80. All elements that are to be recognized by the computer as independent must be separated by spaces. This convention implies, in addition to

words, all punctuation marks are to be separated by blanks. For example, the ending of the last sentence would be punched " separated by blanks ." . Textual division and paragraphs in PPOSE or lines in POET are indicated by other conventional markers. These will be discussed below.

Output for all processing modes consists of 56-character logical records containing indexing information and the word or punctuation mark itself. A printed listing of this output may be obtained by putting 'PRINT=YES' as the PARM value on the EXEC card (cf. card 3 of Example B). If no printed listing is desired, use 'PRINT=NO'. If the processing mode is PLAY, stage directions can be included in the data set by using the statement, 'STAGE=YES' in this same statement. 'STAGE=NO' will result in the omissions of these portions of input text. See below for how to indicate stage directions.

PROS is the processing mode for prose text. It will be used as the main example in this discussion. In addition to being blank delimited, as discussed above, the text should have the following markers for major textual division:

Paragraph: end of paragraph is indicated by a double period (..) instead of the single period for the last sentence. Paragraph ending with some other mark of punctuation, such as ? or !, should include the punctuation mark, followed by a space and a double "..".

Chapter: end of chapter is indicated by $$$.

Volume: end of volume is indicated by $$$$.

Page: (Optional) place appropriate number in columns 73-75.

Sequence: (Optional) place appropriate number in columns 76-80.

The index numbers for volume, chapter, and page as well as the sequence number may (at any point of data) be set to begin with a user-specified value by placing in the data set a card with a '$', the appropriate index level and the value to which this counter is to be set. For example, if the user wishes to begin his paragraph counter with paragraph five then he should insert a card with '$PARAGRAPH 0005' on it. Similar updating procedures apply for most processing modes.

The Output record will have the following format:

(Length in bytes)

6	3	3	4	5	3	3	1	8	2	18
LINEAR # IN TEXT	VOL-UME #	CHAP-TER #	PARA-GRAPH #	SEN-TENCE #	WORD IN SENT.	PAGE #	IDIOM	PRE-FIX	LENGTH	WORD

The prefix field is used by the PREFIX program following the sort if prefix information is desired. Otherwise the field may be used by the researcher for categorizing information or for whatever use he may have in subsequent programs. The idiom flag will be set to 1 if the WORD represents a group of words used as a single, idiomatic expression; otherwise, it will be zero (0). This feature is not operative at present. The length field

contains the number of characters in the WORD. Thus, an actual output record might look like this:

34	1	1	1	3	4	12	0	-	2	at

In this example, the word at appears as the 34th word in the text; and it is the fourth word in the third sentence of paragraph one, chapter one, volume one, of the text. Also, it is found on page twelve; it is not an idiom, and it is of length two.

For storage of his output, the user should provide a data set on some medium such as tape or disk with the ddname OUTPUT.

MILT is the special processing mode for Paradise Lost and analogous texts. Input should be blank delimited, one line per card image. If a line is continued from one card to the next, indicate this with an "@" in column 72. Volume numbers should be placed in columns 73-74, and sequence numbers in columns 76-80, if used.

Paragraphs (if used) are indicated by double periods (..).

Books or chapters are indicated by $$$.

Volume is as indicated in columns 73-74.

No update facilities exist for this processing mode. Printout and sequence checking are indicated as in PROS. Output is identical to that from PROS except that volume is usually one, line number replaces sentences, and word in sentence becomes word in line. Again, the user should supply an output data set

for logical output records.

PLAY is the processing mode for prose plays. Printout, and sequence checking are indicated as before. If stage directions are to be included in the data set, use 'STAGE=YES', otherwise 'STAGE=NO'. Regardless of whether or not they are to be included, they should be punched with an asterisk preceding each word; for example, (*lights *dim *slowly *.)

Index information generated is act, scene, paragraph, sentence, word in sentence.

Paragraph is indicated by double periods (..).

Scene is indicated by $$$.

Act is indicated by $$$$.

Act and Scene can be reset by using $ACT ### or $SCENE ###.

Output records are as before except that act replaces chapter and scene replaces paragraph. The user should provide an output data set.

VPLA should be used for plays that are written in verse. Printout and sequence checking should be indicated as described on pages 2 and 3. If stage directions are to be included in the data set then place 'STAGE=YES' on the EXEC card. As was the case for PLAY, stage directions should be punched with an asterisk (*) preceding each such word regardless of whether or not they are to be included in the output data set. Data should be punched with one line of text per card.

Index information generated is volume, act, scene, line, and word in line. Scene and act are indicated as for PLAY. Act and Scene numbers can be reset as in PLAY.

As before, the user should provide an output data set.

POET is the processing mode for poetry. Text should be punched one line of poetry per card. If a line must be continued from one card to another, place an 'at' sign (@) in column 72. If the poetry is stanzaic, then the user must place a number which indicates the stanza in columns 73-75. Sequencing numbers, as before, should be placed in columns 76-80. Indexing information generated will be stanza, line, and word-in-line. No updating facilities exist for this mode. Volume, chapter, and page will all appear with the value one (1) in the output data set. As before, the user should provide an output data set.

SPOK is a processing mode for transcription of oral discourse. Indexing information generated will be for series, session, speaker, sentence, and word in sentence. Series, session, and speaker counters can be reset by inserting a card on which '$', level, and the desired number have been punched. For example, '$SPEAKER 003' would set the speaker counter at an initial value of three.

Within the data itself the following conventions hold:

A change in SPEAKER is indicated by double periods (..).

A change in SESSION is indicated by $$$.

A change in SERIES is indicated by $$$$.

As before, page numbers and sequence number should be placed in columns 73-75 and 76-80 respectively.

2.2. SORT

Both PREFIX and SUFFIX assume input data records that have

been sorted into alphabetical order and which are in the format created by INDEX. Thus the same sort procedure may be used prior to either of these programs. It is recommended that the researcher use one of the utility sorts maintained at his installation. The catalogued procedure set has been defined so as to anticipate 250,000 tokens. If this estimate of the number of tokens is sufficiently wrong, then the user may change that value by redefining the SORT FIELDS which is input to the sort.

2.3. PREFIX

The current version of PREFIX is a revised version of that described in last year's annual report. It takes as input the sorted output of any of the textual modes of the INDEX program. After determining that a word has a legitimate English PREFIX, the program creates a duplicate data record of that word but with the prefix placed in the eight-character PREFIX field included in the format of the INDEX data set. The remainder of the word--without prefix--is then placed in the WORD portion of the record. Thus, the data set is enlarged by duplicate forms of the stems of prefixed words. This data set, if it is to be processed by SUFFIX, must be sorted with the same SORT utility used after INDEX.

2.4. SUFFIX

SUFFIX is a program that groups words having the same root. It does not "strip" the various suffixes from such words but

leaves them as they appear in the text and indicates that they have the same root by assigning to each record for each word-token in a root group the same number, called a matchcount (MATCNT). Thus, for example, all records for complete, completely, and completing, etc. might have the MATCNT, 536. This number has only relative meaning. The next root-group, in alphabetical sequence, would have the MATCNT 537, etc. By this number one can determine words that have the same root form but which differ in ending, thereby facilitating analyses such as semantic and thematic studies which may not involve syntax.

Input is assumed to be 56-character logical records, sorted, from either PREFIX or INDEX. Output will be 69-character records, identical to input records except for the addition of the MATCNT number, the number of word tokens for each MATCH group and the number of tokens for each unique word type. This record has the following format.

(Length in bytes)

6	3	3	4	5	3	3	1	5	4	4	8	2	18
LINE AR#	VOL	CH	PARAGRAPH	SENTENCE	WORD IN SENT	PAGE #	IDIOM FLAG	MATCNT	MATCH-FREQ	TYPE FREQ.	PREFIX	WORD LN	WORD

These records are output in MATCNT order. Under MATCNT, individual records are further ordered according to alphabetical order of word

and then the linear position of each occurrence of the word type in the text.

If the user wishes to eliminate function words from this data set he may do so by using 'FUNCT=NO' on the EXEC card. Similarly, punctuation marks may be deleted by stating 'PUNCT=NO'. Otherwise they will be included, with all function words grouped together with a MATCNT of 99998 and all punctuation marks with 99999. Unless 'PRINT=NO' is included, a printed copy of the output will be produced.

As before, the user must supply an Output data set on some medium. It is mandatory that the user include a heading to be printed at the top of each page of printed output. This heading may identify the particular text being processed or contain any other information that the user wishes to include. It should be punched on a single card which is placed after the card '//S.HEADING DD *' .

3. Activation of VIA1

The standard operation of VIA1 uses INDEX, SORT and SUFFIX. The JCL to execute this is:

```
//  (JOB card)
//JOBLIB       DD   DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//             EXEC VIA1
//I.INPUT      DD   (DD information for input to INDEX)
//S.OUTPUT     DD   (DD information for output from SUFFIX)
//S.HEADING    DD   *
```

If the user wishes to include PREFIX in the VIA1 data preparation, he should use the following JCL:

```
//  (JOB card)
//JOBLIB       DD   . . .
//             EXEC VIA1,PREFIX=EVEN
//I.INPUT      DD   . . .
//S.OUTPUT     DD   . . .
//S.HEADING    DD     *
```

To provide run time parameters to INDEX, the EXEC card should be modified to read as follows:

```
//             EXEC VIA1,PARM.I='(INDEX parameters)'
```

or

```
//             EXEC VIA1,PREFIX=EVEN,PARM.I='(INDEX parameters)'
```

If it is desired to save the output data set from INDEX, the following DD card should also be included:

```
//I.OUTPUT     DD   (DD information for INDEX output)
```

To execute only INDEX and save the output, use the following JCL:

```
//  (JOB card)
//JOBLIB       DD   . . .
//             EXEC VIA1,PARM.I='. . .',ONLYI=ONLY
//I.OUTPUT     DD   . . .
//I.INPUT      DD   . . .
```

Example A.

In this example, the standard VIA1 (without PREFIX) is executed. In the run time parameters to INDEX, PROS is specified as the processing mode and a printed copy of the output is requested. The input text is in the DD* card deck following the JCL.

```
//  (JOB card)
//JOBLIB       DD   DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//             EXEC VIA1,PARM.I='PROC=PROS,PRINT=YES'
//I.INPUT      DD   *
   (input text cards)
/*
//S.OUTPUT     DD   (DD information for output)
//S.HEADING    DD   *
    (heading card)
/*
```

Example B.

The same run as that of Example A, but including the use of PREFIX, would require the following JCL:

```
//  (JOB card)
//JOBLIB       DD   DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//            EXEC VIA1,PREFIX=EVEN,PARM.I='PROC=PROS,PRINT=YES'
//I.INPUT      DD   *
  (input text cards)
//S.OUTPUT     DD   (DD information for output)

//S.HEADING    DD   *
  (heading card)
/*
```

PART II

4. Introduction

VIA2 is composed of the three independent procedures MAPTEXT, LIST, and RING, and the utility procedures BLDLIST and BLDRING. In addition there is a subroutine mechanism, called SELECT, which occurs in several of the procedures and is documented separately.

5. SELECT

5.1. Function

SELECT is the mechanism employed by the VIA2 subsequences MAPTEXT, LIST and RING to select those sections of text to be processed by the respective programs. SELECT is used only after the text has been processed by INDEX so that associated with each token of the text is the indexing quadruple I = (V,C,P,S). The interpretation of I is dependent upon which processing option

(PROC) was chosen for INDEX and could be:

```
 i).  if PROC = PROS then V = Volume
                          C = Chapter
                          P = Paragraph
                          S = Sentence
ii).  if PROC = MILT then V = Volume (usually 1)
                          C = Chapter or Books
                          P = Paragraph
                          S = Sentence
iii).  if PROC = PLAY then V = Act
                          C = Scene
                          P = Paragraph
                          S = Sentence
 iv).  if PROC = VPLA then V = Volume
                          C = Act
                          P = Scene
                          S = Line
  v).  if PROC = POET then V = Volume
                          C = Chapter
                          P = Stanza
                          S = Line
 vi).  if PROC = SPOK then V = Series
                          C = Session
                          P = Speaker
                          S = Sentence
```

For example, if the text is an instance of oral discourse and hence INDEX was run in the SPOK mode, then the level P would correspond to the Speaker index.

The user must define those intervals of text to be processed as well as specify the level of indexing to be consulted for the text selection. Because the indexing is hierarchical to a depth of 4 the user must include the specification of all levels which cover the level of interest.

The specification language enables the user to select either a single section of text or else a connected sequence of sections of text or else combinations of the above two. The single section of text is selected by specifying to the required level the particular values that the indexing parameters must achieve in

order for that section to be processed. The collection of indices must be enclosed by parentheses and separated by commas. For example, a single sentence could be isolated by the entry (1,3,4,5) which specifies volume 1, chapter 3, paragraph 4, sentence 5. Likewise a single chapter could be isolated by the entry (3,2) which specifies volume 3, chapter 2.

The specification language contains constructs with which to specify a connected sequence of text sections. The form of this type of specification is (s1, s2) where both s1 and s2 are specifications for single sections of text. The interpretation is to include that portion of text which lies inclusively between the text whose indexing parameters are s1 and the text whose indexing parameters are s2.

A complete specification is then a sequence of comma-separated entries of either type with the requirement that each entry be to the same level of specificity and that the level be that of the LEVEL parameter of the EXEC card. Furthermore, the entries must begin in column 1 of a data card and if a continuation beyond column 71 is required, then the program expects an * to be punched in column 72.

5.2. Activation

This mechanism is activated by supplying the processing program with a run time parameter by:

```
                MAPTEXT=EVEN,PARM.M='LEVEL=*,...'
// EXEC VIA2,RING=EVEN,PARM.R='LEVEL=*,...'
                LIST=EVEN,PARM.L='LEVEL=*,...'
```

where * is to be one of

A to process the entire text
V to process on the volume level
C to process on the chapter level
P to process on the paragraph level
S to process on the sentence level

The following three examples should make clear the nature of the language and the activation of the mechanism.

Example from MAPTEXT:

To specify that the selection is to be on the paragraph level and that the text is to be selected from volume 2, chapter 3, paragraphs 7 - 12; volume 3, chapter 4, paragraph 6; and volume 7, chapter 2, paragraphs 3-33:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC VIA2,MAPTEXT=EVEN,PARM.M='LEVEL=P,...'
//M.SYSIN DD *
((2,3,7),(2,3,12)),(3,4,6),                                    *
((7,2,3),(7,2,33))
    (remaining M.SYSIN cards for MAPTEXT)
/*
```

Example from LIST:

To specify that the entire text is to be processed:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC VIA2,LIST=EVEN,PARM.L='LEVEL=A'
```

Example from RING:

To request that volumes 1,5,7 and 9-11 be selected for processing:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC VIA2,RING=EVEN,PARM.R='LEVEL=V,...'
//R.SYSIN DD  *
(1),(5),(7),((9),(11))
(remaining R.SYSIN cards for RING)
/*
```

5.3. Notes

The text selection always assumes that in the case where LEVEL is not defined on the EXEC card that its value will be A, and hence will process the entire text and will not expect to find additional input on the SYSIN dataset.

The mechanism contains an internal sort, so that the specifications may be entered in any order. However, whenever a request is of the form (s1,s2) then the indexing parameters specified by s1 must occur in the text prior to the indexing parameters specified by s2.

6. MAPTEXT

6.1. Function

MAPTEXT is a VIA2 program which provides a mechanism for isolating stylistic patterns within a text by printing special graphics for designated keywords. MAPTEXT scans a user-supplied input text and substitutes for each occurrence of a user-defined keyword its respective single replacement character and substitutes a period (.) for all other words. The resulting print-out therefore consists of lines of periods with single replacement characters interspersed, thereby generating a

visual representation of possible stylistic patterns. MAPTEXT aligns the printed output so that sentences always begin a new line, and it provides indexing information at the extreme right of each printed line.

6.2. Options

Because one may find it to be preferable to define stylistic patterns as a function of root group distribution rather than as a function of type distribution, MAPTEXT provides an option which implements this facility. If this option is selected (by specifying ROOT_GROUP=YES as the PARM.M value on the EXEC card) all of the words in a root group (i.e., with the same match count number) form an equivalence class and the replacement action applies to every word in the class.

MAPTEXT provides for the activation of text selection as documented in SELECT. The input (on dataset M.SYSIN) to the text selection mechanism must preceed that defining the pair (replacement character, word).

6.3. Activation

MAPTEXT, being a subsequence from the VIA2 catalogued procedure, is activated by the JCL:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC  VIA2,MAPTEXT=EVEN
//M.INPUT DD (DD information for the output dataset from
             INDEX)
//M.SYSIN DD *
 (parameter cards for SELECT, if any)
C1   W1
C2   W2
 :    :
Cn   Wn
/*
```

where each C_i is a replacement character for the word W_i.

As stated earlier, this module has two options which are controlled by the variables ROOT_GROUP and LEVEL whose values are transmitted by the PARM.M field on the EXEC card. The default values (ROOT_GROUP=NO,LEVEL=A) will result in the entire text being processed and in performing the character substitutions on the basis of exact token matching.

MAPTEXT expects that the M.SYSIN dataset will be defined and will contain:

i). the specification of the text intervals to be processed if the level value is not A.

ii). a list of pairs (C,W) where C is the (single) replacement character for the word W (whose length is no greater than 18 characters). Each C is to be punched in column 1 of a data card and its corresponding W is to begin in column 5 of the same card.

Therefore the following JCL and data will result in the substitution of W for all occurrences of WAR and P for all occurrences of PEACE in the entire text:

```
//  (JOB card)
//JOBLIB  DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC  VIA2,MAPTEXT=EVEN
//M.INPUT DD  (DD information for the output dataset from
              INDEX)
//M.SYSIN DD  *
W     WAR
P     PEACE
/*
```

The JCL and data provided below will result in MAPTEXT substituting P for all occurrences of the root group of PEACE (PEACE,PEACEABLE,PEACEFUL, etc.) and W for all occurrences of

the root group of WAR (WAR, WARRED, WARRING, etc.) in only chapter 1 of volume 1 of the input text:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC  VIA2,MAPTEXT=EVEN,PARM.M='ROOT_GROUP=YES,LEVEL=C'
//M.INPUT DD (DD information for the output dataset from
              INDEX)
//M.SYSIN DD  *
(1,1)
W    WAR
P    PEACE
/*
```

6.4. Notes

The VIA1 catalogued procedure has been defined so that the output dataset from INDEX is not saved and hence the user of MAPTEXT must provide a JCL override to define I.OUTPUT in such a way as to make it available as the INPUT dataset to MAPTEXT. This override to VIA1 is effected by supplying the card:

```
//I.OUTPUT  DD (DD information for the output dataset from
                INDEX with LRECL=56)
```

7. LIST

7.1. Function

LIST is the VIA2 subsequence which prints out a text specific thesaurus from a given text and a given list thesaurus. The list thesaurus takes the form of a list of ordered pairs (primary word, secondary word). More specifically, from a thesaural tree defined to a maximum depth of five described as a list thesaurus, these programs prune the tree so that the label of each node either occurs in the text or a word with the same match count as the label occurs in the text. Consider

the following example:

Suppose we had the thesaural tree (to a depth of 5), where A,B,C,D,E,F are all words not necessarily in the input text

A

B
C
D

E
F

A

B
C
D

This is the tree that would result from the list thesaurus of ordered pairs:

(A,B) (A,C) (A,D) (D,E) (D,F) (F,A)

Now suppose that the text contained no word with the same match count as that of C or F but it did contain words with the same match count as A,B,D,E; the tree which would result would then be:

A

B
D

E.

There are two avenues available to the researcher who uses LIST. He may either prepare his own list thesaurus of ordered pairs or he may supply just a list of primary words and enable the sequence of programs (see the section entitled BLDLIST)

which will extract the list thesaurus from this list of primary words by using a ring thesaurus which is stored on disk. In either case the list of thesaural associations must be ordered so that the concatenations of primary word with secondary word forms an alphabetically ordered list.

7.2. Options

LIST provides for the activation of text selection as documented in SELECT.

7.3. Activation

LIST, being a subsequence from the VIA2 catalogued procedure, is activated by the JCL (which requests processing of the entire input text because PARM.L='A'.)

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC  VIA2,LIST=EVEN,PARM.L='A'
//L.INPUT1  DD (DD information for the list thesaurus)
//L.INPUT2  DD (DD information for the output dataset
                from SUFFIX)
```

If the researcher wished to process with LIST only paragraph 17 of chapter 12 of volume 3 then he should code:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC  VIA2,LIST=EVEN,PARM.L='P'
//L.INPUT1 DD (DD information for the list thesaurus)
//L.INPUT2 DD (DD information for the output dataset
               from SUFFIX)
//L.SYSIN DD *
(3,12,17)
```

7.4. Notes

It is important to observe that a subtree will sprout only from those nodes whose labels are primary words so that

secondary branching will occur only in the case that a secondary word is also a primary word (as in the case of F and A from the introductory example). Therefore if the list of ordered pairs which makes up the list thesaurus is not sufficiently rich to include several intersections of the secondary word set with the primary word set, the resulting output may be somewhat trivial.

8. BLDLIST

8.1. Function

BLDLIST is the VIA2 subsequence which generates a list of ordered pairs of the form (primary word, secondary word) from a user-supplied list of primary words such that one ordered pair is created for each secondary word that occurs in a thesaural category (currently we are using Roget's International Thesaurus) with a primary word. Consequently, if the stored thesaurus was:

Category 1: A,B,C
Category 2: C,D,E
Category 3: E,F.

and if C was the single user-supplied primary word, then the list thesaurus would be:

(C,A)
(C,B)
(C,C)
(C,D)
(C,E)

The list is subsequently sorted so that the concatenates (primary word, secondary word) form an alphabetically ordered sequence.

The typical usage would request the execution of LIST immediately following the execution of BLDLIST, but this sequencing is not mandatory; consequently, this mechanism can be used to generate a list thesaurus independently of the LIST processing system.

8.2. Activation

BLDLIST, being a subsequence of VIA2, is activated by the JCL:

```
//  (JOB card)
//JOBLIB  DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC VIA2,BLDLIST=EVEN
//BL.SYSIN DD *
(list of primary words)
/*
```

The list of primary words must be such that one word is punched per card with the word beginning in column 1 and having no more than 18 characters.

8.3. Notes

If the user needs to save the list of ordered pairs generated by BLDLIST then he should code:

```
//BL.OUTPUT DD (DD information for the list of ordered
                pairs with LRECL=40)
```

Typically the user will follow BLDLIST with LIST. The VIA2 catalogued procedure is defined in such a way that L.INPUT1 is the same dataset as BL.OUTPUT, so that the activation of the sequence BLDLIST, LIST would become:

```
//  (JOB card)
//JOBLIB DD DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//  EXEC VIA2,BLDLIST=EVEN
//BL.SYSIN DD *
(list of primary words)
//  EXEC VIA2,LIST=EVEN
//L.INPUT2  DD (DD information for the output dataset
                from SUFFIX)
```

9. RING

9.1. Function

RING is the package of VIA2 programs which uses a given ring-structured thesaurus to identify verbal associations in or with a text. The text must first be processed by VIA1. The thesaurus may be provided in either of two ways: the user may use the copy of Roget's International Thesaurus[2] which is provided in ring-structure format as part of the VIA system, or the user may supply his own specialized thesaurus using the package of programs BLDRING (cf. Section 10 of this manual).

The function of RING is to compare the text (or a section of text) with the thesaurus and, choosing certain elements as key linguistic units, to search the thesaurus for words or phrases associated with each key. A printout is produced which represents in graphic form the relationships identified by the thesaurus search. The choice of the key search elements,

---

[2] Permission to convert Roget's International Thesaurus into machine readable form and to use it in this research was given by Thomas Y. Crowell Co., publishers.

the depth to which the search in the thesaurus is allowed to wander, and various search pattern techniques, are under the control of the user.

9.2. Usage Considerations; Data Sets Provided by the User

To execute RING the user must provide the text, the initial thesaurus, text selection parameters for SELECT (if any), a TEXTSECT card, and ANALYSIS request cards. This section contains the information sufficient to enable the user to provide these data and to understand the use of RING. A summary of the data sets provided by the user is given in Section 9.2.6.

9.2.1. Text, Text Selection Parameters, and TEXTSECT card

The entire text should first be processed by VIA1. Even if the user intends to analyze the text in sections using multiple runs of RING -- or other program packages of VIA2 -- the whole text should be processed once by VIA1. This will ensure that global textual information will be correct, and will also provide a less expensive "rerun point" for analysis runs. The primary output text of VIA1,S.OUTPUT, is the dataset to be used for R.INPUT.

The subprogram called SELECT (cf. Section 5) is incorporated in RING. The parameter cards for SELECT are the first cards in the R.SYSIN data set. The entire collection of text extracted for one run is called a "text section", whether or not it is contiguous. By means of a TEXTSECT card, which follows the selection parameter cards in the R.SYSIN data set, the text

section is assigned a number. The TEXTSECT card has the following format:

TEXTSECT=n[,MSGPARM='LIST'];[character string]

n is a decimal integer which is the number of the current section of text. The optional parameter MSGPARM='LIST' requests the printing of warning messages produced by RING during the text-thesaurus comparison phase. Generally, these messages note garbled text, the inability of the program to establish any associations with a textual word not in the thesaurus, or a lack of sufficient space in the scratch-pad area of the thesaurus (cf. Section 9.2.2). The default for this parameter is 'NOLIST', which suppresses message printing. A count of the messages that would have been printed is given. The optional character string following the semicolon will be printed as a heading at the top of the first page of output. Examples:

TEXTSECT=1,MSGPARM='LIST';FROST - ENTIRE TEXT

TEXTSECT=2;SECTION 2 OF JOYCE: PORTRAIT. 5/9/70

If several runs of RING are made on different text sections the TEXTSECT numbers provide a means of identifying each section to the program, which retains certain text-specific information by section. TEXTSECT numbers must be assigned sequentially in order of processing. If only one run is to be made on a text, the TEXTSECT number should be 1. After selection, the text section is condensed into a type data set.

9.2.2. Thesaurus

The thesaurus is composed of the four data sets RT.THSCTL, RT.VOCAB, RT.DRCTRY, and RT.THES. Roget's International Thesaurus is provided by default. Alternatively, the user may provide his own thesaurus. To do so, the VIA2 utility procedure BLDRING (Section 10) should be used to construct the four data sets from a thesaurus prepared by the user. This work should be completed before RING is executed. In the subsequent execution of RING, overriding DD cards for the user's own thesaurus must be included in the JCL.

As the text section is compared with the thesaurus, certain text-specific information is recorded in a "scratch-pad" area in the VOCAB data set of the thesaurus. This information includes frequency in the text section and the earliest TEXTSECT number of the sections in which the word appears. Also entered are words from the text not in the thesaurus which have a root identical to some word already in the thesaurus. To facilitate reruns, prior to processing, the scratch-pad information is examined and only information related to text sections with numbers less than the current TEXTSECT number is retained. For this reason the sequential assignment of TEXTSECT numbers is essential. Since the smallest permissible TEXTSECT number is 1, using that number will ensure a clean scratch-pad area, and it should always be used for the first section of text processing.

If two or more users are concurrently using the same thesaurus, it is necessary only for each user to keep his own private copy of the RT.VOCAB data set, since the other data sets

of the theasurus are never modified.

9.2.3. Analysis and ANALYSIS cards

The primary output of RING is a printout arranged so as to indicate the patterns of verbal relationships defined by the thesaurus. The data are displayed in the form of a tree-structured graph, where the nodes of the tree are verbal entities (words and phrases) and thesaural category designators. A separate tree is printed for each key search element, with the key element as the root of the tree. Each path in the tree represents a path in the search of the thesaurus. This method of displaying the results of the thesaurus search makes it possible to indicate a sense of the "closeness" of semantic association and to indicate indirect associations induced by some common direct association: two elements are directly associated if they appear as adjacent nodes on some path in the tree.

The tree is printed with the root at the top left of a page, branching to the right, with each level of the tree a different column on the page. The column immediately to the right of the root contains the labels of all the categories containing the root. In the next column to the right is printed, beneath each category, the remaining words and phrases in that category. This column then, contains all verbal elements directly related to the root, and the intervening category labels show in what sense the relation stands. This process is repeated with each new word and phrase as the root of its subtree. Each successive column of

categories and words represent s another level in the depth of the thesaurus search and thus another level remoteness from the root.

As an example consider the following thesaurus:

CATGRY 1: AUTHENTIC, AUTHORITATIVE, ORTHODOX
CATGRY 2: CANONICAL, CATHOLIC, ORTHODOX
CATGRY 3: AUTHENTIC, GENUINE, BONE FIDE
CATGRY 4: AUTHORITATIVE, DOGMATIC, OPINIONATED

A search keyed on AUTHENTIC would produce the following tree:

AUTHENTIC

CATGRY 1

AUTHORITATIVE

CATGRY 4

DOGMATIC
OPINIONATED

ORTHODOX

CATGRY 2

CANONICAL
CATHOLIC

CATGRY 3

BONA FIDE
GENUINE

A variety of methods for choosing search keys and a variety of modes for searching the thesaurus are available. For example, the user may specify a particular word as a search key, or he may instruct the program to make every word that occurs in the text a number of times exceeding a user-defined threshold a search key. He may choose search keys based on word root or thesaurus category membership. The user may specify that thesaurus searches be limited to words in the text, he may permit words not in the

text, or he may choose one of several compromises between these extremes.

Parameters for controlling the choice of search keys and the modes of searching the thesaurus are contained in the ANALYSIS cards entered in the R.SYSIN data set following the TEXTSECT card. Each Analysis card causes the selection of one or more search keys and each search key causes the initiation of a thesaurus search which will result in the printing of one tree. Up to 100 ANALYSIS cards is permitted per run. Following the last ANALYSIS card a card with the words GO AHEAD must appear in the R.SYSIN data set. If it is missing, RING will terminate after updating the scratch-pad area of the thesaurus and generating the search keys. Eliminating the GO AHEAD card thus provides an economical means of rerunning to update the thesaurus and search key file, without initiating thesaurus searches.

The format of ANALYSIS cards is as follows:

ANALYSIS n,TYPE='m'[,optional parameter list];

n is an arbitrary decimal integer identifying the analysis and will be used to identify all printout associated with this analysis. m is the number 1, 2, 3, or 4. This parameter is used to designate the procedure for selecting search keys. The remaining parameters are MODE, THRESHOLD, CAT, WORD, DEPTH, and KEYLIST. The four types of key generation are:

TYPE='1' - Frequent categories. Every category occurring in the text more than a specified number of times is to be used to key a search. The number of times to be used

as a threshold must be specified by a THRESHOLD parameter in the same card. The program will sum the number of occurrences in the text of every word in each thesaurus category. If a word occurs in more than one category, its total will be added to each. Every category whose total number of occurrences is equal to or greater than the threshold, will be used to key a search. This type of analysis is lengthy, but enables the system to choose significant content in the text, even though it is not identified by the high frequency of any particular word, because the significance is based on the high occurrence of categories.

TYPE='2' - Frequent roots. Every root occurring in the text more than a specified number of times is to be used to key a search. The number of times to be used as the threshold must be specified in the THRESHOLD parameter. The program will total the number of occurrences of every MATCNT in the text. If the total for a MATCNT is equal to or greater than the threshold, every word in the MATCNT will be used to key a search. This is done by generating a KEY entry for each word in the MATCNT. This type of analysis is somewhat faster than type 1. The system chooses significant content in the text based on the frequency of word roots.

TYPE='3' - Category. The category must be specified by a CAT parameter in the same card. It will be used to key a search. No count considerations are used. This type of analysis is much faster than the previous types and is useful for searching for relationships to a particular category, however obscure.

TYPE='4' - Word. A particular word must be specified by a WORD parameter. It will be used to key a search. This type of analysis is as fast as TYPE 3, and is useful for searching for relationships to a particular word, however obscure. The word need not be in the text, but if not, must be in the thesaurus. Thus, for example, this type of analysis, combined with search mode D, may be used to find all words in the text related to that on the parameter card.

The MODE parameter:

MODE = 'x'

specifies the mode of thesaurus search used in the SPRINT program. x must be one of the letters A, B, C, D, or E. If the MODE parameter is omitted or incorrectly specified, mode A will be taken by default and a message printed to that effect. The modes are best described in terms of the tree-shaped output showing the word and category relationships. The five modes are:

MODE = 'A' - Text limited. All nodes are in the text. As the program searches down a path of related words in the thesaurus, it will abandon that path as soon

as it encounters a word not in the text. The path down to that point will be printed.

MODE = 'B' - Text oriented. Root and leaves are in the text. If the root is not in the text, nothing is printed. If it is, each path is pursued until no new words can be found that are in the text. The path is printed up through the last textual word encountered. Thus, this word becomes a leaf of the tree. Intermediate nodes may or may not be in the text.

MODE = 'C' - Text rooted. Root in the text. Only the root is required to be in the text. Each path is printed down through the number of levels specified by the DEPTH parameter. This kind of search is used to look for words, in or out of the text, that are related to the root, a word or category in the text.

MODE = 'D' - Text related. Leaves in the text. This mode is similar to mode B, except that the root is not required to be in the text. This kind of search is used to look for words in the text related to a given word or category, whether or not it is in the text.

MODE = 'E' - Subthesaurus. The entire subthesaurus rooted at the key is printed, down through the depth specified in the DEPTH parameter. Nothing is required to be in the text.

The THRESHOLD parameter:

THRESHOLD = 'n'

is only used in type 1 and type 2 analyses. n specifies the frequency threshold.

The CAT parameter:

CAT = 'category'

is only used in type 3 analyses. Every word in the category specified becomes a search key.

The WORD parameter:

WORD = 'word'

is only used in type 4 analyses. The word specified becomes a search key whether or not is is in the text. The word must be in the thesaurus, or no relationships will be found and no output will appear.

The DEPTH parameter:

DEPTH = 'n'

specifies the depth of search that is to be made in the thesaurus. n must be an integer 1 through 9. Search depths greater than 9 will be reduced to 9 by the syntax checking routines of RING and a message to that effect will be printed. If the DEPTH parameter is omitted or incorrectly specified, a depth of 3 will be taken by default and a message to that effect will be printed.

The KEYLIST parameter:

KEYLIST = 'option'

is used to specify whether a listing of the keywords for this analysis is requested. The options are LIST and NOLIST. NOLIST is default. If LIST is specified, RING will print a listing of all keys generated for the ANALYSIS request.

Examples of ANALYSIS request cards:

```
ANALYSIS 1, TYPE ='2',MODE = 'D', THRESHOLD = '100',
                        DEPTH = '4';
ANALYSIS 2, TYPE = '1', THRESHOLD = '50';
ANALYSIS 3, TYPE = '4', MODE = 'D', DEPTH = '9',
                    KEYLIST = 'LIST', WORD = 'MIND';
ANALYSIS 4, TYPE = '3', CAT = '100.1';
```

For ANALYSIS 1, search keys will all be words whose roots occur 100 times or more in the text section. Thesaurus searching is limited to a depth of 4, and the tree printout is to be pruned so that all leaves of the tree are in the text. The search key and intermediate nodes need not be in the text.

For ANALYSIS 2, search keys are the words of each category that appears 50 times or more in the text section. Thesaurus searching is limited to a depth of 3 by default, and the default search mode of 'A' means that only words in the text section will be printed.

For ANALYSIS 3, the word MIND is to be the only search key. Thesaurus searching will go to the maximum depth permitted, 9 levels, and the search mode is the same as the mode of ANALYSIS 1. The printed tree will show all words in the text associated with the word MIND, and it will show all such associations. A copy of the search key record will be printed, as requested by the KEYLIST parameter.

For ANALYSIS 4, search keys will be all words in Category 100.1. The search depth is 3 by default and the search mode 'A' by default, meaning that only words in the text will be printed.

9.2.4. Search Keys for a Sequence of Runs

When a sequence of runs is made on different sections of text, a permanent file of the search keys is created by merging the new keys generated for each section with the keys from previous sections. For this purpose the data set RK.NEWKEYS should be saved after each run. The user accomplishes this by overriding the DD card for the RK.NEWKEYS file. The data set RK.NEWKEYS of a run is a new input data set RK.OLDKEYS to the next run in the sequence. An example is provided in the section on JCL.

9.2.5. Printing Conventions

The following printing conventions are used in the printout trees. All words not in the text are enclosed in parentheses. If a word is occurring for the first time in the current section of text, it is preceded by a dash line. If the method of saving search keys described in Section 9.2.4 is used, roots which qualified as search keys in some earlier section of text but do not qualified as keys in the current section, are nevertheless used to generate a thesaurus search; the root is marked with an asterisk. If the root also qualifies as a search key in the current text section it is marked with a period instead. To reduce the amount of output, when a category appears more than once at the deepest level of a tree, its words are printed only the first time it appears; thereafter, only the category designator is printed. At the conclusion of each tree printout, a summary of all categories appearing in the tree is provided.

9.2.6. Summary of Data Sets

Required:

R.INPUT - The input text, previous output from VIA1 (S.OUTPUT)
R.SYSIN - Parameter cards for SELECT
TEXTSECT card
ANALYSIS cards
GO AHEAD card

Optional: (use to define private thesaurus)

RT.THSCTL
RT.VOCAB
RT.DRCTRY thesaurus
RT.THES

Optional: (use when search keys are to be recirculated with successive runs)

RK.OLDKEYS - Search keys output from previous run
RK.NEWKEYS - Search keys output from current run

9.3. Activation

The usual JCL to execute RING using the standard system thesaurus. Search keys are not saved.

```
//  (JOB card)
//JOBLIB    DD   DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//          EXEC VIA2,RING=EVEN,PARM.R='(SELECT parameter)'
//R.INPUT   DD   (DD information for S.OUTPUT)
//R.SYSIN   DD   *
   (parameter cards for SELECT)
   (TEXTSECT card)
   (ANALYSIS cards)
   (GO AHEAD card)
/*
```

The JCL to execute RING using a private thesaurus supplied by the user. There is an input file of search keys and the updated file of search keys is saved:

```
//  (JOB card)
//JOBLIB       DD   . . .
//             EXEC . . .
//R.INPUT      DD   . . .
//R.SYSIN      DD   *
                  .
                  .
                  .
/*
//RT.THSCTL    DD   (DD information for BR.THSCTL)
//RT.VOCAB     DD   (DD information for BR.VOCAB)
//RT.CRCTRY    DD   (DD information for BR.DRCTRY)
//RT.THES      DD   (DD information for BR.THES)
//RK.NEWKEYS   DD   (DD information for new file of keys)
//RK.OLDKEYS   DD   (DD information for old file of keys)
```

The following example shows JCL for a sequence of three runs on three sections of text. The systems thesaurus is used. The file of search keys is recirculated, but not kept after the final run. In this example each run is a separate job, but they could be steps of one job.

```
//RUN1  (JOB card)
//JOBLIB       DD   . . .
//             EXEC . . .
//R.INPUT      DD   . . .
//R.SYSIN      DD   *
  (parameter cards for SELECT)
  TEXTSECT=1;
  (ANALYSIS cards)
  GO AHEAD
/*
//RK.NEWKEYS   DD  DSN=(user's data set # 1). . .
//RUN2 (JOB card)
//JOBLIB       DD   . . .
//             EXEC . . .
//R.INPUT      DD   . . .
//R.SYSIN      DD   *
  (parameter cards for SELECT)
  TEXTSECT=2;
  (ANALYSIS cards)
  GO AHEAD
/*
//RK.NEWKEYS   DD  DSN=(user's data set # 2). . .
//RK.OLDKEYS   DD  DSN=(user's data set # 1). . .
```

```
//RUN3  (JOB card)
//JOBLIB          DD   . . .
//                EXEC . . .
//R.INPUT         DD   . . .
//R.SYSIN         DD   *
  (parameter cards for SELECT)
  TEXTSECT=3;
  (ANALYSIS cards)
  GO AHEAD
/*
//RK.OLDKEYS      DD   DSN=(user's data set # 2). . .
```

9.4. Individual Program Functions

RING consists of the following sequence of steps in the cataloged procedure VIA2:

```
R    -  PL/I program PRERING
PRS  -  system sort
RT   -  PL/I program THESAUR
RS   -  system sort
RK   -  PL/I program KEYUP
RST  -  PL/I program SPRINT
```

PRERING is composed of the SELECT function (cf. Section 5) and a set of code which condenses the selected token data set into a type data set. The first system sort sorts the output text section into alphabetical order on type.

The remaining steps are documented in detail in the 1968-1969 annual report of this research project. We provide a brief description of the function of each here. THESAUR reads the ANALYSIS cards, generates the search keys, and performs the text-thesaurus comparison. The second sort sorts the search keys. KEYUP merges the new search keys with any previous keys and marks special keys. SPRINT reads the search keys, searches the annotated thesaurus, and prints the resulting trees.

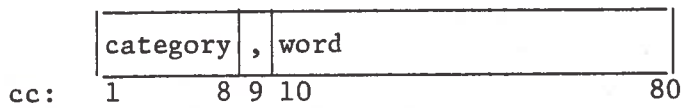
10. BLDRING

10.1. Function

BLDRING is the sequence of VIA2 programs used to create the four data sets of a ring-structured thesaurus from a thesaurus prepared by the user. The user provides the input thesaurus as a list of ordered pairs and the storage space and DD cards for the output thesaurus data sets.

10.2. Input Data

The input thesaurus is a file of 80-character records of the form:



Before being input to BLDRING, the file should be sorted into index order -- that is, with word as major sort field and category as minor sort field. This data set is the file BR.ORGNX.

For example, the sample thesaurus of Section 9.2.3 would be entered as follows:

```
CATGRY 1,AUTHENTIC
CATGRY 2,AUTHENTIC
CATGRY 1,AUTHORITATIVE
CATGRY 4,AUTHORITATIVE
CATGRY 4,BONA FIDE
CATGRY 3,CANONICAL
CATGRY 2,CATHOLIC
CATGRY 4,DOGMATIC
CATGRY 3,GENUINE
CATGRY 4,OPINIONATED
CATGRY 1,ORTHODOX
CATGRY 2,ORTHODOX
```

In addition to the thesaurus, the user must enter a card in the BR.SYSIN file containing the block sizes of the three data sets

BR.VOCAB, BR.DRCTRY, and BR.THES. This card is the only card in the BR.SYSIN file. Its format is:

VBLKSIZE=n_1,DBLKSIZE=n_2,TBLKSIZE=n_3;

where n_1 is the block size of BR.VOCAB and must be a multiple of 36,

n_2 is the block size of BR.DRCTRY and must be a multiple of 16,

and n_3 is the block size of BR.THES and must be a multiple of 8. These numbers must agree with the BLKSIZE figures in the DD cards of their respective files. A recommended choice for all three block sizes is the approximate size of one disc track, subject to the factor constant. On the IBM Model 2314 disk, one could specify VBLKSIZE=7272, DRLKSIZE=7280 and TBLKSIZE=7288.

10.3. Efficiency Considerations

If the thesaurus is very large (above 10,000 entries) or if the user plans a significant amount of computing with it, he should investigate the issue of optimum block sizes for the thesaurus. The utility programs VBLKSIZE and OPTBS are available for this purpose. These programs are documented in the annual report of this project for the year 1968-1969. This problem has been solved for the standard system thesaurus.

10.4. Activation

The JCL necessary to activate BLDRING is the following:

```
//  (JOB card)
//JOBLIB       DD  DSN=UNC.IS.F2312.SEDELOW.VIALIB,DISP=SHR
//             EXEC VIA2,BLDRING=EVEN
//BR.ORGNX     DD  (DD information for input thesaurus)
//BR.THSCTL    DD  (DD information for output thesaurus)
//BR.VOCAB     DD  (DD information for output thesaurus)
//BR.DRCTRY    DD  (DD information for output thesaurus)
//BR.THES      DD  (DD information for output thesaurus)
//BR.SYSIN     DD   *
   VBLKSIZE=n1,DBLKSIZE=n2,TBLKSIZE=n3;
/*
```

B. PREFIX -- Revised Version

By John B. Smith

The PREFIX program described in detail in Automated Language Analysis: 1967-1968 has been revised during the past year. As a result, the logic has been greatly simplified -- the program is less than half as long as the first version -- and the run time reduced. Before discussing the revised algorithm, I shall describe briefly the data structures and the logic of the earlier version of PREFIX: however, the reader is urged to consult last year's annual report for a more detailed description and listing of the program.

The earlier version of PREFIX was basically a table-lookup procedure based upon a list of English prefixes, a list of words accompanying each list, and a key. A word that has initial letters that match a prefix does not necessarily have a legitimate prefix. For example, the word ate does not consist of the at prefix and a stem. Consequently, it was necessary to associate with each prefix additional information in the form of a list of words that indicated when a word with matching initial characters is or is not a legitimate prefixed word. Our task would have been simpler if we could have included in this list all words that are exceptions to the rule. This procedure would work well for the prefix in -- there are only some two to three hundred words listed in the Random House Dictionary that contain these initial letters but which are

not prefixed words. However, the exceptions for the prefix a are voluminous. The technique that was employed was to use either an exception or an inclusion list -- whichever is shorter -- and to indicate the kind of list by a key. Thus, associated with each prefix is a list of either exceptions or inclusion words and a key indicating the kind of list.

To optimize processing time, it was decided to assume that text words would be passed in alphabetical order against the alphabetized list of prefixes, resulting in the need for only one pass of both lists. This works well except when one prefix is "contained in" the succeeding prefix -- that is, when the first prefix is shorter in length than the prefix that follows it and when it matches the prefix that follows it character by character. For example, a is "contained in" ab, ad, etc. In this case, words with legitimate a prefixes may come after words with ab prefixes: atypical would come after all words with ab, ad, anti, etc., prefixes. To solve this problem a list of these "troublesome" prefixes was kept and after "normal" processing failed to reveal a prefix, the prefixes on this list were tested. The result was a procedure of some three hundred statements and fairly complex logic.

A simple and more elegant solution to the problem was found by using truth-table logic. (A truth table is matrix representation of all possible combinations of values for relevant parameters and the resulting logical condition.) In the case of PREFIX there are three relevant parameters: the initial letters of a word match or don't match an English

prefix, the word is or is not found in the associated CLUD list, and the CLUD list is an inclusion or exclusion list. The following table (Table # 1) summarizes all possible combinations of these factors and indicates whether or not the word has a legitimate prefix. Ones indicate that the parameter or condition is "true" or present; a zero indicates that the condition is not met.

Clearly, a word has a prefix only in those cases in which the initial letters match. But more significantly, it is meaningless to test for this condition until a check of the CLUD list and its accompanying key is made. To facilitate this procedure, a different data organization was used. The CLUD list was sorted alphabetically and a pointer attached to indicate the associated prefix and key (see Table 2). The complete text can thus be processed with a single pass against the alphabetized CLUD list. A scan of the CLUD list reveals whether a text word is on the list or not. The main processing procedure then branches into two simple nested conditional logic paths. If the word is found, then a test of the associated key is made. If it is a one, indicating an inclusion list, a test for prefix match is made. If all three tests are successful, then we can say that the word has a legitimate English prefix. On the other hand, if the text word is not found in the CLUD list, a quick search of the prefixes beginning with the same letter of the alphabet as the text word is made. Those, and only those, prefixes with a key of zero are tested against the initial letters of the word. If a match is found, we can conclude that

Initial Letter Match	CLUD Match	KEY	Has Prefix?
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 1

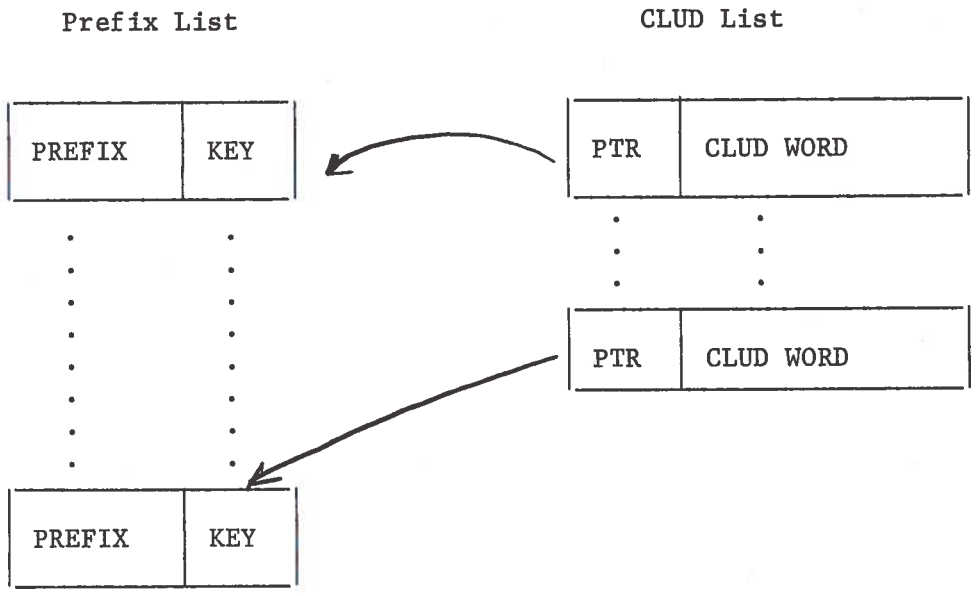


Table 2

the word has a prefix.

Thus, by changing the data representation, the processing algorithm is greatly simplified. Resulting processing time for a text of some 100,000 word-tokens is reduced from 8-10 minutes to 6-8 minutes, a 20-25% increase in efficiency. PREFIX is now catalogued in the VIA1 sequence. For instructions on how to use it, see the User's Manual included in this report.

III. Professional Activities of Project Personnel

Sally Y. Sedelow

Publications:

"Report by the Committee on Courses and Curricula for the Conference on Computer Technology in the Humanities," in Report of the Conference on Computer Technology in the Humanities, ed. L. D. Ohle, University of Kansas, pp. 27-33. See also Appendix C, pp. 49-59, and, with Walter Sedelow, Appendix D, pp. 61-63.

"Computer Education for the Humanist," Proceedings of Symposium Introducing the Computer into the Humanities Curriculum, IBM Manual # 320-1044, IBM, Poughkeepsie, N.Y., 1970. Also printed in Proceedings, Symposium on Computers in Undergraduate Curricula; North Carolina Educational Computing Service, November, 1969, pp. 24-30.

"The Computer in the Humanities: A Contradiction?" Computers in Undergraduate Education, ed. William Viavant, University of Utah, 1969, pp. 123-139.

"Categories and Procedures for Content Analysis in the Humanities," The Analysis of Communication Content, ed. Gerbner, et. al., John Wiley & Sons, Inc., 1969, pp. 487-499. Co-Author with Walter A. Sedelow, Jr.

Review, Louis T. Milic, A Quantitative Approach to the Style of Jonathan Swift, Style, Spring, 1969, pp. 205-207.

Automated Language Analysis, Report on research for the period March 1, 1968 to February 28, 1969, Contract N000 14-67-A-0321, Office of Naval Research, University of North Carolina. DDC # AD 691-451. 286 pp.

Papers/Seminars/Addresses/etc.:

*Speaker, "Computer Systems to Facilitate Research in the Humanities," Virginia Polytechnic Institute, as Distinguished Visting Scholar, May, 1970.

*Speaker, "Computer Education for the Humanist," Symposium: Computers in Undergraduate Curricula, North Carolina Educational Computing Service, November, 1969.

*Speaker, "Computer-Aided Analysis of Humanistic Artifacts," University of British Columbia, September, 1969.

*Speaker, "VIA and MAPTEXT: Programs for Stylistic Analysis," University of Alberta at Edmonton, September, 1969.

*Speaker, "Teaching Computing to the Humanist," IBM Program for Deans of Humanities and Liberal Arts, Poughkeepsie, New York, August, 1969.

*Speaker, "A Curriculum for the Humanist," IBM Symposium on Introducing the Computer into the Humanities, Poughkeepsie, New York, June, 1969.

*Speaker, "Communicating with the Computer About Humanistic Research," State University of New York, Conference at Brockport, April, 1969.

*Speaker, "Computer-Aided Research in the Humanities,"
Virginia Polytechnic Institute (Distinguished Visiting Scholar), May, 1970;
Clarke College, Iowa. (ACM/NSF Visiting Scientist Program.), May, 1970;
University of Kansas, February, 1970;
University of Nebraska at Omaha, (ACM/NSF Visiting Scientist and Program), December, 1969;
University of New Mexico, November, 1969;
Maryville College, Tennessee, (ACM/NSF Visting Scientist Program),October, 1969;
University of California at Berkeley, June, 1969.

*Speaker, "Computer-Aided Stylistic Analysis,"
Omaha, Nebraska, ACM Chapter, December, 1970;
University of Illinois, May, 1969;
Colgate University, Hamilton, New York, April, 1969;
State University of New York at Buffalo, April, 1969.

*Speaker, "The Computer and Liberal Education,"
Virginia Polytechnic Institute, (Distinguished Visting Scholar), May, 1970;
Lawrence University, (ACM/NSF Visting Scientist Program), April, 1970.

Activities:

*Member, Technical Area Committee 7 on "Science and Humanities," for Fifth IFIP Congress, Ljubljana, Yugoslavia, 1971.

*Consultant, ACM College Consulting Service, 1970- .

*Visiting Scientist, National Science Foundation, Association for Computing Machinery Visiting Scientist Program, 1969-1970.

*Proposal Evaluation, National Endowment for the Humanities, 1969- .

*Member of Board of Directors for the Rhetoric Society of America, 1969- .

*Consultant, Coordinated Science Laboratory, Cognitive Memory Project, University of Illinois, May 19-23, 1969.

*Chairman, Curriculum Section, ACLS/NSF Conference on Computing Technology in the Humanities, University of Kansas, September 2-6, 1969.

*Consultant, University of New Mexico, November 29-30, 1969.

*Proposal Evaluation, Canada Council, 1968- .

*Member of Advisory Panel, National Science Foundation's Institutional Computing Services Section, 1968- .

*Chairman, Special Interest Committee on Language Analysis and Studies in the Humanities, Association for Computing Machinery, 1968-1970.

*Field Reader of Proposals, U.S. Department of Health, Education, and Welfare, 1966- .

*Co-Editor, Computer Studies in the Humanities and Verbal Behavior, 1966- .

Martin Dillon

Publications:

"Advanced Basis Selection for A Class of Large Linear Programming Problems," Ninth Annual Southeast Regional Conference of the ACM, May, 1970.

Papers/Seminars/Addresses/etc.:

"Linear Programming Applications in Library Management," School of Library Science, University of North Carolina, March, 1969.

"A Contextual Theory of Meaning," Computer Science Department, Purdue University, March, 1969.

Gerald A. Fisher, Jr.

Publications:

"A Computer Investigation of Verbal Characteristics of Effective Classroom Lecturing," American Educational Research Association Journal 6 No. 4, November, 1969. Co-author with Jack Hiller and Walter Kaess.

"Program for Essay Analysis," in The Analysis of Essays by Computer, E. B. Page and D. H. Paulus, Final Report, Project #6-1318, pp. 199-242. Co-author with Mary Ann Fisher.

"A Parsing Program" in The Analysis of Essays by Computer, E.B. Page and D. H. Paulus, Final Report, Project #6-1318, pp. 252-263.

Papers/Seminars/Addresses/etc.

Paper, "On the representation of Formal Languages Using Automata on Networks," Tenth Annual Symposium on Switching and Automata Theory, Kitchner-Waterloo, October, 1969.

Paper, "Current Perspectives in Automated Content Analysis," ACM National Conference, San Francisco, August, 1969. Co-author with Jack Hiller and Donald Marcotte.

Paper, "A Computer Investigation of Verbal Characteristics of Effective Classroom Lecturing," American Educational Research Association Conference, Chicago, February, 1968.

Participant, NSF Summer Research Conference in Automata and Computational Complexity, Plattsburgh, New York, June 23-28, 1969.

Instructor, Study Institute on Computer-Aided Language Analysis in Education, American Educational Research Association, Chicago, February, 1968.

Instructor, Summer Institute for Humanistic Computation, Lawrence, Kansas, June-August, 1970.

Walter A. Sedelow, Jr.

Publications:

*German edition of "Stylistic Analysis" (from Automated Language Processing) in Literary Science and Linguistics: Results and Perspectives, edited by Jens Ihwe for Verlag Gehlen. Forthcoming.

*The Librarians Speaking, University of Georgia Press, 1970, pp. 130-141. (Interviewed and edited by Guy R. Lyle).

*"A Suggested Program for . . . Training in Computer-Aided Research in the Humanities," Report of the Conference on Computer Technology in the Humanities, Lawrence, Kansas: University of Kansas, June, 1970, pp. 61-63. Co-author with Sally Y. Sedelow.

*"Categories and Procedures for Content Analysis in the Humanities," in The Analysis of Communication Content: Developments in Scientific Theories and Computer Techniques, George Gerbner, et. al., ed., New York: John Wiley and Sons, Inc., 1969, pp. 487-499. Co-author with Sally Y. Sedelow.

*"Computers in the Social/Behavioral Sciences," in Proceedings of the Park City Conference; Computers in Undergraduate Education, Vol. I, Salt Lake City: University of Utah, September 1969, pp. 210-212.

*Editor, "Concepts in Human Sciences Education," Proceedings of the Park City Conference; Computers in Undergraduate Education, Vol. II, Salt Lake City: University of Utah, September 1969, pp. 26-79.

*"History as Language," Computer Studies in the Humanities and Verbal Behavior, Vol. I, No. 4, December 1968. (Published in September 1969.)

*Book Review, Review of Burton R. Pollin, Godwin Criticism, A Synoptic Bibliography (Toronto: University of Toronto Press, 1967), in American Journal of Edonomics and Sociology, January, 1970, Vol. 29, No. 1, pp. 108-112.

*Book Review, Review of Norman Birnbaum and Gertrud Lenzer, eds., Sociology and Religion, A Book of Readings (New York: Prentice-Hall, 1969.), Amherst Alumni News, Vol. XXII, No. 2, pp. 36-37.

*Series Editor, The Free Press/Macmillan Company, 1968- .

*Associate Editor, Social Forces, 1966-70.

*Board of Editors, Computer Studies in the Humanities and Verbal Behavior.

Professional Experience:

*Professor, Sociology and Computer Science, University of Kansas, 1970- .

*Professor, Sociology and Computer and Information Science; and Research Professor, Institute for Research in Social Science, University of North Carolina at Chapel Hill, 1968-70.

*Dean, School of Library Science, University of North Carolina at Chapel Hill, 1967-70.

Activities:

*Referee, Technical Papers, Fall Joint Computer Conference, 1970.

*"Language Analysis by Machine" and "The Intersection of Computer Science," N.S.F. Summer Institute in Numerical and Statistical Methods of Digital Computing . . . , University of Missouri at Rolla, July 8, 1970.

*"Current Issues in the Sociology of Science," IBM Management Study Program, Research Triangle Park, North Carolina, March 17, 1970.

*"The Computer and Undergraduate Education," Rollins College, Winter Park, Florida, January, 1970, (N.S.F. Visiting Scientist Program.)

*Member, Interdisciplinary Committee for a UNC-CH Operations Research and Systems Analysis Curriculum, 1969- .

*Chairman, Section on Informational and Social Aspects of Advanced Technology, and Steering Committee Member, "Sciences in Interaction: A NASA-Related University Program in the Life, Physical, and Social Sciences," 1969- .

*Principal Investigator, "Computer-Aided Analysis of Inter-disciplinary Discourse Barriers," NASA Project 325-NAS-4-401.

*Visiting Scientist, National Science Foundation/Association for Computing Machinery Visiting Scientist Program, 1969-70.

*Conferee, re Computing and the Social Sciences, University of Nebraska at Omaha, December 1969.

*Review Panel Member, Research and Studies Program, Office of Science Information Service, National Science Foundation, December 1969.

*Conferee, re Computing and the Social Sciences, University of New Mexico, November 1969.

*Invited Participant, National Science Foundation/American Council of Learned Societies' Conference on Computer Technology in the Humanities, University of Kansas, September 1969.

*Member, Steering Committee of the Special Interest Committee for Social Science Computation (SICSOC) of the Association for Computing Machinery (ACM), June 1969- .

*Consultant, Jacksonville (Illinois) State Hospital, February 1968-70.

*Member, Faculty Council Committee on University Self-Study Subcommittee on Professional Schools, University of North Carolina, Chapel Hill, 1968-69.

*Member, American Council of Learned Societies' Committee on Information Technology, February 1968- . (Also Sub-Committee on Funding, 1969- .)

*Consultant, Computer-Aided Linguistic Analysis Project (sponsored by the U.S. Office of Naval Research), University of North Carolina, Chapel Hill, March 1967- .

*Member, University Research Council Sub-Committee for the Social Sciences and Professional Schools, University of North Carolina, Chapel Hill, 1967-70.

*Member, Committee on University Government, University of North Carolina, Chapel Hill, 1967-70.

*Trustee, International Social Science Institute, 1966- .

Walter L. Smith

Publications:

(Jointly written with William E. Wilkinson) "On branching processes in random environments," Ann. Math. Statist. 40: 814-827 (1969).

"Some remarks on a distribution occurring in neural studies," Essays in Probability and Statistics, pp. 707-732, edited by R. C. Bose, et. al., U.N.C. Press, Chapel Hill (1970).

"A contribution to the theory or remnants," Proceedings of the 37th Session of the International Statistical Institute, Book 2, pp. 368-370, London (1969).

"Some results using general moment functions," Journal of the Australian Mathematical Society, Vol. X (1969), pp. 429-441.

H. William Buttelmann

Publication:

"Syntax-Semantics Systems as Structure Manipulation Systems: Phrase Structure Grammars and Generalized Finite Automata", Ph.D. Dissertation, University of North Carolina at Chapel Hill, July 1970.

Papers/Seminars/Addresses, etc.

*Lecture, "Structure Manipulation Systems,"
Ohio State University, March, 1970;
University of Kansas, March, 1970;
Rice University, April, 1970.

*Participant, NSF Symposium in Mathematics, Morehouse College, September, 1969.

John B. Smith

Publication:

"PREFIX," in Sally Yeates Sedelow, Automated Language Analysis: 1968-1969.

"Principle Component Analysis: A Discussion," in Sedelow.

Professional Activities:

*Degrees: Ph.D., U.N.C., August 1970.

*Participant: Conference on Humanistic Computation," Kansas University, September 1970.

*Research Assistant, Institute for Humanistic Computation, Kansas University, June 1970 - August 1970.

APPENDIX A

Program Documentation of DETEXT - A Text Decomposition and File Creation Program Set

by

Joe R. Ragland

Operation and Use of DETEXT

DETEXT was designed to allow flexibility in using the program. As a result, optional parameters with default values have been provided to control and specify environmental criteria relative to a particular application of the program.

EXEC Card Parameters

Certain specifications must be provided as key-word parameters through the PARM option of the execute card. If the parameters are not present default values will be assumed in each case.

The parameters are as follows:

SIZE — Amount of core to be given to the OS/360 sort program (in addition to the PL/I DETEXT program). Default SIZE is 44K bytes.

CDMGIN=(xx,yy) — Specifies the margins to be used in scanning input text cards or card images from the text input file. CDMGIN=(1,72) is the default value.

TLEN — Maximum token length to be scanned. Tokens longer than TLEN will be truncated on the right. TLEN=18 characters is the default.

INPUT — Indicates the DD name of the DD card from which text input is to be read. SYSIN is the default.

EXEC Card Example

```
//TRUN     EXEC   PGM=DETEXT,PARM='SIZE=56000,INPUT=TEXTIN'
```

The resulting options are SIZE=56K, CDMGIN=(1,72), TLEN=18 and INPUT=TEXTIN would be a dataset specified by a dd Statement of

the form //TEXTIN DD . . .

Text Input

Text input consists of a title for the text and the text prose itself. The title (a single card) is always input from the standard SYSIN dataset. The text may or may not be input from this dataset. If the DD name specified by the INPUT parameter is other than SYSIN then it is necessary to provide a single card as SYSIN to be used as title identification. If the INPUT parameter takes the SYSIN default or if INPUT specifies SYSIN then text cards follow the title in the SYSIN dataset. However, in this case only, a blank card must be inserted between the title card and the first card of the text. This blank card is not necessary if INPUT specifies a dataset other than SYSIN. If the blank card is omitted then the first text card is ignored.

Text input may be terminated by inserting a card with $$$$$ punched in columns 1-5. In the absence of this card, input text scanning will continue until the end-of-file condition occurs on the dataset named by the INPUT parameter.

Specifying DETEXT Run Datasets

The four output files (TOKEN, TYPE, GLOSS, INDTAB) and one temporary file (POINTER) must be specified by DD statements at execution time. PL/I stream I/O is used for the TOKEN, TYPE, GLOSS and INDTAB files. Record I/O is used with the POINTER file. For a given run the TOKEN and TYPE files will

require significantly more space than the GLOSS and INDTAB files. The stream I/O files should specify DCB=(BLKSIZE=7200,RECFM=V) for 2314 disk and DCB=(BLKSIZE=3600,RECFM=V) for 2311 disk.

The temporary file POINTER should be specified as DCB=(RECFM=FB,LRECL=400). The BLKSIZE should be 7200 or 3600 as appropriate.

In addition to the above files, certain other datasets must be included. Specifically these are the SORTWK01, SORTWK02, SORTWK03, SORTLIB, and SYSOUT datasets for the sort. For this particular application of the OS/360 sort only three sort work areas are desired. Finally, the standard PL/I print dataset, SYSPRINT, must be defined.

DETEXT Run Example

```
//GEN  JOB  -----------------
//     EXEC  PGM=DETEXT,PARM='TLEN=22,CDMGIN=(1,80)'
//SYSPRINT  DD  SYSOUT=A
//TOKEN     DD  UNIT=DISK,SPACE=(CYL,(5,3)),DCB=(BLKSIZE=7200,RECFM=V)
//TYPE      DD  UNIT=DISK,SPACE=(CYL,(5,3)),DCB=(BLKSIZE=7200,RECFM=V)
//GLOSS     DD  UNIT=DISK,SPACE=(CYL,(1,1)),DCB=(BLKSIZE=7200,RECFM=V)
//INDTAB    DD  UNIT=DISK,SPACE=(CYL,(1,1)),DCB=(BLKSIZE=7200,RECFM=V)
//POINTER   DD  UNIT=DISK,SPACE=(TRK,(5,3)),                          *
//                DCB=(RECFM=FB,LRECL=400,BLKSIZE=7200)
//SORTWK01  DD  UNIT=DISK,SPACE=(TRK,200,,CONTIG)
//SORTWK02  DD  UNIT=DISK,SPACE=(TRK,200,,CONTIG)
//SORTWK03  DD  UNIT=DISK,SPACE=(TRK,200,,CONTIG)
//SORTLIB   DD  DSN=SYS1.SORTLIB,DISP=SHR
//SYSOUT    DD  SYSOUT=A
//SYSIN     DD  *
DETEXT  TEXT  ID  CARD
 (Blank card)
FIRST TEXT CARD
   .
   .
   .
$$$$$
/*
```

In the above example, text input is from the SYSIN dataset (by default option), the maximum token length is 22 characters, input scanning is from column 1 through column 80 and 44,000 bytes is to be given the OS/360 sort for processing.

```
STMT LEVEL NEST
                    /* DETEXT - TEXT DECOMPOSITION & FILE BUILDING - MAIN PROCEDURE */
  1                 DETEXT: PROC(PARM) OPTIONS(MAIN);
                    /*********************************************************************
                     *                                                                   *
                     * DETEXT: A TEXT DECOMPOSITION AND FILE BUILDING PROGRAM.           *
                     *                                                                   *
                     * SECTION ONE: DETERMINE PARAMETER OPTIONS AS FOLLOWS:              *
                     *                                                                   *
                     *    SIZE             SIZE OF CORE AVAILABLE (IN ADDITION TO DETEXT)*
                     *                     FOR HOLDING AND RUNNING THE OS/360 SORT.     *
                     *    COMGIN=(XX,YY)   MARGINS FOR SCANNING INPUT CARDS FOR TEXT.   *
                     *                     COMGIN=(1,72) IS DEFAULT.                    *
                     *    TLEN             MAXIMUM TOKEN LENGTH TO BE USED.             *
                     *                     TLEN=18 IS DEFAULT.                          *
                     *    INPUT            FILE DD NAME FOR TEXT INPUT                  *
                     *                     'SYSIN' IS DEFAULT.                          *
                     *                                                                   *
                     *        THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF THE     *
                     *   REQUIREMENTS FOR THE M.S. DEGREE IN COMPUTER AND INFORMATION    *
                     *   SCIENCE AT THE UNIVERSITY OF NORTH CAROLINA AT CHAPEL HILL      *
                     *   UNDER THE DIRECTION OF DR. GERALD A. FISHER.                    *
                     *                                                                   *
                     *        THE AUTHOR WISHES TO EXPRESS HIS APPRECIATION TO THE       *
                     *   TRIANGLE UNIVERSITIES COMPUTATION CENTER FOR PROVIDING COMPUTER  *
                     *   FACILITIES TO DEVELOP THIS PROGRAM.                             *
                     *                                                                   *
                     *            AUTHOR:        JOE R. FAGLAND                          *
                     *                           TRIANGLE UNIVERSITIES COMPUTATION CENTER*
                     *                           P. O. BOX 12175                         *
                     *                           RESEARCH TRIANGLE PARK, N. C.  27709    *
                     *                           MAY 20, 1970                            *
                     *                                                                   *
                     *********************************************************************/
  2     1           DCL NC(2:1) CHAR(1) INITIAL('N',' ');
  3     1           DCL PARM CHAR(40) VAR;
  4     1           DCL P CHAR(42),(TOKEN,TYPE) FILE;
  5     1           DCL INPUT CHAR(8) VAR;
  6     1           DCL TLEN BINARY FIXED;
  7     1           TLEN=18;                    /* DEFAULT TOKEN LENGTH */
  8     1           SIZE=44000;                 /* DEFAULT STORAGE SIZE */
  9     1           L1=1; L2=72;                /* DEFAULT CARD MARGINS */
 11     1           INPUT='SYSIN';              /* DEFAULT INPUT FILE */
 12     1           IF LENGTH(PARM)=0 THEN GO TO START;
 14     1           P=','||PARM||',';
 15     1           J=INDEX(P,'TLEN=');         /* CHECK FOR TOKEN LENGTH OPTION */
 16     1           IF J¬=0 THEN DO;K=INDEX(SUBSTR(P,J+5),',')-1; TLEN=SUBSTR(P,J+5,K);END;
 21     1           J=INDEX(P,'SIZE=');         /* CHECK FOR STORAGE SIZE OPTION */
 ..     1           IF J¬=0 THEN DO; K=INDEX(SUBSTR(P,J+5),',')-1;
  .     1     1         SIZE=SUBSTR(P,J+5,K); END;
 27     1           J=INDEX(P,'COMGIN=');       /* CHECK CARD MARGIN OPTION */
 28     1           IF J¬=0 THEN DO; K1=INDEX(SUBSTR(P,J+8),',');
 31     1     1         K2=INDEX(SUBSTR(P,J+8+K1),',');
 32     1     1         IF K2¬=0 THEN DO; L1=SUBSTR(P,J+8,K1-1);
 35     1     2             L2=SUBSTR(P,J+8+K1,K2-1);
 36     1     2         END; END;
 38     1           J=INDEX(P,'INPUT=');        /* CHECK FOR INPUT FILE */
```

```
/* XREF - TEXT PROCESSING & FILE BUILDING - MAIN PROCEDURE */

STMT LEVEL NEST

 33     1          IF I<>0 THEN DO;
 41     1    1         K=INDEX(SUBSTR(P,I+6),',')-1;
 42     1    1         INPUT=SUBSTR(P,I+6,K);    END;
 44     1          START:
                   TLEN=MAX(TLEN,11);      /* 11 IS MINIMUM TOKEN LENGTH DUE TO SORT REQ */
 45     1          PUT EDIT('DFTEXT VERSION 1 -- OPTIONS USED THIS RUN ARE: TLEN=',TLEN)
                        (A,F(2));
 46     1          PUT EDIT('SIZE=',SIZE,'COMMON=(',L1,',',L2,')','INPUT=',INPUT)
                        (SKIP,X(47),A,F(6),SKIP,X(47),A,F(2),A,F(2),A,SKIP,X(47),A,A);
 47     1          PUT PAGE;
 48     1          CALL SORT1(TLEN,SIZE,L1,L2,INPUT);
 49     1          CLOSE FILE(GLOSS);     CLOSE FILE(INDTAB);
 51     1          CLOSE FILE(TOKEN);  CLOSE FILE(TYPE);
 53     1          CALL SORT2(SIZE,TLEN);
 54     1          CLOSE FILE(TOKEN);
 55     1          CALL LIST(TLEN);
 56     1          STOP; END DFTEXT;
```

155

```
STMT LEVEL NEST
                /* SORT1 - PROCEDURE TO SCAN INPUT, SORT TOKENS, AND BUILD FILES */
  1             SORT1: PROC(TLEN,SIZE,L1,L2,INPUT);
                /*********************************************************************
                 *                                                                   *
                 * SORT1: A PROCEDURE TO READ INPUT TEXT, BUILD AND LINEARLY         *
                 *        NUMBER TOKENS, CREATE "TYPES", "GLOSS", AND "INDTAB" FILES *
                 *        AND ESTABLISH "TOKEN" FILE FOR SORTING BY THE SORT2        *
                 *        PROCEDURE.                                                 *
                 *                                                                   *
                 *    FILE FORMATS:                                                  *
                 *                                                                   *
                 *      IN THE FILE FORMATS BELOW, A(2) IS THE UNSPEC OF HALF-WORD   *
                 *      FIXED BINARY AND A(4) IS THE UNSPEC OF FULL-WORD FIXED       *
                 *      BINARY.  THE HEADER RECORDS ARE IN CHARACTER FORM AND CONSIST*
                 *      OF FILE NAME (5 BYTES), TEXT ID (90 BYTES), MAXIMUM TOKEN    *
                 *      LENGTH (3 BYTES), TOTAL NUMBER OF TOKENS (6 BYTES), EIGHT    *
                 *      NUMBERS (5 BYTES EACH) GIVING THE NUMBER OF OCCURRENCES      *
                 *      OF EACH CATEGORY.                                            *
                 *                                                                   *
                 *      TYPE(FINAL)   HEADER RECORD OF 134 BYTES.                    *
                 *                    TYPE RECORDS IN ALPHA ORDER BY TOKEN:          *
                 *                    TOKEN LENGTH              A(2)                 *
                 *                    TOKEN                     A(TOKEN LENGTH)      *
                 *                    FREQUENCY OF OCCURRENCE   A(2)                 *
                 *                    OCCURRENCE POSITION 1     A(4)                 *
                 *                    OCCURRENCE POSITION 2     A(4)                 *
                 *                       ...                    ...                  *
                 *                    OCCURRENCE POSITION(FREQ) A(4)                 *
                 *                                                                   *
                 *      GLOSS(FINAL)  HEADER RECORD OF 134 BYTES.                    *
                 *                    GLOSS RECORDS IN ALPHA ORDER BY TOKEN:         *
                 *                    TOKEN LENGTH              A(2)                 *
                 *                    TOKEN                     A(TOKEN LENGTH)      *
                 *                    FREQUENCY OF OCCURRENCE   A(2)                 *
                 *                                                                   *
                 *      TOKEN(TEMP)   HEADER RECORD OF 134 BYTES.                    *
                 *                    TOKEN RECORDS IN ALPHA ORDER BY TOKEN:         *
                 *                    LINEAR TEXT-ORDER NO.     A(4)                 *
                 *                    TOKEN LENGTH              A(2)                 *
                 *                    TOKEN                     A(TLEN)   *FIXED*    *
                 *                    8-BIT STRING              A(1)                 *
                 *                    POINTER TO TYPE RECORD    A(2)                 *
                 *                                                                   *
                 *      INDTAB(FINAL) HEADER RECORD OF 134 BYTES.                    *
                 *                    INDEX RECORDS BY CATEGORY NUMBER:              *
                 *                    CATEGORY NUMBER           A(1)   *CHAR*        *
                 *                    TOKEN POINTER             A(4)                 *
                 *                                                                   *
                 *********************************************************************/
  2     1         DCL INSERT ENTRY(CHAR(35),CHAR(42),FIXED BINARY(31,0),
                        FIXED BINARY(31,0),ENTRY,ENTRY),
                    INSERT ENTRY(FIXED BINARY(31,0));
  3     1         DCL ENTIS RETURNS(CHAR(TLEN+7)),
                    SFIELD CHAR(35), PFIELD CHAR(42),
                     RETCDE FIXED BINARY(31,0);
  4     1         DCL IDX(8) CHAR(5);
```

```
STMT LEVEL NEST

  5     1          DCL INPUT CHAR(8) VAR;
  6     1          DCL COMPR CHAR(TLEN);
  7     1          DCL CARD CHAR(80);
  8     1          DCL TLEN FIXED BINARY, SIZE FIXED BINARY(31,0);
  9     1          DCL STR2 CHAR(400), 2(100) CHAR(4) DEFINED STR2, BUFF CHAR(400);
 10     1          DCL PSW FIXED BINARY;
 11     1          DCL (K,KTP) FIXED BINARY(31,0);
 12     1          DCL IN, (P1,P2) PICTURE '999';
 13     1          DCL HD CHAR(80);              /* INPUT TEXT NAME CARD */
 14     1          DCL D(12) CHAR(4) INITIAL('.','?','!','..','¢¢$','¢¢$$','¢','¢¢',
                                              '¢¢¢','¢¢¢¢'),
                        BM(10) BIT(8) INITIAL('10000000'B,'10000000'B,'10000000'B,
                              '11000000'B,'11100000'B,'11110000'B,'11111000'B,
                              '11111100'B,'11111110'B,'11111111'B);
     1          DCL TLK CHAR( 2N+1);
 16     1          DCL ENTREC CHAR (TLEN+7) DEF ENTRECD,
                    1 ENTRECD UNALIGNED,
                      2 LEN CHAR(4),
                      2 LEN CHAR(2),
                      2 REST CHAR(TLEN),
                      2 BOB BIT(9);
 17     1          DCL LATCH BIT(8);
 18     1          DCL INDATA FILE INPUT INTERNAL RECORD;
 19     1          DCL INDCTR(3), B BIT(1);
 20     1          P1=TLEN;
 21     1          P2=TLEN+7;
 22     1          SRFLD=' SORT FIELDS=(7,'||R1||',CH,A,1,4,CH,A) ': /* CONTROL STAT */
 23     1          RZFLD=' RECORD TYPE=F,LENGTH=('||P2||') ';
 24     1          GET FILE(SYSIN)  EDIT(CARD)(A(80)) COPY;   /* INPUT TEXT HEADER CARD */
 25     1          HD=CARD;
 26     1          OPEN FILE(INDATA) TITLE(INPUT);
 27     1          OPEN FILE(INDTAB) OUTPUT;   INDCTR=0;
 29     1          KTP=0;                          /* LINEAR TOKEN COUNTER */
 30     1          K=1; J=L2; I=1; BOB='11111111'B; LATCH='00000000'B;
 35     1          MARK,TJ=0;
 36     1          ON ENDFILE(INDATA) II=1;
 38     1            CALL IHESRTD(SRFLD,RZFLD,SIZE,RETCDE,ENT15,ENT35);
 39     1            IF RETCDE=16 THEN DO;
 41     1   1             PUT SKIP LIST('SORT1 WAS UNSUCCESSFUL');
 42     1   1             STOP;  END;
 44     1          ENT15: PROC RETURNS (CHAR(TLEN+7)); /* PASS RECORDS TO SORT */
 45     2          IF ISW=1 THEN DO;      /* TEST IF INDEX ENTRIES PRESENT */
 47     2   1        RFST=SUBSTR(IDX(IK),1,1);       /* EXTRACT CATEGORY NUMBER */
 48     2   1        LEN=SUBSTR(IDX(IK),2);          /* EXTRACT TOKEN POINTER */
 49     2   1        LEN=' ';
 50     2   1        IF IK=IM THEN ISW=0;
 52     2   1        IK=IK+1;
 53     2   1        CALL IHESADC(12);
 54     2   1        RETURN(ENTREC);             /* RETURN INDEX ENTRIES */
 55     2   1      END;
 56     2          SCAN: I=I+1;
 57     2          IF I>L2 THEN DO;
 59     2   1        I=1;
  2     2   1        READ FILE(INDATA) INTO (CARD);
```

```
STMT LEVEL NEST

 61      1    1     IF KTR>=10000 THEN LI=1;    /* PUTS STAT FOR THIS LIMITED RUN ONLY */
 62      1    1         IF LI=1 THEN GO TO TERMIN;   /* LI=1 WHEN SYSIN EXHAUSTED */
 65      1    1         END;
 66      2         LI=L2-I+1;
 67      2         I=INDEX(SUBSTR(CARD,I,LI),' ');  /* FIND NEXT BLANK */
 68      2         IF I=1 THEN GO TO SCAN;  /* ADJACENT BLANKS */
 71      2         IF I=0 THEN DO;   /* REMAINDER OF CARD IS TEXT */
 72      2    1         SUBSTR(TOK,K)=SUBSTR(CARD,I,LI);
 73      2    1         K=MIN(KLII,TLEN+1);
 74      2    1         J=L2;
 75      2    1         GO TO SCAN;
 76      2    1         END;
                             /* WE HAVE A BLANK PRECEDED BY J-1 LETTERS OF TEXT */
 77      2         SUBSTR(TOK,K)=SUBSTR(CARD,I,J);
 78      2         IF TOK='$$$$$$' THEN GO TO TERMIN;     /* TEST FOR END OF TEXT */
 80      2         K=MIN(K+J-1,TLEN+1);  /* K-1 IS LENGTH OF WORD */
 81      2         KK=K-1;                    /* KK IS LENGTH OF TOKEN */
                             /* TEST FOR END OF SENTENCE */
 82      2         IF SUBSTR(TOK,1,1) >= 'a' THEN GO TO LTEST;
 84      2         DO JJ=1 TO 10;
 85      2    1         IF SUBSTR(TOK,1,KK)=D(JJ) THEN DO;
 87      2    2              LATCH=LATCH|DM(JJ);
 88      2    2              IF DM(JJ)='10000000'B THEN DO;
 89      2    3                MARK=1;  /* SET END-OF-SENTENCE INDICATOR */
 91      2    3                GO TO PUTOUT;       END;
 93      2    2                ELSE IF TOK='..' & MMARK=0 THEN DO; /* NO SENT END PRESENT*/
 95      2    3                  TOK='.';  GO TO PUTOUT;  END;
 98      2    2                  ELSE DO; /* OMIT THIS DELIMITER */
 99      2    3                    K=1; GO TO SCAN;  END;
102      2    2              END;
103      2    1         END;
104      2         LTEST: IF LATCH='00000000'B THEN GO TO PUTOUT;
106      2         BOOL(BOOL,LATCH,'0110'B);  /* EXCLUSIVE OR */
107      2         IM=0;
108      2         DO JJ=1 TO 8;
109      2    1      B=SUBSTR(LATCH,JJ,1);
110      2    1      IF B THEN DO;  INDCTR(JJ)=INDCTR(JJ)+1;
113      2    2         IM=IM+1;   ISW=1;
115      2    2         PUT STRING(IDX(IM)) EDIT(JJ,III)(F(1),A);
116      2    2      END;
117      2    1      IK=1;
118      2    1    END;
119      2         LATCH='00000000'B;
1.       2         PUTOUT: KTR=KTR+1;
121      2         UNSPEC(LEN)=UNSPEC(KK);
122      2         UNSPEC(LIN)=UNSPEC(KTR);
123      2         REST=SUBSTR(TOK,1,KK);
124      2         MMARK=MARK;  /* SAVE END-OF-SENTENCE INDICATOR */
125      2         MARK=0;
126      2         K=1;
127      2         CALL INTSARC(12);
128      2         RETURN(SNTREC);
129      2         TERMIN:
130      2           K,IM,ISW=0;  I,L=1;      COMBR=' ';
```

```
SMNT LEVEL NEST
                /* SORT1 - PROCEDURE TO SCAN INPUT, SORT TOKENS, AND BUILD FILES */

132      2          PUT FILE(TOKEN) EDIT('TOKEN',HD,TLEN,KTR,(INDCTR(II) DO II=1 TO 8))
                      (A(5),A(80),F(3),F(6),(8) F(5));          /* TOKEN FILE HEADER */
133      2          PUT FILE(TYPE)  EDIT('TYPE ',HD,TLEN,KTR,(INDCTR(II) DO II=1 TO 8))
                      (A(5),A(80),F(3),F(6),(8) F(5));          /* TYPE FILE HEADER  */
134      2          PUT FILE(GLOSS) EDIT('GLOSS',HD,TLEN,KTR,(INDCTR(II) DO II=1 TO 8))
                      (A(5),A(80),F(3),F(6),(8) F(5));          /* GLOSS FILE HEADER */
135      2          PUT FILE(INDTAB) EDIT('INDEX',HD,TLEN,KTR,(INDCTR(II) DO II=1 TO 8))
                      (A(5),A(80),F(3),F(6),(8) F(5));          /* INDEX FILE HEADER */
136      2          CALL THRSAFC(3);
137      2          RETURN;
138      2        END ENT15;
139      1          ENT35: PROC(OUTREC);              /* RETURN SORTED RECORDS */
140      2            DCL OUTREC CHAR(TLEN+7),
                       1 OUTRECD UNALIGNED DEF OUTREC,
                         2 LIN CHAR(4),
                         2 LEN CHAR(2),
                         2 REST CHAR(TLEN),
                         2 BOP BIT(9);
141      2            DCL (LL,LT,FO) CHAR(2);
142      2             IF LEN=' ' THEN DO;     /* LEN=' ' FOR INDEX RECORDS */
144      2  1             PUT FILE(INDTAB) EDIT(REST,LIN)(A(1),A);
145      2  1             CALL THRSAFC(4);
146      2  1             RETURN;
147      2  1           END;
148      2              K=K+1;                          /* K COUNTS RETURNED RECORDS */
149      2              IF REST=COMPR THEN I=I+1; ELSE DO; /* I IS FREQUENCY COUNTER */
152      2  1             IF IN=1 THEN DO;      /* IN=0 ON FIRST ENTRY ONLY */
154      2  2             UNSPEC(FO)=UNSPEC(I);
155      2  2             L=L+1;                  /* L = INDEX OF TOKEN IN TYPE FILE */
156      2  2             LT=UNSPEC(LT);
157      2  2             PUT FILE(TYPE) EDIT(LT,COMPR,FO)(A(2),A(TT),A(2));
158      2  2             PUT FILE(GLOSS) EDIT(LT,COMER,FO)(A(2),A(TT),A(2));
159      2  2             IF PSW=1 THEN DO;     /* PSW=1 IF POINTER FILE USED */
161      2  3               CLOSE FILE(POINTER);
162      2  3               ON ENDFILE(POINTER) BEGIN;
164      3  3                     CLOSE FILE(POINTER);  GO TO HIT1;  END;
167      2  3               DO NN=1 BY 1;
168      2  4                 READ FILE(POINTER) INTO(BUFF);
169      2  4                 PUT FILE(TYPE) EDIT(BUFF)(A(400));
170      2  4               END;
171      2  3             END;
172      2  2             HIT1: PUT FILE(TYPE) EDIT((Z(NN) DO NN=1 TO N-1))((N-1) A(4));
173      2  2             PSW=0;
174      2  2             END;
175      2  1             I,IN,N=1;  COMPR=REST;  UNSPEC(LT)=UNSPEC(LEN);
178      2  1           END;
179      2              UNSPEC(LL)=UNSPEC(I);    /* LL = POINTER TO TYPE FILE */
180      2              PUT FILE(TOKEN) EDIT(OUTREC(1:11))(A);
181      2              Z(N)=LIN;                 /* SAVE LINEAR # OF THIS TOKEN */
182      2              IF N=100 THEN DO;         /* OVERFLOW POINTERS TO FILE */
184      2  1             PSW=1;  N=0;
186      2  1             WRITE FILE(POINTER) FROM(STR7);
187      2  1           END;
188      2              N=N+1;
```

```
                * [illegible] TO SORT FORMT, SORT TOKENS, AND BUILD FILES */                    159

STMT LEVEL NEST

183     2             IF K<KTP THEN DO;  /* KTP = # RECORDS PASSED TO SORT */
191     2    1           CALL IHESARC(4);
192     2    1           RETURN; END;
194     2              UNSPEC(FO)=UNSPEC(T);   /* PASS HERE ON LAST RECORD RETURNED ONLY*/
195     2              II=UNSPEC(LT);
196     2              PUT FILE(TYPE) EDIT(LT,COMPR,FO)(A(2),A(II),A(2));
197     2              PUT FILE(GLOSS) EDIT(LT,COMPR,FO)(A(2),A(II),A(2));
199     2              IF PSW=1 THEN DO;       /* PSW=1 IF POINTER FILE USED */
200     2    1         CLOSE FILE(POINTER);
201     2    1         ON ENDFILE(POINTER) GO TO HIT2;
203     2    1         DO NN=1 BY 1;
204     2    2           READ FILE(POINTER) INTO(BUFF);
205     2    2           PUT FILE(TYPE) EDIT(BUFF)(A(800));
206     2    2         END;
207     2    1         END;
208     2              HIT2: PUT FILE(TYPE) EDIT((T(NN) DO NN=1 TO N-1))((N-1) A(4));
209     2              CLOSE FILE(POINTER);
210     2              CALL IHESARC(8);              /* FINAL SORT CALL */
211     2              RETURN;
212     2            END ENT35;
213     1          STOP: END SORT1;
```

```
STMT LEVEL NEST
                /* SORT2 - PROCEDURE TO SORT TOKENS BACK INTO ORIGINAL TEXT ORDER */
  1             SORT2: PROC(SIZE,TLEN);
                /*********************************************************************
                 *                                                                   *
                 *  SORT2: A PROCEDURE TO INPUT TOKEN RECORDS AND SORT INTO ORIGINAL *
                 *         TEXT ORDER.  THE LINEAR ORDER NUMBER IS DROPPED AFTER     *
                 *         SORTING.                                                  *
                 *                                                                   *
                 *      TOKEN FILE FORMATS:                                          *
                 *                                                                   *
                 *      IN THE FILE FORMATS BELOW, A(2) IS THE UNSPEC OF HALF-WORD   *
                 *      FIXED BINARY AND A(4) IS THE UNSPEC OF FULL-WORD FIXED       *
                 *      BINARY.  THE HEADER RECORDS ARE IN CHARACTER FORM AND        *
                 *      CONSIST OF 'TOKEN' (5 BYTES), TEXT ID (80 BYTES), MAXIMUM    *
                 *      TOKEN LENGTH (3 BYTES), NUMBER OF TOKENS (6 BYTES), EIGHT    *
                 *      NUMBERS (5 BYTES EACH) GIVING THE NUMBER OF OCCURRENCES      *
                 *      OF EACH CATEGORY.                                            *
                 *                                                                   *
                 *        AS INPUT     HEADER RECORD OF 134 BYTES.                   *
                 *                     TOKEN RECORDS IN ALPHA ORDER BY TOKEN.        *
                 *                     LINEAR TEXT-ORDER NO.      A(4)               *
                 *                     TOKEN LENGTH               A(2)               *
                 *                     TOKEN                      A(TLEN)   *FIXED*  *
                 *                     8-BIT STRING               A(1)               *
                 *                     POINTER TO TYPE RECORD     A(2)               *
                 *                                                                   *
                 *        AS OUTPUT    HEADER RECORD OF 134 BYTES.                   *
                 *                     TOKEN RECORDS IN ORIGINAL TEXT ORDER.         *
                 *                     POINTER TO TYPE RECORD     A(2)               *
                 *                     8-BIT STRING               A(1)               *
                 *                     TOKEN LENGTH               A(2)               *
                 *                     TOKEN                      A(TOKEN LENGTH)    *
                 *                                                                   *
                 *********************************************************************/
  2     1       DCL IHESRTD ENTRY(CHAR(30),CHAR(42),FIXED BINARY(31,0),
                   FIXED BINARY(31,0),ENTRY,ENTRY),
                   IHESARC ENTRY(FIXED BINARY(31,0));
  3     1       DCL P1 PICTURE '999';
  4     1       DCL SFLD CHAR(30), RFIELD CHAR(42), RETCDE FIXED BINARY(31,0);
  5     1       DCL E15 RETURNS(CHAR(TLEN+9));
  6     1       DCL SIZE FIXED BINARY(31,0), TLEN FIXED BINARY;
  7     1       DCL HD CHAR(134);
  8     1       DCL ENTREC CHAR(TLEN+9) DEF ENTRECD,
                  1 ENTRECD UNALIGNED,
                    2 LIN CHAR(4),
                    2 LEN CHAR(2),
                    2 REST CHAR(TLEN),
                    2 BOB BIT(8),
                    2 LL CHAR(2);
  9     1       DCL B CHAR(1);
 10     1         GET FILE(TOKEN) EDIT(HD)(A(134));    /* READ TOKEN FILE HEADER */
 11     1         P1=TLEN+9;
 12     1         SFLD=' SORT FIELDS=(1,4,CH,A) ';     /* MAKE UP SORT CONTROL */
 13     1         RFIELD=' RECORD TYPE=F,LENGTH=('||P1||') ';
 14     1         II=0;
 15     1         ON ENDFILE(TOKEN) II=1;    /* SET ENDFILE SWITCH */
```

```
STMT LEVEL NEST

 17      1            CALL IHESRTD(SELD,RETELD,SIZE,RETCDE,E15,E35);
 18      1            IF RETCDE=16 THEN DO;
 20      1    1          PUT SKIP LIST('SORT2 WAS UNSUCCESSFUL');
 21      1    1          STOP;  END;
 23      1            E15: PROC RETURNS (CHAR(TLEN+9));  /* PASS TOKENS TO SORT */
 24      2                 DO N=1 BY 1;
 25      2    1              GET FILE(TOKEN) EDIT(ENTREC)(A(TLEN+9));
 26      2    1              IF LL=1 THEN GO TO LASTOK; /* LL=1 ON ENDFILE TOKEN */
 28      2    1              CALL IHESARC(12);
 29      2    1              RETURN(ENTREC);
 30      2    1             END;
 31      2                  LASTOK:
                            CLOSE FILE(TOKEN);
 32      2                  PUT FILE(TOKEN) EDIT(HD)(A(134));     /* WRITE TOKEN FILE HEADER */
 33      2                  CALL IHESARC(8);
 34      2                  RETURN;
 35      2            END E15;
 36      1            E35: PROC(OUTREC);
 37      2                 DCL OUTREC CHAR(TLEN+9),
                               1 OUTRECD UNALIGNED DEF OUTREC,
                                  2 LTN CHAR(4),
                                  2 LEN CHAR(2),
                                  2 REST CHAR(TLEN),
                                  2 BOB BIT(8),
                                  2 LL CHAR(2);
 38      2                  UNSPEC(B)=UNSPEC(BOB);
 39      2                  PUT FILE(TOKEN) EDIT(LL,B,LEN,REST)(A(2),A(1),A(2),A(UNSPEC(LEN)));
 40      2                  CALL IHESARC(4);
 41      2                  RETURN;
 42      2             END E35;
 43      1           END SORT2;
```

```
STMT LEVEL NEST
                    /* LIST - PROCEDURE TO PRODUCE PARTIAL LISTINGS OF GENERATED FILES */
  1                 LIST: PROC(TLEN);
  2      1          DCL                  LEN CHAR(2), TOK CHAR(300) VAR, PTR CHAR(4),
                        FREQ CHAR(2);
  3      1          DCL HD CHAR(134);
  4      1          DCL TLEN FIXED BINARY;
  5      1          DCL N FIXED BINARY(31,0);
  6      1          DCL PT CHAR(2);
  7      1          DCL BOB PTR(8), P CHAR(1);
  8      1          PUT PAGE;
  9      1          OPEN FILE(GLOSS);       OPEN FILE(INDTAB);
 11      1          OPEN FILE(TOKEN);  OPEN FILE(TYPE);
 13      1          GET FILE(TOKEN) EDIT(HD)(A(134));
 14      1          PUT EDIT('TOKEN FILE',HD)(A,SKIP,A);
 15      1          ON ENDFILE(TOKEN) GO TO PTYPE;
 17      1            DO I=1 BY 1 TO 1000;
 18      1    1         GET FILE(TOKEN) EDIT(PT,P,LEN)(A(2),A(1),A(2));
 19      1    1         J=UNSPEC(LEN);
 20      1    1         GET FILE(TOKEN) EDIT(TOK)(A(J));
 21      1    1         BOB=UNSPEC(B);
 22      1    1         *=UNSPEC(PT);
 23      1    1         PUT SKIP EDIT(*,BOB,J,TOK)(P(6),X(1),B(8),X(1),F(3),X(1),A(J));
 24      1    1       END;
 25      1          PTYPE: PUT PAGE;
 26      1            ON ENDFILE(TYPE) GO TO GLOS;
 28      1            GET FILE(TYPE)  EDIT(HD)(A(134));
 29      1            PUT EDIT('TYPE FILE',HD)(A,SKIP,A);
 30      1            DO I=1 BY 1 TO 1000;
 31      1    1         GET FILE(TYPE) EDIT(LEN)(A(2));
 32      1    1         UNSPEC(J)=UNSPEC(LEN);
 33      1    1         GET FILE(TYPE) EDIT(TOK,FREQ)(A(J),A(2));
 34      1    1         UNSPEC(K)=UNSPEC(FREQ);
 35      1    1         PUT SKIP EDIT(J,TOK,K)(P(3),X(1),A(J),X(1),P(6));
 36      1    1           DO L=1 TO K;
 37      1    2             GET FILE(TYPE) EDIT(PTR)(A(4));
 38      1    2             UNSPEC(N)=UNSPEC(PTR);
 39      1    2             PUT EDIT(N)(X(1),P(6));
 40      1    2           END;
 41      1    1       END;
 42      1          GLOS: PUT PAGE;
 43      1          GET FILE(GLOSS) EDIT(HD)(A(134));
 44      1          PUT EDIT('GLOSS FILE',HD)(A,SKIP,A);
 45      1          ON ENDFILE(GLOSS) GO TO INDEX;
 47      1          DO I=1 BY 1 TO 1000;
 48      1    1       GET FILE(GLOSS) EDIT(LEN)(A(2));
 49      1    1       J=UNSPEC(LEN);
 50      1    1       GET FILE(GLOSS) EDIT(TOK,FREQ)(A(J),A(2));
 51      1    1       K=UNSPEC(FREQ);
 52      1    1       PUT SKIP EDIT(J,TOK,K)(P(3),X(1),A(J),X(1),P(6));
 53      1    1     END;
 54      1          INDEX: PUT PAG ;
 55      1          GET FILE(INDTAB) EDIT(HD)(A(134));
 56      1          PUT EDIT('INDEX FILE',HD)(A,SKIP,A);
 57      1          ON ENDFILE(INDTAB) GO TO FREM;
 59      1          DO I=1 BY 1 TO 1000;
 60      1    1       GET FILE(INDTAB) EDIT(II,PTR)(P(1),A(4));
```

```
        /* LIST - PROCEDURE TO PRODUCE PARTIAL LISTING OF GENERATED FILES */

                                                                                    163
STMT LEVEL NEST

 61      1    1       N=UNSPEC(PTR);
 62      1    1       PUT SKIP EDIT(II,N)(F(1),X(1),F(6));
 63      1    1     END;
 64      1          TERM: END LIST;
```